On Computer-Aided Methods for Modeling and Analysis of Organizations

Cover illustration: Pieter Bruegel der Ältere “Turmbau zu Babel”, Kunsthistorisches Museum, Wien

SIKS Dissertation Series No. 2008-02

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

VRIJE UNIVERSITEIT

On Computer-Aided Methods for Modeling and Analysis of Organizations

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op donderdag 10 januari 2008 om 10.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Olexiy Albertovitsj Sharpanskykh

geboren te Zaporizhzhya, Oekraïne

promotor: prof.dr. J. Treur

Моим родителям

Preface

The dissertation has been finished. It gives a good feeling to realize that three and a half years of persistent work have resulted into the holistic (hopefully useful) outcome. I am glad that I had the opportunity to work on and to finish this dissertation successfully at the Vrije Universiteit Amsterdam, in the Agent Systems Research Group. First of all, I am grateful for that to my supervisor Jan Treur and to Catholijn Jonker, who recognized the potential in me and gave me the chance to prove myself in pursuing research. Hope that their expectations have been justified.

Before coming to Amsterdam my major background was in the Computer Science area. However, during the last years of my university studies in Ukraine I had a particular interest in doing multidisciplinary research, on the edges of exact and humanitarian/natural sciences. This interest influenced the choice of the topic for my Master thesis – approaches and techniques for adaptive distance learning systems. Luckily, after receiving my Master I have got the opportunity to continue multidisciplinary research in Amsterdam. The major topic of the projects in which I was involved from the beginning was formal agent-based modeling and analysis of organizations. Although I possessed knowledge on formal methods, the area of organization modeling was new to me. Many fruitful discussions that we had in our group during my first year contributed to my quick acquaintance with the agent-based organization modeling domain and the related techniques. However, since our projects required modeling of real (human) organizations, the existing approaches from the area of agent systems that aimed mainly at improving the performance of artificial organizations, were not sufficient for our purposes. During my PhD period I performed investigations also in other areas that study different aspects of organizations, such as Social Science, Enterprise Information Systems, Computational Organization Theory, Social Simulations… Methods and techniques of each of these areas have their own advantages and drawbacks. Moreover, from my observations, sometimes weaknesses of methods in one area were addressed in another area, and vice versa. However, I also observed that currently not much interaction between different areas exists. Often methods and techniques are restricted to a particular scientific school and tradition. This situation in the conditions of the increasing complexity of organizations can be metaphorically compared to the building of the legendary Tower of Babel, the image of which I used for the cover. This metaphor is also applicable to the process of organization modeling. Many modern organizations

are characterized by high behavioral and structural complexities, involving many parties with diverse goals performing for a wide range of tasks. Thus, mistakes, inconsistencies and performance bottlenecks are not rare in modern organizations. Some of these organizational flaws may result into less ideal performance, whereas others can seriously affect the vitality of an organization. This dissertation describes formally grounded methods for analysis of organizations based on complex structural and behavioral relations that exist between different organizational aspects.

Although I had sufficient freedom to make choices and to realize my ideas in the research, many parts of this work have resulted from the collaboration. In the Netherlands I really learned the value of teamwork, which is also crucial in modern organizations. Particularly, I would like to acknowledge Jan in collaboration with whom the formal foundations of the dissertation were developed; Catholijn and pInar with whom I started our organization modeling activities; Viara with whom we had very nice and very productive collaboration during my last year. Further, I would like to acknowledge my (former and current) colleagues with many of whom I collaborated in a number of projects: Annerieke, Charlotte, Egon, Fiemke, Ghazanfar, Lai, Lourens, Mark, Martijn, Matthijs, Michel, Peter-Paul, Radu, Savas, Tibor, Vera, Waqar and Zulfiqar. With many of these people I developed very good personal relations, which I value a lot. I am glad we undertook many trips to conferences and activities beyond the work.

Also, I would like to acknowledge the partners within the projects, in which I was involved: CIM (Cybernetic Incident Management), DEAL (Distributed Engine for Advanced Logistics), and CARE INO III (in particular, Sybert Stroeve and Henk Blom from the NLR Air Transport Safety Institute).

I am very grateful to the sponsors who not only allowed me to accomplish this research successfully, but also provided means to travel to many conferences around the world: the Dutch Ministry of Economical Affairs, the Netherlands Organisation for Scientific Research (NWO), and the European Organization for the Safety of Air Navigation. This was particularly important to me, since getting in touch with the research community, presenting my research results and receiving feedback always gave me a strong impulse to continue investigations and provided grounds for new ideas.

Further, I want to thank the members of the reading committee: Olivier Boissier, Jaap Boonstra, Kathleen Carley, Catholijn Jonker, Michael Petit, and Stefan Schlobach for the time they spent on the reading of this dissertation and for their encouraging comments.

Work is a significant part of life, but life is not confined to work only. The Netherlands became the country, in which I began to live completely independently, building everything from nothing. I am glad I met nice people here, with whom I hope we shall stay in touch for a long time. I want to thank Paula for the opportunity to live in her house in a very nice and quite neighborhood in Amstelveen. It was very nice to spend my leisure time with Nicholas, Jurie, Elizabeth, Maria, Edgard, Tatiana, Ton, Vadim, Maxim, Boris, Dmitry, Marius and Katerina. Taking a distance from the Slavic cultural tradition and getting a broader view over the West European cultural and historical tradition helped me to develop a deeper understanding of many important things…

I spent very nice time in Germany with my old and new German friends. Ich danke allen - Alexander, Susanne, Wolfgang, Christian und Gereon, Marlies und Hubert, Magda und Waldemar, Rita und Wolfgang, Beata und Dietmar, Maria und Johann, Ralf, Ingrid und Alois, Elisabeth - für die warmherzige Gastfreundschaft und für alles, was wir zusammen unternommen und erlebt haben. Durch Euch alle habe ich mich in Deutschland immer wieder wie zuhause gefühlt. Further many warm greetings to Helene, Rodney, Mike, Frederick and Jerome.

Of course, my family and friends in Ukraine and Russia play a special role in my life. Few words to them in Russian: Отдельные слова благодарности за поддержку и веру моей семье и друзьям. Это большое счастье, что Вы есть! Я осознаю это каждую минуту и ценю.

October 2007
Rome – Amsterdam

Table of Contents

Part I

Introduction and Related Work

Chapter 1

Introduction

The modern world is unthinkable without organizations. The rapid scientific, societal and technological development of the last centuries, coupled with the changed environmental conditions gave rise to a great diversity of organizational forms and types of interaction between them. The structural and behavioral complexity of organizations is interdependent on the complexity of the environment, in which these organizations are situated. The complex, dynamically changing environment with insufficient resources often creates challenging obstacles for the satisfaction of the primary goals of any organization – to survive and to prosper. To be successful an organization should effectively and efficiently organize its internal structure and activities, in such a way that the fit with the environment is achieved. In reality these requirements are difficult to fulfill, since no universally applicable recipes exist that ensure the successfulness of an organization in all times and in all cases [27]. Therefore, most modern organizations suffer from different performance inefficiencies, inconsistencies and bottlenecks that may have (serious) consequences for the organizational vitality. Often only a small number of these flaws can be easily identified (e.g., evident inconsistencies and mistakes), whereas latent organizational flaws can be revealed using more profound (formal) analysis methods. To enable such formal analysis, this thesis introduces formal modeling approaches that allow representing diverse aspects of organizational reality. For verification and validation of organizational specifications developed using the proposed modeling techniques this thesis contributes a set of automated formal analysis techniques. Such techniques aim at identifying inconsistencies and performance bottlenecks both in the organizational structure and behavior. The proposed modeling and analysis techniques form a basis for a general organization modeling and analysis framework, which is described in this thesis.

In this chapter first the motivation for the research is presented in Section 1.1. Then, the research goals are described in Section 1.2. Section 1.3 discusses the professional significance of the study. The research methodology is described in

Section 1.4. An overview of the proposed framework is provided in Section 1.5. The delimitations of the study are given in Section 1.6. Definitions of key terms are given in Section 1.7. Finally, an overview of the thesis is given in Section 1.8.

1.1 Motivation

In the areas of management and organization theory a range of theories, guidelines and best practices concerning the design and management of effective and efficient organizations has been developed [10, 20, 22, 23, 25, 27]. However, most of these theoretical findings are specified in an informal, imprecise and ambiguous way. Consequently, such theories and guidelines are difficult to apply in practice. One of the attempts to identify concrete, practically applicable recommendations for designing and managing an organization situated in a particular type of environment has been undertaken within the framework of the contingency theory [10, 27]. The key thesis of the contingency theory is that to ensure the effectiveness and the efficiency of an organization, its structure and behavior should be defined depending on particular environmental characteristics [27]. Although the contingency theory provides many useful insights into the principles on which the organizational structure and dynamics are based, still these principles are formulated at a high level of abstraction. To achieve high organizational performance these principles should be carefully adapted in the context of particular organizational settings. In the process of adaptation inconsistencies and inefficiencies may by introduced that cannot be foreseen and identified by the contingency theory. To identify these inconsistencies and inefficiencies analysis techniques are required.

Many of the techniques for analysis of organizational performance developed in organization theory are informal and imprecise (cf. [4, 26]), which undermines the feasibility and the rigor of the analysis results. For more precise evaluation of the organizational performance, for identification of performance bottlenecks and organizational conflicts, and for estimation (prediction) of consequences of different environmental influences, organization structures and behaviors on organizational performance, detailed organizational analysis based on a formal organization model should be performed. Furthermore, a formal organization model constitutes a basis for many automated processes within enterprises (e.g., computer integrated manufacturing [6], production management [11]) and provides a foundation for inter-enterprise cooperation. Approaches for formal organization modeling and analysis have been developed in a number of areas, which may be divided into two groups: traditional and agent-based organization modeling methods.

Traditional organization modeling methods

The first formal computational organization modeling approaches have been developed in the areas of the system dynamics theory [14] and operation research [21]. Organizational models specified in system dynamics are based on numerical variables and equations that describe how these variables change over time. Operation research proposes mathematical methods for identifying best possible solutions to problems related to coordination and execution of the operations within an organization that improve or optimize the organizational performance [21]. Both

system dynamics- and operation research-based modeling approaches abstract from single events, entities and actors of organizations and take an aggregate view on the organizational dynamics. Such methods may be useful for the analysis of the organizational dynamics at macro levels (e.g., market fluctuations, general trends of the organizational development), however, provide little help for the investigation of the behavior of organizational individuals and (the dynamics of) relations between them at (the most) detailed levels. As a high complexity of social dynamics results from a large number of diverse local interactions among organizational actors, the examination of an organization at the most detailed level may help identifying causes of organizational malfunctioning and inefficiencies at more general levels.

The agent-based approaches to complex systems have been used for modeling and analysis of both human and artificial organizations.

Agent-based organization modeling

Agent-based modeling approaches take into account the local perspective of a possibly large number of separate components and their specific behaviors (i.e., interactions) in a system. The concept of an agent may be used to model both human beings as well as hardware and software components of socio-technical systems such as organizations. Currently many definitions exist for the concept of an agent. One of them is given in [15]:

> An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.

In the area of Multi-Agent Systems the organizational paradigm is used to improve computational properties of distributed algorithms, based on agent systems [3, 13, 18, 24]. In particular, representation of a multi agent system as an organization consisting of roles and groups can facilitate handling high complexity and poor predictability of the dynamics in a system, and thus, allows building better algorithms [19]. Organization-oriented models of multi-agent systems have been also used to support processes of real organizations: goal-related aspects of organizations are considered in [17], models for different types of organizational interactions are described in [19], an approach to coordinate the execution of tasks is shown in [9]. Although organization-oriented models developed in the area of multi-agent systems can be computationally effective for particular tasks, those known to the author lack the ontological expressivity required to conceptualize a wide range of concepts and relations of human organizations. Furthermore, such frameworks only rarely use the extensive theoretical basis from Organization Theory. Many agent-based approaches that incorporate findings from Social Science are considered in the area of computational organization theory.

In the area of computational organization theory [5] computational and mathematical techniques are applied for the investigation of human organizations; development, testing and improvement of organization theories. Many models in this area are based on the agent paradigm. Such models aim at the representation, investigation and prediction of processes in organizations considered at three representation levels: (1) the macro level that focuses on an organization as a whole and its relations with the environment (e.g., other organizations, markets); (2) the

meso level that focuses on the interaction between individuals and/or groups in the organizational context; and (3) the micro level that focuses on an individual of an organization, his/her characteristics and behavior in an organization. Organizational factors that exert an influence on the behavior of agents are diverse: norms and regulations related to the task execution and to communication, a power (authority) system, a reward/punishment system. Furthermore, organizational factors are interrelated (e.g., a power structure influences the execution of tasks). However, often models used in computational organization theory consider only a limited number of the organizational aspects directly related to the considered research problem and do not reveal (inter-) dependencies that exist between these and other (indirectly related) organizational aspects. Neglecting indirect relations between aspects may result into limited evaluation possibilities of different organizational processes and may undermine the practical feasibility of organizational models.

To perform a profound evaluation of the organizational performance and to enable analysis and prediction of organizational behavior under different environmental influences, more sophisticated modeling and analysis techniques are required that employ concepts and relations between them across different perspectives on organizations and establish relations between different representation levels (i.e., micro, meso and macro).

1.2 Research objectives

The main research objective of the thesis is to develop expressive scalable agent-based formal methods for modeling structures and processes of organizations of most types and the related techniques for elaborated manifold computational analysis of organizational specifications. The modeling methods should be able to represent both predefined formal organizational structures of diverse types and arbitrary behavior of autonomous organizational actors. Furthermore, these methods should allow explicit representation of relations that exist between different aspects of organizational reality. Also, dedicated analysis techniques that make use of these relations should be developed.

To enable reliable analysis of organizational specifications, the formal foundations (for modeling languages and analysis techniques) should be defined. Furthermore, to be feasible and applicable in practice the developed techniques should be supported by theoretical findings and empirical evidences from Social Science (in particular, organization theory and Psychology). Also, the developed techniques should be integrated into a general framework for organization modeling and analysis. It is intended to apply within the framework automated formal methods together with (to a great extent) informal social theories to verify and validate organizational specifications, and to gain a better understanding of general principles, on which the structure and dynamics of different forms of organizations are based.

Another objective of the thesis is the identification of methodological guidelines for the development of organizational specifications. Furthermore, the proposed modeling and analysis techniques are required to be implemented. Also, a preliminary validation and evaluation of the proposed framework should be performed.

1.3 Significance

The proposed organization modeling and analysis techniques developed as a part of this dissertation are based on formal many-sorted predicate languages. The formal modeling of organizations enables more precise, sophisticated and rigorous types of analysis in comparison to informal analysis methods proposed in Social Science [16, 23, 25]. Formal organizational specifications expressed using languages as proposed may describe complex processes (e.g., adaptation) in organizations by complex logical formulae, which cannot be directly used for automated analysis (simulation, verification). To enable automated analysis of organizational behavior, the dissertation proposes an automated procedure for transformation of behavioral specifications of dynamic systems (e.g., of an organization) into executable format that allows formal analysis and execution.

Furthermore, to reduce the complexity of organization modeling and analysis and to increase the scalability of organizational specifications, a formal approach has been developed using which organizations can be modeled and analyzed at different aggregation levels. To ensure the integrity of a complete organizational specification, the approach provides means to establish and to prove relations between different aggregation levels. Moreover, the proposed formal techniques are based on the findings from Social Science, which allow designing and analyzing realistic organization models.

Generalized organization modeling and analysis with explicit identification of relations between different aspects of organizations considered in the dissertation has a number of advantages compared with a variety of specialized modeling approaches for particular organizational aspects:

(1) a unified set of concepts and relations between them is used for modeling different types of organizations;
(2) the relations between the concepts from different perspectives (e.g., goal-oriented, process-oriented, human-oriented) are explicitly defined;
(3) computational analysis can be performed across multiple views;
(4) indirect relations between different organizational and environmental characteristics and processes can be investigated.

Nowadays a number of highly expressive enterprise modeling frameworks exist. However, the frameworks known to the author either do not have formal foundations [2, 6] or have limited verification possibilities [2]. This thesis proposes diverse formal verification and validation means for analyzing expressive organizational specifications. Such analysis means can be applied by organizational designers and managers for the evaluation and improvement of different aspects of an organizational structure and dynamics, for the estimation of organizational performance.

1.4 Research Methodology

As a point of departure for this research, based on the analysis of theoretical and practical findings from Social Science a number of paradigmatic types of modern organizations have been identified and investigated. Based on the results of these investigations coupled with the needs of practitioners identified from the managerial

literature, during the project meetings and workshops, a set of requirements for computational formal methods for modeling and analysis of organizations of diverse types have been identified. Useful contributions to this set were provided during the meetings on the projects CIM (Cybernetic Incident Management), DEAL (Distributed Engine for Advanced Logistics), and CARE INO III (from the air traffic management domain). The most important high level requirements are the following:

(1) e*xpressivity*: the modeling methods should provide languages with sufficient expressivity to represent different structural and behavioral aspects of organizations of different types; furthermore, relations between different aspects of the organizational reality should be specified explicitly;
(2) *a strong connection to Social Science*: the meaning attached to the introduced modeling concepts and the rules of correct use of these concepts in organizational specifications should be specified based on the literature from Social Science;
(3) *automated formal analysis:* the language used for the formalization of organizational specifications should also allow rigorous automated formal analysis of these specifications (e.g., by simulation, verification and validation) both within particular views on organizations and across multiple views;
(4) *complexity*: since specifications for real organizations may be very complex, means to handle a high complexity and to increase scalability of modeling and analysis should be identified;
(5) *support for the execution of organizational scenarios*: the developed methods should allow designing organizational specifications that form a basis for enterprise information systems, which support and control the execution of organizational scenarios;
(6) *usability*: the framework based on the developed techniques should be usable and convenient for organizational practitioners: modelers, designers, analysts, etc.

Thereafter, many existing organization and enterprise modeling and analysis frameworks and techniques have been evaluated against the identified requirements. To name a few from the area of multi-agent systems: Gaia, SODA, AGR, DESIRE, AAII, MOISE+, TROPOS, OperA, ISLANDER, OMNI, TAEMS; from the area of enterprise information systems: CIMOSA, GRAI, Zachman, ARIS, TOVE, GERAM, PERA. The results of the study indicate that none of the investigated frameworks satisfied the requirements completely. On the one hand, some of the enterprise modeling frameworks developed in the area of Enterprise Information Systems (e.g., CIMOSA, GRAI) provide considerable ontological expressivity, however, do not have any formal foundations. Further, even the frameworks based on formal modeling languages (e.g., TOVE, Zachman) provide only limited organization analysis means. On the other hand, (computational) formal frameworks developed in the areas of Artificial Intelligence and computational organization theory known to the author have a limited expressivity, focusing on particular aspects of organizations abstracting from the structural and behavioral complexity of organizations. In particular, the ISLANDER [12] framework focuses primarily on normative aspects of an

organization, whereas process-related and intentional aspects are not elaborated much. On the contrary, the framework TAEMS [9] addresses detaily the execution of tasks in an organization; however, only a little attention is devoted to different types of relations between the organizational roles in this framework.

The development of formal organization modeling and analysis methods that satisfy the identified requirements began from the identification of the organizational perspectives with the relevant concepts and relations based on the input from Social Science and Enterprise Information Systems. The fulfillment of this step contributed to the satisfaction of the requirements (1) and (2). The identified modeling perspectives are similar to the ones recommended by the Generalized Enterprise Reference Architecture and Methodology (GERAM) [2], which is often used as a basis for comparison of the existing enterprise modeling frameworks and serves as a template for the development of new ones. The identification of particular modeling perspectives on organizations with their languages and techniques also decreases the complexity of modeling and analysis, thus, contributing to the satisfaction of the requirement (4).

Then, the choice of a formal apparatus used for the formalization of the methods being developed has been performed. A formal language used for the formalization of a particular view was required to be expressive enough to represent different (quantitative and qualitative) structural and behavioral aspects of this view. Furthermore, to satisfy the requirement (3) such a language should allow performing different types of computational analysis (e.g., by simulation, verification and validation). Moreover, to enable analysis across different interrelated views, the formal languages of these views should be syntactically and semantically compatible with each other.

Sorted predicate logic restricted to finite sorts has been chosen as a formal basis for defining dedicated modeling languages for each view. To express temporal relations in specifications of the views, the dedicated languages of the views are embedded into the Temporal Trace Language (TTL) [28], which is a variant of reified order-sorted temporal predicate logic. To enable automated formal analysis on organizational specifications the formal syntax and semantics of TTL have been defined. Furthermore, to enable different particular types of analysis several executable sublanguages of TTL have been defined.

The predicate-based ontologies used for the formalization of the views are intuitive, close to natural language. These ontologies share a terminological basis with the one used by practitioners (e.g., organizational managers, analysts), which contributes to the satisfaction of requirement (6). Furthermore, the concepts and relations of these languages can be represented graphically. The graphical interface has been implemented for the performance-oriented view, whereas other views are specified textually using dedicated modeling tools.

Within every view a set of structural and behavioral *constraints* imposed on the specifications of the view can be identified. Some of these constraints are formulated based on the theoretical findings from Social Science. An example is the constraint expressing the transitivity of a “subordinate-superior” relation in an organizational authority structure. Other constraints are domain-dependant (e.g., based on the formal regulations of a particular organization). The algorithms for verification of

satisfaction of different types of constraints with respect to specifications of different views have been developed and implemented.

To reduce the complexity of modeling and analysis (the requirement (4)) and to support the scalability of organizational specifications, the means for specifying, depicting and analyzing organizational specifications at different aggregation levels have been developed. For example, the specification of an organizational structure and dynamics can be performed at the level of departments or at the level of roles within a particular department. Furthermore, organizational processes can be considered with different level of details. To ensure the consistency of a whole organizational specification the (structural and dynamic) relations between different aggregation levels of an organizational representation should be established and their correctness should be formally proved. An approach for automated formal verification of such relations is described in this thesis.

The dedicated modeling and analysis techniques have been integrated into a general framework. Organization specifications developed using the proposed framework can be also used as a basis for enterprise information systems that guide and execute organizational scenarios (the requirement (5)). Examples of the application of the framework for this purpose are considered in the thesis.

For the development (designing) of organizational specifications for both new and existing organizations using the proposed framework, methodological guidelines are provided. These guidelines comprise the general guidelines for designing organizational specifications over multiple views, and the guidelines for every particular view.

The evaluation and validation of the proposed framework have been performed in the context of three research projects from the domains of transport logistics, of incident management and of air traffic control. The investigated organizations combined features of mechanistic organizations (e.g., a functional specialization, a high level of task formalization, a hierarchical structure of managerial positions) with some features of organic organizations inherent in some structural units (e.g., a flat power structure, informal work style, adaptivity and flexibility of the task execution). In all these projects organizational specifications have been built using the proposed modeling framework representing (parts of) real organizations. During the organizational modeling process different types of formal and informal organizational knowledge have been used. In particular, formal knowledge was obtained from different organizational documents (e.g., charts, job descriptions, procedures, regulations, norms, reports), available descriptions of similar organizations from the same domain. Informal knowledge also plays an important role in creating feasible organizational specifications. Such knowledge was obtained from interviews, questionnaires, and brainstorming sessions. Then, the constructed specifications have been investigated using different types of the developed analysis techniques (a brief overview of the proposed analysis techniques is given in Section 1.5). The performed analysis aimed at establishing the consistency of organizational specifications and at inspecting and improving of efficiency and effectiveness of the organization operation by identifying inconsistencies and performance bottlenecks. The results of the automated analysis were provided for the further evaluation and consideration to organizational domain experts and managers. Major results of these investigations are described in the thesis.

1.5 Overview of the proposed modeling and analysis methods

In line with GERAM [2] four interrelated modeling views have been introduced: *The performance-oriented view* describes organizational goal structures, performance indicators structures, and relations between them. *The process-oriented view* contains information about the organizational functions and processes, how they are related, ordered and synchronized and the resources they use and produce. Within *the organization-oriented view* organizational roles, their authority, responsibility and power relations are defined. In *the agent-oriented view* different types of agents with their capabilities are identified, models of agent behavior are specified based on social theories, and principles of allocating agents to roles are formulated.

The sorted predicate logic-based languages of the views provide high expressivity for conceptualizing a variety of concepts and relations and allow expressing both quantitative and qualitative aspects of different views. To express temporal relations in specifications of the views, the dedicated languages of the views are embedded into the Temporal Trace Language (TTL). In TTL the organizational dynamics are represented by a trace, i.e. a temporally ordered sequence of states. Each state is characterized by a unique time point and a set of state properties that hold (i.e., are true). State properties are specified using the dedicated language(s) of the view(s). Temporal (or dynamic) properties are defined in TTL as transition relations between state properties.

A set of constraints imposed on a specification of a particular view is represented by a *logical theory* that consists of formulae constructed in the standard predicate logic way from the terms of the dedicated language of the view (and of TTL if temporal relations are required). Since the views are related to each other by sets of common concepts, also these concepts can be used in the constraints expressions (see Fig. 1).

The constraints are divided in two groups: (1) *generic constraints* need to be satisfied by any specification of the view; (2) *domain-specific constraints* are dictated by the application domain and may be changed by the designer. Two types of generic constraints are considered: (1) *structural integrity and consistency constraints* based on the rules of the specification composition; (2) *constraints imposed by the physical world*. Two examples of the generic constraints are the following: “not consumed resources become available after all processes are finished”, “non-sharable resources cannot be used by more than one process at the same time”. Domain-specific constraints can be imposed by the organization, external parties or the physical world of the specific application domain. Two examples of the domain-specific constraints are the following: “the amount of driving hours for each driver should not exceed 6 hours per day” (imposed by a law), “agent A has no access to resource of type rt” (imposed by the organization). The developed organization modeling tools allow defining parameterized templates (macros) for complex constraints that can be instantiated in different ways, which can also be used as support for designers not skilled in logics.

The environmental conditions in which an organization operates can be represented by both generic and domain-specific constraints. Furthermore, for particular purposes (e.g., for simulation of particular scenarios) an environment can be represented by a separate modeling component. In this case either an aggregated view on the

environment may be taken (e.g., to represent the global behavior of markets) or a more elaborated specification of the environment may be created. In the latter case, the internal specification for the environment can be specified using one of the existing world ontologies (e.g., CYC, SUMO, TOVE). It can be also defined by a set of objects with certain properties and states and with causal relations between objects. In general no particular restrictions on the representation of the environment are imposed by the framework.

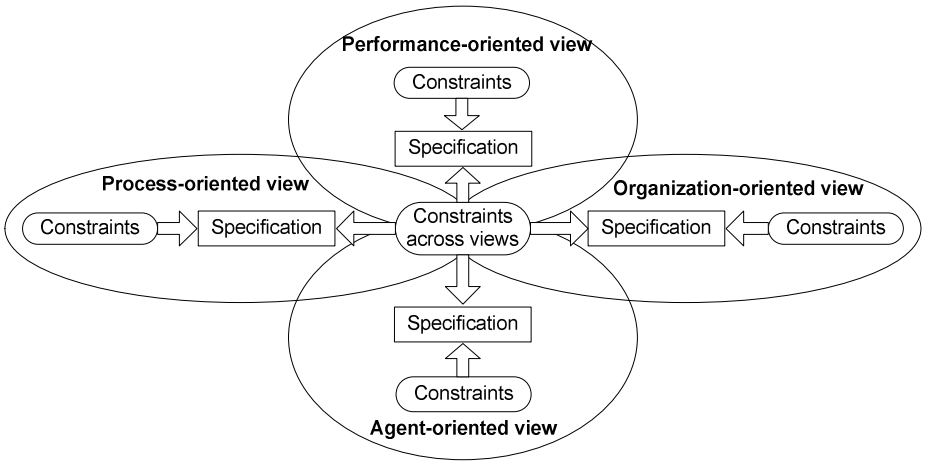


Fig. 1. Modeling an organization using the proposed framework. Each arrow denotes that a set of constraints is imposed on a specification.

A specification of the view is *correct* if the corresponding theory *T* of constraints is satisfied by this specification, i.e., all sentences in *T* are true in the many-sorted first-order structure(s) corresponding to the specification. Thus, the specification defines a set of models of the theory *T*.

The proposed analysis techniques will be described in the following.

The first analysis type focuses on the verification of specifications of every view, i.e., establishing the correctness of a specification of a view with respects to a set of constraints defined in this view. Furthermore, some constraints may be specified across views, i.e., using concepts and relations of several views. In this case the satisfaction of such constraints is checked with respect to a combined organizational specification that comprises the specifications of these views. The algorithms developed for the verification of constraints of different types in the proposed framework are more efficient than general-purpose methods for verifying specifications (e.g., model checking [7]).

A specification of a view can be represented and analyzed at different aggregation levels. In general, modeling and analyzing an organizational specification of a particular aggregation level is computationally much cheaper than dealing with the whole detailed organizational specification. However, to guarantee the consistency of a complete organization specification that comprises the specifications of different aggregation levels, structural and behavioral relations between these levels should be identified and formally proven. For the formal verification of relations between

different aggregation levels an approach based on model checking [7] has been developed and automated.

Based on correct (combined) specifications of the views further analysis of some properties of interest can be performed. Such properties are usually not implied by an organizational specification; they may be checked with the aim to optimize the organizational operation by discovering and eliminating bottlenecks, to test hypotheses on the organizational behavior under different circumstances, or to investigate theories from Social Science. If such properties are required to hold with respect to all possible executions of an organizational specification, then model checking techniques may be used for the verification. If the satisfaction of properties is required to be established with respect to a limited set of organizational executions (e.g., in some scenario(s) under certain environmental conditions), then simulation is performed. For the simulation of different scenarios of organizational behavior with agents allocated to organizational roles a dedicated tool is used. By performing simulation this tool generates a trace. Then, properties of interest formalized in TTL can be checked on a simulation trace using another dedicated checking tool called TTL Checker.

Correct organizational specifications can be also used to guide and control the actual execution of processes in organizations. The execution data recorded by an enterprise information system and structured in the form of an execution trace can be checked for conformity to a formal organization (i.e., specifications and constraints defined in particular views). To this end, the relations and constraints specified for particular views are translated into properties expressed in the execution language used for the formalization of the trace. They are checked in real time on the trace.

A trace can also be analyzed after the execution of an organizational scenario is completed. For this type of analysis, next to the properties obtained from the formal organization, the designer may specify in TTL and check other properties.

The introduced techniques can be used for modeling and analysis of structures and dynamics of organizations of different types. In particular, they allow modeling mechanistic organizations that represent systems of hierarchically linked job positions with clear responsibilities that operate in a relatively stable (possibly complex) environment. At the same time the techniques can be applied for modeling and analysis of organic organizations characterized by highly dynamic, constantly changing, organic structure with non-linear behavior. Although the structure and behavioral rules for organic organizations can be hardly identified and formalized, nevertheless by defining a limited number of (temporary) constraints and by performing agent-based simulations with changing characteristics of proactive agents, useful insights into the functioning of such organizations can be gained (e.g., how different organizational and environmental factors influence the work motivation and performance of employees). Furthermore, the techniques allow reuse of parts of models constructed within particular organizational views.

1.6 Research delimitations

Using the terminology of computational organization theory the main focus of this research is on the micro and meso levels of the organization representation and

relations between them. Although interaction between an organization and the environment in which it is situated is modeled at the macro level in this research, still a thorough investigation of macro level processes (e.g., different aspects of interorganizational cooperation) is out of scope of this thesis. Nevertheless, since all levels of the organization representation are closely related to each other, the investigation of how macro level processes influence the processes of meso and micro levels and vice versa is also a part of this research. Also, a preliminary study showed that many modeling principles and analysis techniques identified at the meso and micro levels can be also applied for the macro level.

Another delimitation of this research is related to the application of the proposed methods for automated enterprise management. In sections 1.4 and 1.5 it was indicated that organizational specifications may be used as a basis for an EIS. An example of the implementation of a specification for the process-oriented view in an EIS to guide and control the execution of organizational scenarios is described in the thesis. However, the implementation and the exploitation of an EIS built based on a complete organization model description for the automated enterprise management is out of scope of this research.

1.7 Definitions

Aggregation level of the organization's representation: a level of abstraction at which the structure and/or behavior of an organization are represented.

Constraint: an expression over organizational objects and/or processes that limits the set of all possible behaviors of an organization.

Correct organizational specification: an organizational specification of some view that satisfies all the constraints of this view defined for the organization.

Formal organization: a fixed set of rules of intra-organization procedures, regulations and structures often specified in the formal documents of the organization.

Organization: a structure that comprises sets of interrelated roles, which are intentionally organized to ensure a desired (or required) pattern of activities [1].

Organizational specification of a view: a representation of the structural and/or dynamic aspects related to a particular view of an organization using a representation language.

Perspective (view) of an organization: an organizational facet that addresses particular (closely related) aspects of an organization that may be both static (structural) and dynamic in nature.

Trace: a temporal development of processes of a system (e.g., an organization) represented in the form of an ordered time-indexed sequence of states.

1.8 Overview of the thesis

The thesis represents a collection of papers structured in six parts.

Part I provides the introduction to the thesis (Chapter 1) and an overview of the related work on modeling and analysis of organizations from different areas (Chapter 2).

Part II presents the formal theoretical foundations, on which the proposed modeling and analysis methods are based. This part introduces the syntax and the semantics of the Temporal Trace Language (TTL) used for specifying dynamic aspects of systems (e.g., organizations, biological organisms, hardware). Furthermore, in this part TTL is compared to other known formalisms. Moreover, the part describes a normal form for TTL formulae that enables different types of computational analysis of TTL specifications and a procedure for the transformation into the normal form, which creates a basis for automated analysis. Examples of the application of the transformation procedure for the purpose of simulation and analysis of agent behavioral specifications are provided in this part. Also, the part introduces another type of analysis technique and a software tool that aim at checking dynamic properties expressed in TTL on a set of simulation and/or empirical traces.

The modeling and analysis methods for the performance-oriented view, the process-oriented view, the organization-oriented view and the agent-oriented view with the related methodological guidelines and the analysis techniques are described in Part III.

The general guidelines for using the proposed modeling framework in a process of organization design, and an approach for designing specifications from the organization-oriented view are described in Part IV.

A case study from the area of the air traffic control management that demonstrates the application of the proposed framework is provided in Part V.

Finally, Part VI presents the conclusions and the directions for the future work.

References

1. Biddle, B.: Role Theory: Concepts and Research, Krieger Publishing Co. (1979)
2. Bernus, P. et al. (eds.): Handbook on Architectures of Information Systems, Springer-Verlag, Heidelberg (1998) 209-241.
3. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A.: Tropos: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agent and Multi- Agent Systems, vol. 8(3), 203-236 (2004)
4. Campbell, D.: Outcomes Assessment and the Paradox of Nonprofit Accountability. Nonprofit Management and Leadership, 12(3) (2002) 243-260.
5. Carley, K.M.: A comparison of artificial and human organizations. Journal of Economic Behavior & Organization, 31(2), 175-191 (1996)
6. CIMOSA – Open System Architecture for CIM; ESPRIT Consortium AMICE, Springer-Verlag, Berlin (1993)
7. Clarke, E.M., Grumberg, O., and Peled, D.A.: Model Checking. MIT Press (2000)
8. Dastani M, Hulstijn J, Dignum F, Meyer J-J.: Issues in Multiagent System Development. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'04), ACM (2004) 922-929

9. Decker, K.: TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. Foundations of Distributed Artificial Intelligence, Chapter 16, O'Hare, G. and Jennings, N. (eds.), Wiley Inter-Science, 429-448 (1996)
10. Donaldson, L.: The Contingency Theory of Organizations. Sage, London (2001)
11. Doumeingts, G., Vallespir, B., and Chen, D.: Decisional Modelling using the GRAI Grid, In: Bernus, P., Mertins, K. and Schmidt, G. (Eds): Handbook on Architectures of Information Systems, Springer-Verlag, 313-338 (1998)
12. Esteva, M., Rodriguez-Aguilar, J. A., Sierra, C., Garcia, P., Arcos, J. L. : On the Formal Specification of Electronic Institutions. In the book Agent-mediated Electronic Commerce: the European AgentLink Perspective, LNAI 1991, Springer-Verlag, (2001) 126-147
13. Ferber, J. and Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Y. Demazeau (ed.), Proceedings of Third International Conference on Multi-Agent Systems (ICMAS'98), IEEE Computer Society (1998) 128-135
14. Forrester, J. W.: Industrial dynamics, Waltham, MA: Pegasus Communications (1961)
15. Franklin, S., Graesser, A. Is it an agent, or just a program? A taxonomy for autonomous agents. In Mueller, J.P., Wooldridge, M., and Jennings, N.R., editors, Intelligent Agents III, Springer LNAI 1193, 21-35 (1996)
16. Galbraith, J.R.: Organization design, Addison-Wesley Publishing Company, London Amsterdam Sydney (1978)
17. Giorgini, P., Kolp, M. and Mylopoulos, J.: Multi-Agent Architectures as Organizational Structures. Journal of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers, 13(1) (2006) 3-25.
18. Hannoun, M., Boissier, O., Sichman, J.S., and Sayettat, C.: MOISE: An organizational model for multi-agent systems. In: M. C. Monard and J. S. Sichman (eds.), Proceedings of the 7th International Joint Ibero-American Conference on Artificial Intelligence (IBERAMIA'00) and 15th Brazilian Symposium on Artificial Intelligence (SBIA'00), Atibaia, Brasil, 156-165 (2000)
19. Horling, B., Lesser, V.: A Survey of multi-agent organizational paradigms. The Knowledge Engineering Review, Vol. 19(4) (2005) 281-316
20. Lorsch, J.W. and Lawrence. P.R.: Organization design, Richard D. Irwin Inc., USA (1970)
21. Marlow, W. H.: Mathematics for Operations Research. New York: Dover (1993)
22. Mintzberg, H.: The Structuring of Organizations. Prentice Hall, Englewood Cliffs (1979)
23. Morgan, G.: Images of organizations. SAGE Publications, Thousand Oaks London New Delhi (1996)
24. Omicini, A.: SODA: Societies and infrastructures in the analysis and design of agent-based systems. In: M.J. Wooldridge, P. Ciancarini (eds.), Proceedings of Agent-Oriented Software Engineering Workshop, Springer Verlag (2000)185–193
25. Pfeffer, J.: Organizations and organization theory. Pitman Books Limited, Boston London Melbourne Toronto (1982)
26. Herman, R.D., Renz, D.O.: Doing Things Right and Effectiveness in Local Nonprofit Organizations. Public Administration Review, 64:6, (2004) 694-704
27. Scott, W.R.: Organizations: rational, natural and open systems. (4nd ed). Prentice Hall International Inc., Upper Saddle River, New Jersey (1998)
28. Sharpanskykh, A., Treur, J.: Verifying Interlevel Relations within Multi-Agent Systems. In: Brewka, G., Coradeschi, S., Perini, A., and Traverso, P. (eds.), Proceedings of the 17th European Conference on Artificial Intelligence, Riva del Garda, September, IOS Press (2006) 290-294

Chapter 2

Related Work

This chapter provides a general overview of organization modeling and analysis approaches and techniques developed in four areas: organization theory (Section 1), enterprise information systems (Section 2), organization-oriented multi-agent systems (Section 3), and computational organization theory (Section 4). Note that the literature overviews related to the specific modeling views are given in the corresponding sections of the Part III.

1 Organization theory

Organization theory is a broad discipline that studies structures and dynamics of human organizations. The research methods that are used in organization theory stem from such disciplines as economics, psychology, sociology, political science, anthropology, and system theory. Related practical disciplines include human resources and industrial and organizational psychology. This literature overview focuses in particular on the major theories and trends in the western sociological tradition.

Most of the definitions for an organization in organization theory are based on the concept of rationality that lies in the basis of organizational theory [47]. One of the examples is the definition given by Giddens in [27]:

> An organization is defined as a planned, coordinated and purposeful action of human beings to construct or compile a common tangible or intangible product.

Indeed, according to this definition, organizations are created for certain purposes (or goals). To achieve these goals organizational activities are intentionally planned, coordinated and executed (e.g. using scientific methods).

In organization theory different types of organizations are distinguished. Classical organization theories [43] provide useful insights into the functioning of mechanistic organizations. This type of organizations comprises systems of hierarchically linked job positions with clear responsibilities that use standard well-understood technology and operate in a relatively stable (possibly complex) environment.

In contrast to mechanistic (or functional) organizations, a substantial group of modern organizations are characterized by a highly dynamic, constantly changing, organic structure with non-linear behavior. Such organizations (sometimes called organic organizations [44]) can be investigated using modern organization theories. Modern theories are based on two essential frameworks: the systems framework [58] and the contingency approach [17].

The systems framework is based on the notion of interdependency, which implies that a change in one part of an organization affects the behavior of all other parts. The systems framework is applied for studying matrix and network organizations [44].

The contingency approach [17] focuses on external determinates of organizational structure and behavior called contingencies. A contingency is any variable that moderates the effect of an organizational characteristic on organizational performance. The key thesis of the contingency theory is that to ensure the effectiveness and the efficiency of an organization, its structure and behavior should be defined depending on particular environmental conditions. The contingency approach is claimed to be useful for studying organizations of most of the types and it is claimed to be particularly suitable for organization design.

Organization design is a special topic in the organization theory [39, 25]. Galbraith [25] stated that 'organization design is conceived to be a decision process to bring about a coherence between the goals or purposes for which the organization exists, the patterns of division of labor and inter-unit coordination and the people who will do the work.' Further Galbraith argues that 'design is an essential process for creating organizations, which perform better than those, which arise naturally.' The ideas of Galbraith and others are used extensively in the managerial practice to (re)design efficient and effective organizations [49]. The literature on organizational design proposes an extensive set of factors identified at every level of representation of an organization (i.e., micro, meso, and macro) that influence the choice of specific design parameters (e.g., the group size, the task complexity, reporting relations, the number of employees) related to the organizational structure and dynamics.

Often three aggregation levels of the organizational representation are considered. At the individual (or *micro*) level the behavior of organizational individuals and work groups is investigated. Sometimes the study of individual behavior is separated in the disjoint from organization theory discipline – organizational behavior [35]. Among the topics that are considered at the *micro level* are the following: perceptions of an individual in the organizational context [52], work motivation and satisfaction [57], group formation [4], leadership [64], individual conflicts in organizations [40].

At the level of the whole organization (or *meso* level) different aspects of the organizational structure and dynamics are considered. At this level the following topics are of relevance: organization structure and behavior [2, 42, 44], organization authority and power structures [52, 42, 47], organization normative systems [52], intergroup conflict within an organization [40], technology in organizations [44, 52], organizational change [15].

At the global (or *macro*) level the interaction between the organization and its environment that includes other organizations, society, markets etc. is considered. At this level the behavior of organizations is investigated using the population ecology theory [30] and the resource dependence theory [48]. The following topics are considered at this level: inter-organizational formations (e.g., mergers and consolidations, joint ventures and programs) [52], governmental impact on organizations [44], organizations and politics [1], interactions between organizations and the society [52], organizations and markets [37], virtual organizations [59].

The specifications of organizations that are normally used in organization theory are represented by informal or semi-formal graphical descriptions that illustrate specific aspects of organizations [42, 52] (e.g., decision making, authority and power relations). The disadvantages of such specifications are obvious: (1) lack of generality and relations between different specific types of specifications, and (2) graphically depicted data can not be effectively processed, combined and analyzed. A class of specifications built based on the System Dynamics Theory is an exception in organization theory devoid of both these disadvantages [23]. Organizational descriptions specified in System Dynamics are based on numerical variables and equations that describe how these variables change over time. Although such specifications can be computationally effective (i.e., used for simulations and computational analysis), nevertheless they still lack the ontological expressivity needed to conceptualize a wide range of relations and phenomena that exist in different types of organizations. Furthermore, system dynamics-based modeling approaches abstract from single events, entities and actors of organizations and take an aggregate view on the organizational dynamics. Therefore, such approaches cannot be used for modeling and analyzing organizations at the micro level.

The solution to the ontological expressivity problem has been proposed in the area of enterprise information systems, considered in the following Section.

2 Enterprise information systems

In general, *an enterprise information system* is any computing system that automates the execution of certain process(es) of an enterprise. Such systems constantly collect data about the execution of various processes in manufacturing and production, finance and accounting, sales and marketing, and human resources. The collected data are used for different purposes. For example, based on these data inconsistencies and variances that may occur during the execution of processes can be identified. Furthermore, these data can be used as an input for decision making processes performed by managers. Finally, these data can be provided as input for other processes. Modern enterprises use both general and specialized information systems. General enterprise information systems are dedicated for the control and guidance of the operation of the whole enterprise. Among the specialized enterprise information systems are Material Resource Planning systems (MRP), Customer Relationship Management systems (CRM) and Supply Chain Management system [28].

Enterprise-wide information systems are often built based on enterprise architectures. An *enterprise architecture* is an enterprise-wide, integrating framework

used to represent and to manage enterprise (business) processes, information systems and personnel, so that key (strategic) goals of the enterprise are satisfied. Typically a framework for enterprise architecture includes the following aspects:

(1) modeling framework that consists of a set of modeling concepts (i.e., an ontology) and of a modeling language;
(2) modeling methodology;
(3) partial models (templates).

Enterprise architectures may address methodological issues related to the enterprise modeling. Such issues include the identification of aspects of the organizational reality that should be captured by an organization model. Usually the specific boundaries of a model are dependent upon the specific modeling goals. For example, for the analysis of processes in a supply chain, particular organizations that constitute the chain can be modeled as parts of the same organizational model. Furthermore, enterprise methodologies address a process of engineering of enterprise models. Enterprise model engineering may be expressed in the form of a process model or structured procedure with detailed instructions for each enterprise engineering and integration activity. Several existing enterprise architectures do not have a formal foundation (e.g., CIMOSA [13], ARIS [50]) that allows formal verification and validation of enterprise models built using these architectures.

In the past many different enterprise architectures have been developed CIMOSA [13], GRAI/GIM [18], TOVE [24], ARIS [50]. Based on common features, characteristics and elements of these architectures a generalized framework called GERAM (the Generalized Enterprise Reference Architecture and Methodology) has been developed [26]. The GERAM framework provides a generalized template for the development of elaborated enterprise modeling frameworks, based on which several international standards for enterprise modeling have been created [10, 11, 12, 19, 20]. Fig.1 shows a diagram of the three dimensions of the GERAM modeling framework:

(1) The life-cycle phases dimension describes phases in the life cycle of an enterprise.
(2) The instantiation dimension describes the model instantiation ranging from generic, partial to particular.
(3) The views dimension describes enterprise activities from the viewpoint of content, purpose or implementation.

To characterize the framework proposed in this thesis in relation to the GERAM in the following parts of the thesis, a more detailed description of the GERAM's dimensions is provided below.

To compare enterprise architectures GERAM identifies a number of phases of the life cycle of an enterprise that can be supported by enterprise information systems:

(1) Identification of the boundaries of the enterprise and the relations between internal and external environments.
(2) Concept phase, at which the enterprise's mission, vision, values, strategies, objectives, operational concepts, policies and business plans are identified.
(3) Requirements phase, at which operational requirements of the enterprise, its relevant processes and the collection of all their functional, behavioral, informational and capability needs are identified.
(4) Preliminary design (translation of general user requirements into system requirements).

(5) Detailed design (assignment to concrete human, software, hardware components).
(6) Implementation.
(7) Operation.
(8) Decommissioning.

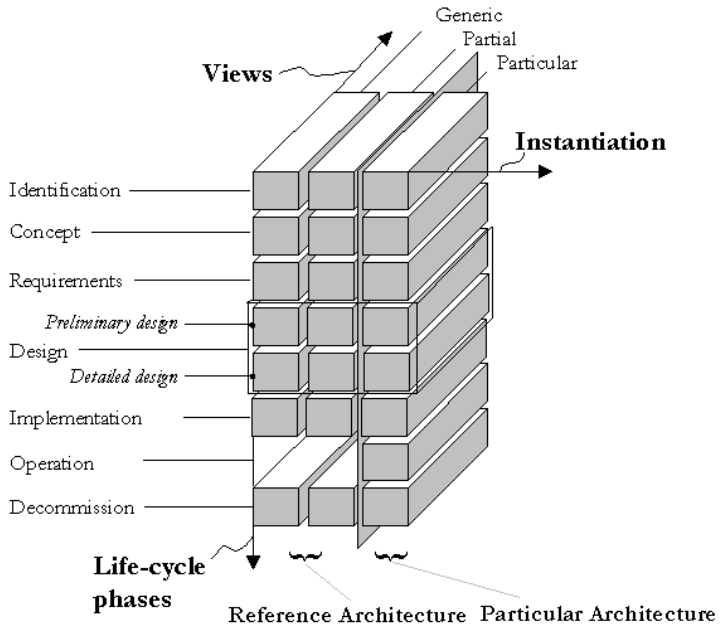


Fig.1. The GERAM Framework (adopted from [26]).

An instantiation of an enterprise model can be generic, partial or particular. Within the *generic* dimension the most generic concepts of enterprise modeling are defined in the form of a meta-model (e.g. entity relationship meta-schema) describing the relationship among modeling concepts available in enterprise modeling languages. *Partial* models capture characteristics common to many enterprises within or across one or more industrial sectors. Thereby these models capitalize on previous knowledge by allowing model libraries to be developed and reused in a "plug-and-play" manner rather than developing the models from scratch. *Particular* models include various designs, models prepared for analysis, executable models to support the operation of the enterprise, etc. Particular representations may consist of several models describing various aspects (or views) of the enterprise.

To reduce the complexity of enterprise models, GERAM proposes a number of dedicated views on enterprises that address particular aspects described along the life cycle phases introduced above. The set of the views of GERAM represents another dimension for the comparison of enterprise architectures. All views of GERAM are divided into three large groups:

(1) Entity model content views.
(2) Entity purpose views.

(3) Entity implementation views.

These views will be discussed in the following.

Entity model content views are dedicated for the user oriented process representation of the enterprise and are defined as follows:

(a) *The function view* presents the functionalities (activities) and the behavior (flow of control) of the business processes of an enterprise. The dynamics of business processes is defined by temporal (ordering) relations, which also determine resource usage/consumption/generation schemas. This view is realized in many existing enterprise architectures and methodologies. Dynamic aspects of the execution of processes are represented using the following formalisms and frameworks: IDEF standards [41], statecharts, Petri-nets [56], event algebra [53], semi-formal languages such as BPML.

(b) *The information view* describes knowledge about objects (material and information) as they are used and produced. Information is represented in the existing architectures by data models. These models comprise entities (objects), attributes (properties of entities), attribute domains, relations among entities, key constraints (expressed as formulae in the first order predicate logic), cardinalities of relations. Different data structures adopted from computer sciences and mathematics are used in the existing architectures [51]: e.g., Entity-Relationship-diagrams used in databases, object-oriented representations, UML class diagrams.

(c) *The resource view* considers resources of an enterprise. Resources are often modeled as separate entities in the existing frameworks, however, with varying level of details.

(d) *The organization view* defines responsibilities and authorities on processes, information and resources. Furthermore, the representation of organizational structure that consists of roles and relations between them (e.g., authority, interaction) is considered in this view. This view is only rarely addressed in the existing architectures. Two of the exceptions are the methodology described in this thesis and CIMOSA [13].

Entity purpose views allow representing the model contents according to the purpose of the enterprise:

(a) *The customer service and product view* addresses the mission of the enterprise entity being studied.

(b) *The management and control view*.

Within these views (strategic, tactical, and operational) goals of an enterprise are defined, with which the business processes of the enterprise should be aligned. Furthermore, decision making activities are addressed in these views.

Entity implementation views describe implementation aspects based on the division between human- and automated tasks:

(a) *The human activities view* represents all information related to the tasks to be done by humans. The view distinguishes between the tasks that may be done by humans (extent of humanisability) and those that will be done by humans (extent of automation).

(b) *The automated activities view* presents all the tasks to be done by machines. This includes information related to those tasks to be carried out by mission support technology and those carried out by management and control

technology (i.e. "technology tasks"). The implementation view distinguishes between the tasks which may be done by machines (extent of automatability) and those which will be done by machines (extent of automation).

Usually enterprise information systems are based on predefined organizational specifications that guide and/or control processes performed by organizational actors. To enable modeling and analysis of behavior of and relations between organizational actors in different organizational and environmental settings, the agent paradigm is particularly useful.

3 Organization-oriented multi-agent systems

An agent is an active object with the ability to perceive, reason, and act. Interactions among agents often take place in the context of certain organizational formations (or structures). On the one hand, such structures may be intentionally designed to enforce certain rules on the behavior of an agent, e.g., which are based on norms and policies, organizational culture etc. On the other hand, organizational structures may *emerge* from the non-random and repeated patterns of interactions among agents. The organization structure provides means to coordinate the execution of tasks in a multi-agent system and to ensure the achievement of organizational goals. Furthermore, by defining the organizational layer in agent models, the predictability of the behavior of the agents can be substantially increased.

In [33] several types of organizational structures are distinguished. Among them hierarchies, holarchies (i.e., hierarchical nested structures that consist of holons; each holon is composed of one or more subordinate entities, and can be a member of one or more superordinate holons), coalitions, teams, congregations (i.e., groups of individuals who have banded together into a typically flat organization in order to derive additional benefits), and federations.

Often organizational structures are specified in terms of roles. A *role* is usually defined as an abstract representation of a set of functionalities performed by an organization. Furthermore, roles are often characterized by sets of requirements (skills, traits and capabilities) that agents should fulfill in order to be allocated to these roles. Note that several agents can be allocated to the same organizational role, and several roles can be enacted by one agent.

Depending on the type of an organizational structure, agents are provided different degrees of autonomy. Usually the behavior of agents is restricted by a set of norms that can be defined at different aggregation levels of the organizational structure. Currently many approaches for modeling normative multi-agent systems have been proposed in the literature. In particular, in [29] different types of organizational norms are defined that may be specified in different types of organizations. Besides prescriptive also descriptive behavioral specifications are provided for multi-agent systems in some approaches [36].

Many of the existing approaches describe organizational specifications, using only two or three aggregation levels; i.e., the level of an individual role, the level of a group composed of roles, and the overall organization level. A few exceptions (e.g., [36]) allow representing as many aggregation levels as required. One of the important

issues considered for multi-level organization-oriented multi-agent systems is the consistency of the structural and behavioral specifications of these systems both of particular aggregation levels and across multiple levels.

Many different methodologies for modeling and design of multi-agent systems that allow representing the organizational layer have been proposed. In Table 1 some of these methodologies are characterized along the following categories: the possibility to specify environment (structure and/or behavior); the availability of means to define internal models of agents and interaction relations between them (e.g., communication); the availability of means to define the organizational layer (both structure and dynamics), the availability of an implementation.

Table 1. Summary of characteristics for some of the methodologies for modeling and design of multi-agent systems. A '+' denotes that a characteristic is addressed in the methodology, '-' denotes that a characteristic is not considered.

Methodology	Environ-ment	Agents		Organization		Implemen-tation
		Internal models	Interaction	Structure	Dynamics	
GAIA	-	-	+	+	-	-
AGR	-	-	+	+	-	+
SODA	+	-	+	+	+	-
MOISE	-	+	+	+	-	+
TROPOS	-	+	+	+	+	+
OperA	+	+/-	+	+	+	+/-

The GAIA methodology [65] addresses two development phases: an analysis and design phase. The analysis phase describes two models: a role model and an interaction model. The role model specifies organizational roles. The interaction model defines the dependencies and relation between roles by means of protocol definitions. At the design phase societies of agents are specified. The design phase provides three models: the agent model, the service model, and the acquaintance model. The agent model identifies agent types, which are sets of roles. The service model identifies the services (or functions) associated with a role. Finally, the acquaintance model identifies the communication links between agent types. GAIA does not capture the internal aspects of agents. The interaction of agents with the environment is not treated separately.

The original AGR methodology proposed in [21] considers only structural aspects of organization models. Each organization model of the AGR comprises a set of interrelated groups that consist of roles (see Fig.2). Groups are related through shared agent(s) allocated to roles within these groups. Furthermore, the AGR places no constraints on the internal architecture of agents and does not provide any implementation details, except for the recommendation to use ACL FIPA language for implementing the communication between agents. In [22] an extension of the AGR is described called AGRE (AGR + Environment). This extension includes a representation of physical (or simply geometrical) environments.

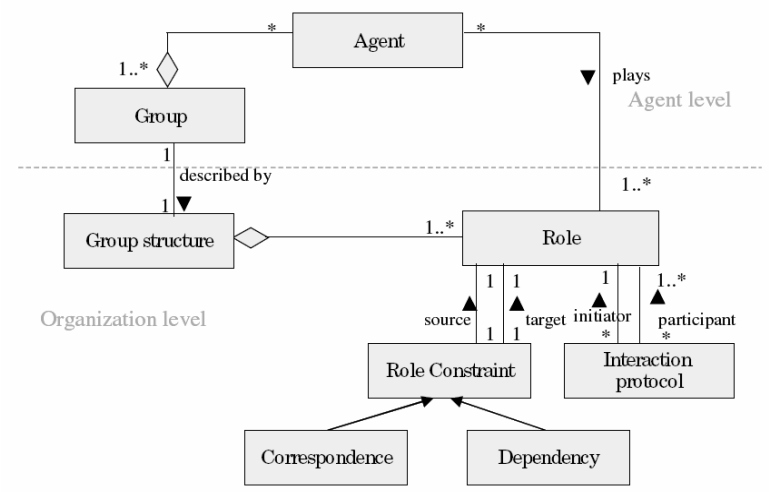


Fig. 2. The meta-model of AGR

The SODA methodology [46] makes a distinction between the analysis and design phases of the development process. The analysis phase provides three models: the role model, the resource model, and the interaction model. The design phase refines the abstract models from the analysis phase and provides three models: the agent model, the society model and the environment model. SODA focuses particularly on inter-agent interactions and does not specify the design of the agents themselves.

The MOISE methodology [31] allows constructing models along three levels: (1) the individual level of agents; (2) the aggregate level of large agent structures; (3) the society level of global structuring and interconnection of the agents and structures with each other. The methodology also addresses some implementation issues of the introduced models. In [34] the original methodology is extended with functional aspects (such as task, plans, and constraints on the behavior of a multi-agent system).

The TROPOS methodology [3] addresses three development phases of multi-agent systems: the analysis, design and implementation phases. The analysis phase is represented by an early and a late requirements phase. The early requirements phase is based on the *i** organizational modeling framework [63]. The late requirements phase results in a list of functional and non-functional requirements for the system. The design phase is divided into an architectural design and a detailed design phase. The architectural design defines the structure of a system in terms of subsystems that are interconnected through data, control and other dependencies. The detailed design defines the behavior of each component. The implementation phase maps the models from the detailed design phase into software by means of Jack Intelligent Agents [32].

The OperA framework [16] focuses on social norms and explicitly defines control policies to establish and reinforce these norms. The framework comprises three components: (1) the organizational model that defines the organizational structure of the society, consisting of roles and interactions; (2) the social model that assigns roles defined in the organization model to agents; and (3) the interaction model that describes possible interactions between agents. Thus, the OperA framework addresses

both organizational structure and dynamics. The internal representation of agents is not clearly defined in this framework. Also, the implementation details are not well explained.

The agent paradigm is also often used in studies within the Computational organization theory.

4 Computational organization theory

In [8] the focus of computational organization theory (COT) is defined as follows:

> The discipline of Computational organization theory focuses on theorizing about, describing, understanding, and predicting the behavior of organizations and the process of organizing using formal approaches (computational, mathematical and logical models). This research includes the development, testing and analysis of computational models, and the development and testing of computational techniques particularly suited to organizational analysis.

In contrast to the traditional organization theory, the COT has a special attention for the investigation of relations between and behavior of organizational individuals modeled by autonomous agents. Further, COT considers organizations as complex intelligent, computational, adaptive entities (synthetic agents), whose dynamics emerges from the behavior of individual agents allocated to the organizational roles. The organizational agents are involved into different types of dynamic networks (e.g., social networks, knowledge networks), which determine different types of relations between the agents. Some of these relations are formally defined and imposed by the organization, whereas others are created by the agents themselves.

A number of models and elaborated modeling frameworks for representing cognitive processes of agents have been developed in COT. For example, in VDT [14] agents are modeled as simple processors with in- and out-boxes; in CORP [8] simple model of experiential learning is used; and in Plural-Soar [9] and TAC Air Soar [55] a fully articulated model of human cognition is used. In [54] mathematical models for studying the value of motivational leadership in teams have been described.

Often analysis of the organizational behavior in COT is performed by agent-based "what-if" simulations [6]. In particular, by using the ORGAHEAD framework [7] the behavior of organizational individuals as they learn, interact and perform tasks is examined. For this information about the formal organizational networks is used. Another computational framework CONSTRUCT-O [5] is dedicated for examining the co-evolution of social structure and culture under different technological and demographic conditions. In contrast to the ORGAHEAD this framework focuses on information diffusion and the impact of technology at the informal or interorganizational level.

One of the research directions in COT is the investigation of social networks of individuals, which are often modeled as multi-agent systems. At the beginning, in contrast to formal organizations, social networks were considered as "proto-organizations" with vaguely defined, constantly changing structures and dynamics. However, in reality social networks often emerge and exist within organizations with

formally defined structures and dynamics. Therefore, to enable feasible analysis of such social networks their relations to different aspects of formal organizations should be identified and taken into account. At present, analysis approaches that take into account both informal and formal aspects of an organization are being developed. In particular, [60] identifies different topologies of network structures within the organizations of certain types.

Furthermore, many studies performed in the area of COT aim at the establishing relations between micro, meso and macro levels of the organizational representation. In particular, in the studies [6, 7, 38, 55, 62] it is investigated how the characteristics and behavior of agents, and different internal organizational configurations influence the organizational performance, adaptability and learning abilities in certain environmental conditions. In [61] the author discusses how the dynamics of single organizations impact the behavioral trends of macro structures, such as markets.

Besides the modeling and analysis of the existing forms of organizations COT also aims at the investigation and testing of new organizational types and theories. For example, in [45] a new organizational form called Edge is introduced and evaluated.

References

1. Bacharach, S.B., Lawler, E.J.: Power and politics in organizations. Jossey-Bass, San Francisco (1980)
2. Blau P.M., Schoenherr R.A.: The structure of organizations. Basic Books Inc., New York London (1971)
3. Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., Perini A. Tropos: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agent and Multi-Agent Systems, 8(3): 203-236 (2004)
4. Campion M.A., Medsker G.J., Higgs A.C.: Relationship between Work Group Characteristics and Effectiveness: Implications for Designing Effective Work Groups. Personnel Psychology, 46: 823-850 (1993)
5. Carley, K.M.: A Theory of Group Stability. American sociological Review, 56: 331-354 (1991)
6. Carley, K.M.: Computational Organization Science: A New Frontier. Arthur M. Sackler Colloquium Series on Adaptive Agents, Intelligence and Emergent Human Organization: Capturing Complexity through Agent-Based Modeling October 4-6, 2001; Irvine, CA, 99, 7257-7262. National Academy of Sciences Press (2002)
7. Carley, K.M., Svoboda, D.: Modeling Organizational Adaptation as a Simulated Annealing Process. Sociological Methods and Research, 25: 138-168 (1996)
8. Carley, K.M., Wallace, W.: Computational organization theory: A New Perspective. Norwell, MA: Kluwer Academic Publishers (2000)
9. Carley, K.M., Kjaer-Hansen, J., Newell, A., Prietula, M.: Plural-SOAR: capabilities and coordination of multiple agents. In: Masuch, M., Warglien M. (eds.) Artificial intelligence in organization and management theory, Elsevier Science (1991)
10. CD 14258. Industrial automation systems - Rules and guidelines for enterprise models. ISO TC184/SC5/WG1 (1996)
11. CEN/TC310. CIM Systems Architecture - Enterprise model execution and integration services - Evaluation report. CEN Report CR: 1831 (1995)
12. CEN/TC310. CIM Systems Architecture - Enterprise model execution and integration services - Statement of Requirements CEN Report CR: 1832 (1995)

13. CIMOSA – Open System Architecture for CIM. ESPRIT Consortium AMICE, Springer-Verlag, Berlin (1993)
14. Cohen, G.P.: The Virtual Design Team: An Information Processing Model of Coordination in Project Design Teams. PhD Thesis, Stanford University, CA. (1992)
15. Cummings T.G., Worley C.G.: Organization development and change. Thomson/South Western (2005)
16. Dignum, V.: A model for organizational interaction: based on agents, founded in logic. Ph.D. Dissertation, Utrecht University (2003)
17. Donaldson, L.: The Contingency Theory of Organizations. Sage, London (2001)
18. Doumeingts G., Vallespir B., Chen D.: Decisional Modelling using the GRAI Grid. In: Bernus, P., Mertins, K. and Schmidt, G. (eds): Handbook on Architectures of Information Systems, Springer-Verlag, 313-338 (1998)
19. ENV 40003. Computer Integrated Manufacturing - Systems Architecture - Framework for Enterprise Modelling CEN/CENELEC (1990)
20. ENV 12204 Advanced Manufacturing Technology - Systems Architecture - Constructs for Enterprise Modelling CEN TC 310/WG1 (1995)
21. Ferber J., Gutknecht O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Y. Demazeau (ed.), Proceedings of Third International Conference on Multi-Agent Systems (ICMAS'98), IEEE Computer Society (1998) 128-135
22. Ferber, J., Michel, F., Baez-Barranco, J.-A.: AGRE: Integrating Environments with Organizations. In: D. Weyns, H. Van Dyke Parunak, M. Fabien (eds.) Proceedings of E4MAS, Lecture Notes in Computer Science, vol. 3374/2005, Springer Verlag (2004) 48-56
23. Forrester, J. W.: Industrial dynamics, Waltham, MA: Pegasus Communications (1961)
24. Fox, M, Barbuceanu M, Gruninger, M, Lin, J.: An Organization Ontology for Enterprise Modelling. In: M. Prietula, K. Carley and L. Gasser (eds.), Simulating Organizations: Computational Models of Institutions and Groups, Menlo Park CA: AAAI/MIT Press, 131-152 (1997)
25. Galbraith, J.R.: Organization design, Addison-Wesley Publishing Company, London Amsterdam Sydney (1978)
26. GERAM: The Generalized Enterprise Reference Architecture and Methodology. IFIP-IFAC Task Force on Architectures for Enterprise Integration, In: Bernus P, Nemes L, Schmidt G. (eds): Handbook on Enterprise Architectures, Springer-Verlag, 21-63 (2003)
27. Giddens, A: Sociology, 5th edn, Polity, Cambridge (2006)
28. Gronau, N.: Enterprise Resource Planning und Supply Chain Management - Architektur und Funktionen. Oldenbourg Wissenschaftsverlag, Munchen (2004)
29. Grossi, D., Aldewereld, H., Vazquez-Salceda, J., Dignum, F.: Ontological Aspects of the Implementation of Norms in Agent-Based Electronic Institutions. Journal of Computational and Mathematical organization theory. Special issue of Normative Multiagent Systems 12 (2-3): 251 -275 (2006)
30. Hannan M.T., Freeman J. The population ecology of organizations. American Journal of Sociology 82: 929-964 (1977)
31. Hannoun, M., Boissier, O., Sichman JS., Sayettat, C.: MOISE: An Organizational Model for Multi-agent Systems. In Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI: Advances in Artificial Intelligence, LNCS, vol. 1952 (2000) 156 - 165
32. Hodgson, A., Roennquist, R., Busetta, P. and Howden, N.: Team Oriented Programming with SimpleTeam. In: J. G. Carbonell and J. Siekmann (eds.), Innovative Concepts for Agent-Based Systems, Lecture Notes in Computer Science, vol. 2564, Springer Verlag (2000) 115-122
33. Horling, B, Lesser, V.: A Survey of multi-agent organizational paradigms. The Knowledge Engineering Review, 19(4): 281-316 (2005)

34. Hubner, J.F., Sichman, J.S. and Boissier, O.: MOISE+: towards a structural, functional and deontic model for MAS organization. In: C. Castelfranchi and L. Johnson (eds), Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02), Bologna, Italy (2002) 501-502
35. Horenberg, J.: Organizational Behavior. Lawrence Erlbaum Associates, Hillsdale, NJ (1994)
36. Jonker C.M., Sharpanskykh, A., Treur, J., Yolum, P.: A Framework for Formal Modeling and Analysis of Organizations, Applied Intelligence, 27(1), 49-66 (2007)
37. Langlois, R.N., Robertson, P. L.: Firms, Markets, and Economic Change: A Dynamic Theory of Business Institutions. London: Routledge (1995)
38. Lin, Z.: The Choice Between Accuracy and Errors: A Contingency Analysis of External Conditions and Organizational Decision Making Performance. In: K.M. Carley and M.J. Prietula (eds.): Computational Organization Theory. L. Erlbaum, Hillsdale, NJ, 67–88 (1994)
39. Lorsch J.W., Lawrence P.R.: Organization design. Richard D. Irwin Inc., USA (1970)
40. March, J.G., Simon, H.A. Organizations. John Wiley & Sons, Inc. (1967)
41. Menzel, C., Mayer, R.J.: The IDEF family of languages. In: Bernus, P. et al. (eds.): Handbook on Architectures of Information Systems, Springer-Verlag, Heidelberg 209-241 (1998)
42. Mintzberg, H.: The Structuring of Organizations, Prentice Hall, Englewood Cliffs (1979)
43. Mooney, J.D.: The principles of organization, Harper & Bros., New York (1947)
44. Morgan, G.: Images of organizations. SAGE Publications, Thousand Oaks London New Delhi (1996)
45. Nissen, M.E.: Computational experimentation on new organizational forms: Exploring behavior and performance of Edge organizations. Journal Computational & Mathematical Organization Theory Springer Netherlands 13(3): 203-240 (2007)
46. Omicini, A.: Societies and infrastructures in the analysis and design of agent-based systems. In: M.J. Wooldridge, P. Ciancarini (eds.), Proceedings of Agent-Oriented Software Engineering Workshop, Springer Verlag (2000)185–193
47. Pfeffer, J.: Organizations and organization theory, Pitman Books Limited, Boston London Melbourne Toronto (1982)
48. Pfeffer, J., Salancik, G.R.: The external control of organizations: A resource dependence perspective, Harper & Row, New York (1978)
49. Romme, AGL: Making a difference: Organization as design. Organization Science, 14: 558-573 (2003)
50. Scheer, A-W., Nuettgens, M. ARIS Architecture and Reference Models for Business Process Management. In: van der Aalst, W.M.P.; Desel, J.; Oberweis, A. (eds.), LNCS 1806, Berlin et al. 366-389 (2000)
51. Schenk, D., Wilson, P.: Information Modeling: The EXPRESS Way. Oxford University Press. (1994)
52. Scott, W.G., Mitchell, T.R., Birnbarum, P.H.: Organization theory: a structural and behavioural analysis, Richard D. Irwin inc., Illinois, USA (1981)
53. Singh, M.P.: Synthesizing distributed constrained events from transactional workflow specifications. In: S.Y.W. Su (ed.), Proceedings of the 12th IEEE International. Conference on Data Engineering, IEEE Computer Society (1996) 616–623
54. Solow, D., Burnetas, A., Piderit, S. K., Leenawong, C.: Mathematical Models for Studying the Value of Motivational Leadership in Teams. Computational & Mathematical Organization Theory, 11: 1: 5-36 (2005)
55. Tambe, M.: Agent Architectures for flexible, practical teamwork. In Proceedings of the AAAI American Association of Artificial Intelligence (1997)
56. Van der Aalst, W. M.P., van Hee, K: Workflow Management: Models, Methods, and Systems. MIT Press (2002)

57. Vroom, V.H. Work and motivation. Wiley, New York (1964)
58. Walter, B. Modern systems research for the behavioral scientist, Aldine Publishing Co, Chicago (1968)
59. Warner, M., Witzel, M.: Managing in Virtual Organizations, Thomson Learning (2004)
60. White, D.R., Owen-Smith, J., Moody, J., Powell, W.W.: Networks, Fields and Organizations: Micro-Dynamics, Scale and Cohesive Embeddings. Computational and Mathematical Organization Theory 10(1):95-117 (2004)
61. White, H.C.: Markets from Networks: Socioeconomic Models of Production. Princeton University Press, NJ (2002)
62. Yilmaz, L.: Validation and Verification of Social Processes within Agent-Based Computational Organization Models. Computational & Mathematical Organization Theory, 12: 4: 283-312 (2006)
63. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. 3rd IEEE Int. Symposium on Requirements Engineering, IEEE Computer Society (1997) 226-235
64. Yukl, G.: Leadership in organizations, 6edn, Englewood Cliffs, NJ: Prentice-Hall (2006)
65. Zambonelli, F., Jennings, N. R., Wooldridge, M.: Developing multiagent systems: the Gaia Methodology, ACM Transactions on Software Engineering and Methodology, 12 (3): 317-370 (2003)

Part II

Formal Foundations

Modelling dynamics poses real challenges for modellers in different disciplines. Currently, continuous modelling techniques based on differential and difference equations are often used to address this challenge, with limited success. In particular, for creating realistic continuous models for natural processes a great number of equations with a multitude of parameters are required. Such models are difficult to analyze, both mathematically and computationally. Further, continuous modelling approaches, such as the Dynamical Systems Theory [3], provide little help for specifying global requirements on a system being modelled and for defining high level system properties that often have a qualitative character (e.g., reasoning, coordination). Also, sometimes system components (e.g., switches, thresholds) have behaviour that is best modelled by discrete transitions. Thus, the continuous modelling techniques have limitations, which can compromise the feasibility of system modelling in different domains. Furthermore, many real world systems (e.g., a television set, a human organization, a human brain) are hybrid in nature, i.e., are characterized by both qualitative and quantitative aspects. To represent and reason about structures and dynamics of such systems, the possibility of expressing both qualitative and quantitative aspects is required. To this end, the Temporal Trace Language (TTL) is proposed, which subsumes languages based on differential equations and temporal logics. Furthermore, TTL supports the specification of the system behaviour at different levels of abstraction, which allows increasing the scalability and reducing the complexity of both system modelling and analysis.

Generally, the expressivity of modelling languages is limited by the possibility to perform effective and efficient analysis of models. Analysis techniques for complex systems include simulation based on system models, and verification of dynamic properties on model specifications and traces generated by simulation or obtained empirically.

For simulation it is essential to have limitations to the language. To this end, an executable language that allows specifying only direct temporal relations can be defined as a sublanguage of TTL; cf. [1]. This language allows representing the dynamics of a system by a (possible large) number of simple temporal (or causal) relations, involving both qualitative and quantitative aspects. Furthermore, using a dedicated tool, TTL formulae that describe the complex dynamics of a system specified in a certain format may be automatically translated into the executable form. Based on the operational semantics and the proof theory of the executable language, a dedicated tool has been developed that allows performing simulations of executable specifications.

To verify properties against specifications of models two types of analysis techniques are widely used: (1) logical proof procedures and (2) model checking [2]. Both techniques allow computation of the entailment relation between the specification of a system model and a property being verified. In particular, by means of model checking entailment relations are justified by checking properties on the set of all theoretically possible traces generated by execution of a system model. To make such verification feasible, expressivity of both the language used for the model specification and the language used for expressing properties has to be sacrificed to a large extent. Therefore, model specification languages provided by most model checkers allow expressing only simple temporal relations in the form of transition rules with limited expressiveness (e.g., no quantifiers). For specifying a complex

temporal relation a large quantity (including auxiliary) of interrelated transition rules is needed. In this part normal forms and a transformation procedure are introduced, which enable automatic translation of an expressive TTL specification into the executable format required for automated verification (e.g., by model checking).

In some situations it is required to check properties only on a limited set of traces obtained empirically or by simulation (in contrast to model checking which requires exhaustive inspection of all possible traces). Such type of analysis, which is computationally much cheaper than model checking, is described in this part.

This part is organised as follows. Chapter 1 describes the syntax and semantics of the TTL language and discusses relations of TTL to other well-known formalisms. Chapter 2 introduces an approach of how quantitative, numerical and qualitative, logical aspects of dynamic systems can be integrated and analyzed using TTL and the related techniques. A general (domain-independent) normal form that enables automatic translation of a TTL specification into the executable format required for different types of computational analysis and a procedure for the transformation into the normal form are introduced in Chapter 3. Different steps of the transformation procedure are illustrated in the context of an example from the area of multi-agent systems. Chapter 4 presents two automated formal analysis techniques that use a normalized system specification: (1) analysis by simulation and (2) verification of relations between different aggregation levels of the system representation. Finally, Chapter 5 introduces another type of analysis technique and a software tool that aim at checking dynamic properties expressed in TTL on a set of simulation and/or empirical traces. Note that the formal techniques described in this part are general and can be applied to any dynamic system (e.g., from the social, biological and cognitive domains). The application of the proposed analysis techniques in organizational context will be described in Part III and in Part V.

References

1. Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J.: A Language and Environment for Analysis of Dynamics by Simulation. International Journal of Artificial Intelligence Tools, 16: 435-464 (2007)
2. McMillan, K.: Symbolic Model Checking, Kluwer Academic Publishers (1993)
3. Port, R. F.; and Gelder. T. van. Mind as Motion: Explorations in the Dynamics of Cognition. MIT Press, Cambridge, MA (1995)

Chapter 1

A Language for Modeling System Dynamics[1]

In this Chapter a language for modeling the dynamics of a system is introduced. For this language called Temporal Trace Language (TTL) first syntax and semantics are introduced (Sections 1 and 2). Then, relations between TTL and some other well-known formalisms are described in Section 3. Finally, some discussions are presented in Section 4.

1 Syntax of TTL

The language TTL is a variant of an order-sorted predicate logic [14]. Whereas standard multi-sorted predicate logic is meant to represent static properties, TTL is an extension of such language with explicit facilities to represent dynamic properties of systems. To specify state properties for system components, ontologies are used which are specified by a number of sorts, sorted constants, variables, functions and predicates (i.e., a signature). State properties are specified based on such ontology using a standard multi-sorted first-order predicate language. For every system component A a number of ontologies can be distinguished used to specify state properties of different types. That is, the ontologies IntOnt(A), InOnt(A), OutOnt(A), and ExtOnt(A) are used to express respectively internal, input, output and external state properties of the component A. For example, a state property expressed as a predicate pain may belong to IntOnt(A), whereas the atom has_temperature(environment, 7) may belong to ExtOnt(A). Often in agent-based modelling input ontologies contain elements for describing perceptions of an agent from the external world (e.g, observed(a) means that a component has an observation of state property a), whereas output ontologies

[1] This chapter is taken from Sharpanskykh, A. and Treur, J.: Analysis of Dynamic Properties Involving Logical and Numerical Aspects, Technical Report TR061204, Artificial Intelligence Department, Vrije Universiteit Amsterdam (2006) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author)

describe actions and communications of agents (e.g., performing_action(b) represents action b performed by a component in its environment).

To express dynamic properties, TTL includes special sorts: TIME (a set of linearly ordered time points), STATE (a set of all state names of a system), TRACE (a set of all trace names; a trace or a trajectory can be thought of as a timeline with a state for each time point), STATPROP (a set of all state property names), and VALUE (an ordered set of numbers). Furthermore, for every sort S from the state language the following TTL sorts exist: the sort S^{VARS}, which contains all variable names of sort S, the sort S^{GTERMS}, which contains names of all ground terms, constructed using sort S; sorts S^{GTERMS} and S^{VARS} are subsorts of sort S^{TERMS}.

In TTL, formulae of the state language are used as objects. To provide names of object language formulae φ in TTL, the operator (*) is used (written as φ*), which maps variable sets, term sets and formula sets of the state language to the elements of sorts S^{GTERMS}, S^{TERMS}, S^{VARS} and STATPROP in the following way:

(1) Each constant symbol c from the state sort S is mapped to the constant name c' of sort S^{GTERMS}.

(2) Each variable x: S from the state language is mapped to the constant name $x' \in S^{VARS}$.

(3) Each function symbol $f: S_1 \times S_2 \times \ldots \times S_n \to S_{n+1}$ from the state language is mapped to the function name $f': S_1^{TERMS} \times S_2^{TERMS} \times \ldots \times S_n^{TERMS} \to S_{n+1}^{TERMS}$.

(4) Each predicate symbol $P: S_1 \times S_2 \times \ldots \times S_n$ is mapped to the function name $P': S_1^{TERMS} \times S_2^{TERMS} \times \ldots \times S_n^{TERMS} \to STATPROP$.

(5) The mappings for state formulae are defined as follows:

a. $(\sim\varphi)^* = not(\varphi^*)$

b. $(\varphi \,\&\, \psi)^* = \varphi^* \wedge \psi^*, \quad (\varphi \mid \psi)^* = \varphi^* \vee \psi^*$

c. $(\varphi \Rightarrow \psi)^* = \varphi^* \to \psi^*, \ (\varphi \Leftrightarrow \psi)^* = \varphi^* \leftrightarrow \psi^*$

d. $(\forall x\, \varphi(x))^* = \forall x'\, \varphi^*(x')$, where x is variable over sort S and x' is any constant of S^{VARS}; the same for ∃.

It is assumed that the state language and the TTL define disjoint sets of expressions. Therefore, further in TTL formulae we shall use the same notations for the elements of the object language and for their names in the TTL without introducing any ambiguity. Moreover we shall use t with subscripts and superscripts for variables of the sort TIME; and γ with subscripts and superscripts for variables of the sort TRACE.

A state is described by a function symbol state: TRACE x TIME → STATE. A trace is a temporally ordered sequence of states. A time frame is assumed to be fixed, linearly ordered, for example, the natural or real numbers. Such an interpretation of a trace contrasts to Mazurkiewicz traces [15] that are frequently used for analysing behaviour of Petri nets. Mazurkiewicz traces represent restricted partial orders over algebraic structures with a trace equivalence relation. Furthermore, as opposed to some interpretations of traces in the area of software engineering [11], a formal logical language is used here to specify properties of traces.

The set of function symbols of TTL includes $\wedge, \vee, \to, \leftrightarrow$: STATPROP x STATPROP → STATPROP; not: STATPROP → STATPROP, and ∀, ∃: S^{VARS} x STATPROP → STATPROP, of which the counterparts in the state language are boolean propositional connectives and quantifiers. Further we shall use $\wedge, \vee, \to, \leftrightarrow$ in infix notation and ∀,∃ in prefix notation for better readability. For example, using such function symbols the state

property about external world expressing that there is no rain and no clouds can be specified as: not(rain) $\wedge$ not(clouds).

To formalise relations between sorts VALUE and TIME, functional symbols –, +, /, •:TIME x VALUE$\rightarrow$ TIME are introduced. Furthermore, for arithmetical operations on the sort VALUE the corresponding arithmetical functions are included.

States are related to state properties via the satisfaction relation denoted by the prefix predicate holds (or by the infix predicate $\models$): holds(state(γ, t), p) (or state(γ, t) |= p), which denotes that state property p holds in trace γ at time point t.

Both state(γ, t) and p are terms of the TTL language. In general, TTL terms are constructed by induction in a standard way from variables, constants and function symbols typed with all before-mentioned TTL sorts.

Transition relations between states are described by dynamic properties, which are expressed by TTL-formulae. The set of *atomic TTL-formulae* is defined as:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds(v_1, u_1) is an atomic TTL formula.
(2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1 = \tau_2$ is a TTL-atom.
(3) If t_1, t_2 are terms of sort TIME, then $t_1 < t_2$ is a TTL-atom.
(4) If v_1, v_2 are terms of sort VALUE, then $v_1 < v_2$ is a TTL-atom.

The set of well-formed TTL-formulae is defined inductively in a standard way using Boolean connectives and quantifiers over variables of TTL sorts. An example of the TTL formula, which describes observational belief creation of an agent, is given below:

'In any trace, if at any point in time t1 the agent A observes that it is raining, then there exists a point in time t2 after t1 such that at t2 in the trace the agent A believes that it is raining'.

$\forall\gamma\ \forall$t1 [holds(state(γ, t1), observation_result(itsraining)) $\Rightarrow$
$\exists$t2 > t1 holds(state(γ, t2) , belief(itsraining))]

The possibility to specify arithmetical operations in TTL allows modelling of continuous systems, which behaviour is usually described by differential equations. Such systems can be expressed in TTL either using discrete or dense time frames. For the discrete case, methods of numerical analysis that approximate a continuous model by a discrete one are often used, e.g., Euler's and Runge-Kutta methods [18]. For example, by applying Euler's method for solving a differential equation dy/dt = f(y) with the initial condition $y(t_0)=y_0$, a difference equation $y_{i+1}=y_i+h^* f(y_i)$ (with $i\geq 0$ the step number and $h>0$ the step size) is obtained. This equation can be modelled in TTL in the following way:

$\forall\gamma\ \forall$t $\forall$v: VALUE holds(state(γ , t), has_value(y, v)) $\Rightarrow$
holds(state(γ , t+1), has_value(y, v + h • f(v)))

The traces γ satisfying the above dynamic property are the solutions of the difference equation.

Furthermore, a dense time frame can be used to express differential equations with derivatives specified using the epsilon-delta definition of a limit, which is expressible in TTL. To this end, the following relation is introduced, expressing that x = dy/dt:

is_diff_of(γ, x, y) :
$\forall$t, w $\forall\varepsilon>0\ \exists\delta>0\ \forall$t', v, v'
0 < dist(t',t) < δ & holds(state(γ, t), has_value(x, w))
& holds(state(γ, t), has_value(y, v))

& holds(state(γ, t'), has_value(y, v'))
⇒ dist((v'-v)/(t'-t),w) < ε

where dist(u,v) is defined as the absolute value of the difference.

Furthermore, a study has been performed in which a number of properties of continuous systems and theorems of calculus were formalized in TTL and used in reasoning. Some results of these studies are presented in Chapter 3.

2 Semantics of TTL

An *interpretation* of a TTL formula is based on the standard interpretation of an order sorted predicate logic formula and is defined by a mapping I that associates each:

(1) sort symbol S to a certain set (subdomain) D_S, such that if $S \subseteq S'$ then $D_S \subseteq D_{S'}$

(2) constant c of sort S to some element of D_S

(3) function symbol f of type $<X_1, \ldots, X_i> \rightarrow X_{i+1}$ to a mapping: $I(X_1) \times \ldots \times I(X_i) \rightarrow I(X_{i+1})$

(4) predicate symbol P of type $<X_1, \ldots, X_i>$ to a relation on $I(X_1) \times \ldots \times I(X_i)$

A *model* M for the TTL is a pair M=<I, V>, where I is an interpretation function, and V is a variable assignment, mapping each variable x of any sort S to an element of D_S. We write V [x/v] for the assignment that maps variables y other than x to V(y) and maps x to v. Analogously, we write M[x/v] = <I, V [x/v]>.

If M=<I, V> is a model of the TTL, then *the interpretation of a TTL term* τ, denoted by τ^M, is inductively defined by:

(1) $(x)^M = V(x)$, where x is a variable over one of the TTL sorts.

(2) $(c)^M = I(c)$, where c is a constant of one of the TTL sorts.

(3) $f(\tau_1,\ldots,\tau_k)^M = I(f)(\tau_1^M,\ldots, \tau_k^M)$, where f is a TTL function of type $S_1 \times \ldots \times S_n \rightarrow S$ and $\tau_1,\ldots, \tau_n$ are terms of TTL sorts $S_1,\ldots, S_n$.

The truth definition of TTL for the model M=<I, V> is inductively defined by:

(1) $\models_M P_i(\tau_1,\ldots,\tau_k)$ iff $I(P_i)(\tau_1^M,\ldots, \tau_k^M) = true$

(2) $\models_M \neg\varphi$ iff $\not\models_M \varphi$

(3) $\models_M \varphi \wedge \psi$ iff $\models_M \varphi$ and iff $\models_M \psi$

(4) $\models_M \forall x(\varphi(x))$ iff $\models_{M[x/v]} \varphi(x)$ for all $v \in D_S$, where x is a variable of sort S.

The semantics of connectives and quantifiers is defined in the standard way. A number of important properties of TTL are formulated in form of axioms:

(1) Equality of traces:

$\forall\gamma_1, \gamma_2 [\forall t [state(\gamma_1, t) = state(\gamma_2, t)] \Rightarrow \gamma_1 = \gamma_2]$

(2) Equality of states:

$\forall s_1, s_2 [\forall a{:}STATPROP [truth\_value(s_1, a) = truth\_value(s_2, a)] \Rightarrow s_1 = s_2]$

(3) Truth value in a state:

$holds(s, p) \Leftrightarrow truth\_value(s, p) = true$

(4) State consistency axiom:

$\forall\gamma, t, p\ (holds(state(\gamma, t), p) \Rightarrow \neg holds(state(\gamma, t), not(p)))$

(5) State property semantics:

a. $holds(s, (p_1 \wedge p_2)) \Leftrightarrow holds(s, p_1)$ & $holds(s, p_2)$

b. $holds(s, (p_1 \vee p_2)) \Leftrightarrow holds(s, p_1)$ | $holds(s, p_2)$

c. $holds(s, not(p_1)) \Leftrightarrow \neg holds(s, p_1)$

(6) For any constant variable name x from the sort S^{VARS}:

holds(s, (∃(x, F))) ⇔ ∃x':S^{GTERMS} holds(s, G), and holds(s, (∀(x, F))) ⇔ ∀x': S^{GTERMS} holds(s, G) with G, F terms of sort STATPROP, where G is obtained from F by substituting all occurrences of x by x'.

(7) Partial order axioms for the sort TIME:

a. $\forall t\ t \leq t$ (Reflexivity)

b. $\forall t_1, t_2 [\ t_1 \leq t_2 \wedge t_2 \leq t_1\ ] \Rightarrow t_1 = t_2$ (Anti-Symmetry)

c. $\forall t_1, t_2, t_3 [\ t_1 \leq t_2 \wedge t_2 \leq t_3\ ] \Rightarrow t_1 \leq t_3$ (Transitivity)

(8) Axioms for the sort VALUE:

a.-c.The same as for the sort TIME

d. Standard arithmetic axioms

(9) Axioms, which relate the sorts TIME and VALUE:

a. $(t + v_1) + v_2 = t + (v_1 + v_2)$

b. $(t \bullet v_1) \bullet v_2 = t \bullet (v_1 \bullet v_2)$

(10) (Optional) Finite variability property (for any trace γ):

$\forall t_0, t_1\ t_0 < t_1 \Rightarrow \exists \delta > 0 [\ \forall t [\ t_0 \leq t \ \&\ t \leq t_1\ ] \Rightarrow \exists t_2 [t_2 \leq t \ \&\ t < t_2 + \delta \ \&\ \forall t_3 [\ t_2 \leq t_3 \ \&\ t_3 \leq t_2 + \delta\ ]\ ] \Rightarrow$
$state(\gamma, t_3) = state(\gamma, t)\ ]$

3 Relation to Other Languages

In this section TTL is compared to a number of existing languages for modelling dynamics of a system.

Executable languages can be defined as sublanguages of TTL. An example of such a language, which was designed for simulation of dynamics in terms of both qualitative and quantitative concepts, is the LEADSTO language, cf. [4]. The LEADSTO language models direct temporal or causal dependencies between two state properties in states at different points in time as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h non-negative real numbers (constants of sort VALUE). In LEADSTO the notation $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$, means:

If state property α holds for a certain time interval with duration g,
then after some delay (between e and f) state property β will hold
for a certain time interval of length h.

A specification in LEADSTO format has as advantages that it is executable and that it can often easily be depicted graphically, in a causal graph or system dynamics style. In terms of TTL, the fact that the above statement holds for a trace γ is expressed as follows:

$\forall t1 [\forall t\ [t1 - g \leq t \ \&\ t < t1\ \Rightarrow\ holds(state(\gamma, t), \alpha)\ ] \Rightarrow$
$\exists d{:}VALUE\ [e \leq d \ \&\ d \leq f\ \&\ \forall t'\ [t1 + d \leq\ t' \ \&\ t' <\ t1 + d + h\ \Rightarrow\ holds(state(\gamma, t'), \beta)\ ]$

Furthermore, TTL has some similarities with the situation calculus [21] and the event calculus [12]. However, a number of important syntactic and semantic distinctions exist between TTL and both calculi. In particular, the central notion of the situation calculus - a situation - has different semantics than the notion of a state in TTL. That is, by a situation is understood a history or a finite sequence of actions, whereas a state in TTL is associated with the assignment of truth values to all state properties (a 'snapshot' of the world). Moreover, in contrast to situation calculus,

where transitions between situations are described by execution of actions, in TTL action executions are used as properties of states.

Moreover, although a time line has been introduced to the situation calculus [17], still only a single path (a temporal line) in the tree of situations can be explicitly encoded in the formulae. In contrast, TTL provides more expressivity by allowing explicit references to different temporally ordered sequences of states (traces) in dynamic properties. For example, this can be useful for expressing the property of trust monotonicity:

'For any two traces $\gamma 1$ and $\gamma 2$, if at each time point t agent A's experience with public transportation in $\gamma 2$ at t is at least as good as A's experience with public transportation in $\gamma 1$ at t, then in trace $\gamma 2$ at each point in time t, A's trust is at least as high as A's trust at t in trace $\gamma 1$'.

$\forall \gamma 1, \gamma 2$ [$\forall$t, $\forall$v1:VALUE [holds(state($\gamma 1$, t), has_value(experience, v1)) &
[$\forall$v2:VALUE holds(state($\gamma 2$, t), has_value(experience, v2) $\rightarrow$ v1$\leq$ v2)]] $\Rightarrow$
[$\forall$t, $\forall$w1:VALUE [holds(state($\gamma 1$, t),has_value(trust, w1)) &
[$\forall$w2:VALUE holds(state($\gamma 2$, t), has_value(trust, w2) $\rightarrow$ w1$\leq$ w2)]]]]

In contrast to the event calculus, TTL does not employ the mechanism of events that initiate and terminate fluents. Event occurrences in TTL are considered to be state occurrences the external world. Furthermore, similarly to the situation calculus, also in the event calculus only one time line is considered.

Formulae of the loosely guarded fragment of the first-order predicate logic [2], which is decidable and has good computational properties (deterministic exponential time complexity), are also expressible in TTL:

$\exists y ((\alpha_1 \wedge \ldots \wedge \alpha_m) \wedge \psi(x, y))$ or $\forall y ((\alpha_1 \wedge \ldots \wedge \alpha_m) \rightarrow \psi(x, y))$,

where x and y are tuples of variables, $\alpha_1 \ldots \alpha_m$ are atoms that relativize a quantifier (the guard of the quantifier), and $\psi(x, y)$ is an inductively defined formula in the guarded fragment, such that each free variable of the formula is in the set of free variables of the guard. Similarly the fluted fragment [19] and $\exists^*\forall^*$ [1] can be considered as sublanguages of TTL.

TTL can also be related to temporal languages that are often used for verification (e.g., LTL and CTL [3, 10]). For example, dynamic properties expressed as formulae in LTL can be translated to TTL by replacing the temporal operators of LTL by quantifiers over time. E.g., consider the LTL formula

G(observation_result(itsraining) $\rightarrow$ ***F***(belief(itsraining)))

where the temporal operator ***G*** means 'for all later time points', and ***F*** 'for some later time point'. The first operator can be translated into a universal quantifier, whereas the second one can be translated into an existential quantifier. Using TTL, this formula then can be expressed, for example, as follows:

$\forall$t1 [holds(state(γ, t1), observation_result(itsraining)) $\Rightarrow$
$\exists$t2 > t1 holds(state(γ, t2), belief(itsraining))]

Note that the translation is not bi-directional, i.e., it is not always possible to translate TTL expressions into LTL expressions due to the limited expressive power of LTL. For example, the property of trust monotonicity specified in TTL above cannot be expressed in LTL because of the explicit references to different traces. Similar observations apply for other well-known modal temporal logics such as CTL.

In contrast to the logic of McDermott [16], TTL does not assume structuring of traces in a tree. This enables reasoning about independent sequences of states (histories) in TTL (e.g., by comparing them), which is also not addressed by McDermott.

4 Discussion

TTL allows the possibility of explicit reference to *time points* and *time durations*, which enables modelling of the dynamics of continuous real-time phenomena. Although the language has a logical foundation, it supports the specification of both qualitative and quantitative aspects of a system, and subsumes specification languages based on differential equations.

Sometimes dynamical systems that combine both quantitative and qualitative aspects are called *hybrid systems* [9]. In contrast to many studies on hybrid systems in computer science (e.g., [26]), in which a state of a system is described by assignment of values to variables, in the proposed approach a state of a system is defined by (composite) objects using a rich ontological basis (i.e., typed constants, variables, functions and predicates). This provides better possibilities for conceptualizing and formalizing different kinds of systems (including those from natural domains). Furthermore, by applying numerical approximation methods for continuous behavior of a system, variables in a generated model become discrete and are treated in the same manner as finite-state transition system variables. Therefore, so-called *control points* [13], at which values of continuous variables are checked and changes in a system's functioning mode are made, are not needed.

Furthermore, more specialized languages can be defined as a sublanguage of TTL. For simulation, the executable language LEADSTO has been developed [4]. For verification, decidable fragments of predicate logics and specialized languages with limited expressivity can be defined as sublanguages of TTL. TTL has similarities (as well as important conceptual distinctions) with (from) situation and event calculi. A proper subclass of TTL formulae can be directly translated into formulae of temporal logics (e.g., LTL and CTL).

Finally, TTL and the related analysis techniques proved their value in a number of research projects in such disciplines as artificial intelligence, cognitive science, biology, and social science. In particular, the analysis of continuous models (i.e., based on differential equations) is illustrated by the case study on trace conditioning considered in [5]. An example of the compositional analysis of a multi-agent system specification in TTL by model checking is described in [22, 8]. In [6] TTL is used for modelling and analysis of adaptive agent behaviour specified by complex temporal relations. The use of arithmetical operations in TTL to perform statistical analysis is illustrated by a case study from the criminology [7].

References

1. Ackermann, W.: Solvable Cases of the Decision Problem. North-Holland Publishing Company, Amsterdam (1962)
2. Andreka, H., Nemeti, I., and van Benthem, J.: Modal Languages and Bounded Fragments of Predicate Logic. Journal of Philosophical Logic 27(3) (1998) 217-274
3. Benthem, J van.: The Logic of Time: A Model-theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse. Reidel, Dordrecht (1983)
4. Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J.: A Language and Environment for Analysis of Dynamics by Simulation. International Journal of Artificial Intelligence Tools, 16: 435-464 (2007)
5. Bosse, T., Jonker, C.M., Los, S.A., Torre, L. van der, and Treur, J.: Formal Analysis of Trace Conditioning. Cognitive Systems Research Journal, 8: 36-47 (2007)
6. Bosse, T., Jonker, C.M., and Treur, J.: On the use of Organisation Modelling Techniques to Address Biological Organisation. Multi-Agent and Grid Systems Journal, 3: 199-223 (2007)
7. Bosse, T., Gerritsen, C., and Treur, J.: Cognitive and Social Simulation of Criminal Behaviour: the Intermittent Explosive Disorder Case. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'07. ACM Press (2007)
8. Broek, E., Jonker, C. M., Sharpanskykh, A., Treur, J., and P. Yolum: Formal Modeling and Analysis of Organizations. In: Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowsk, J. Sichman and J. Vazquez Salceda (eds.), Coordination, Organization, Institutions and Norms in Agent Systems I, Lecture Notes in Artificial Intelligence 3913, Springer-Verlag (2006) 18-34
9. Davoren, J.M.; and Nerode, A: Logics for Hybrid Systems. In Proceedings of the IEEE, 88(7) (2000) 985-1010.
10. Goldblatt, R.: Logics of Time and Computation, 2nd edition, CSLI Lecture Notes 7 (1992)
11. Iglewski, M., and Mincer-Daszkiewicz, J.: Internal design of modules specified in the trace assertion method. Science of Computer Programming, 28 (1997) 139-170
12. Kowalski, R., and Sergot, M.: A logic-based calculus of events, New Generation Computing, 4 (1986) 67-95.
13. Manna, Z., and Pnueli, A.: Verifying Hybrid Systems. In Hybrid Systems, Lecture Notes in Computer Science 736, Springer-Verlag (1993) 4-35.
14. Manzano, M.: Extensions of First Order Logic, Cambridge University Press (1996)
15. Mazurkiewicz, A. Trace Theory. In: Advances in Petri nets II: applications and relationships to other models of concurrency. Springer LNCS, vol. 255 (1987) 279-324.
16. McDermott, D.V.: A Temporal Logic for Reasoning About Processes and Plans. Cognitive Science 6: 101-155(1982)
17. Pinto, J.; and Reiter, R.: Reasoning About Time in the Situation Calculus. Ann. Math. Artificial Intelligence, 14(2-4) (1995) 251-268.
18. Pearson, C.E.: Numerical Methods in Engineering and Science. CRC Press (1986)
19. Purdy, W.C. 1996. Fluted Formulas and the Limits of Decidability, Journal of Symbolic Logic, 61 (1996) 608–620.
20. Rajeev, A., Henzinger, T.A., and Wong-Toi, H. Symbolic analysis of hybrid systems. In Proceedings of the 36th Annual Conference on Decision and Control (CDC), IEEE Press (1997) 702-707
21. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems, Cambridge MA: MIT Press (2001)
22. Sharpanskykh, A., and Treur, J.: Modeling of Agent Behavior Using Behavioral Specifications. In: Fum, D., Missier, F. del, Stocco, A. (eds.), Proceedings of the Seventh International Conference on Cognitive Modelling, ICCM'06 (2006) 280-286

Chapter 2

Integrating Agent Models and Dynamical Systems [1]

Abstract. Agent-based modelling approaches are usually based on logical languages, whereas in many areas dynamical system models based on differential equations are used. This paper shows how to model complex agent systems, integrating quantitative, numerical and qualitative, logical aspects, and how to combine logical and mathematical analysis methods.

1 Introduction

Existing models for complex systems are often based on quantitative, numerical methods such as Dynamical Systems Theory (DST) [23], and more in particular, differential equations. Such approaches often use numerical variables to describe global aspects of the system and how they affect each other over time; for example, how the number of predators affects the number of preys. An advantage of such numerical approaches is that numerical approximation methods and software environments are available for simulation.

The relatively new agent-based modelling approaches to complex systems take into account the local perspective of a possibly large number of separate agents and their specific behaviours in a system; for example, the different individual predator agents and prey agents. These approaches are usually based on qualitative, logical languages. An advantage of such logical approaches is that they allow (automated) logical analysis of the relationships between different parts of a model, for example

[1] This chapter appeared as Bosse, T., Sharpanskykh, A., Treur, J.: In: Baldoni, M., Son, T.C., Riemsdijk, M.B. van, and Winikoff, M. (eds.): Proceedings of the 5th International Workshop on Declarative Agent Languages and Technologies, DALT 2007, 1-16 (2007) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author). The post-proceedings of the workshop will be published by Springer Verlag.

relationships between global properties of the (multi-agent) system as a whole and local properties of the basic mechanisms within (agents of) the system. Moreover, by means of logic-based approaches, declarative models of complex systems can be specified using knowledge representation languages that are close to the natural language. An advantage of such declarative models is that they can be considered and analysed at a high abstract level. Furthermore, automated support (e.g., programming tools) is provided for manipulation and redesign of models.

Complex systems, for example organisms in biology or organisations in the socio-economic area, often involve both qualitative aspects and quantitative aspects. In particular, in the area of Cognitive Science, the lower-level cognitive processes of agents (e.g., sensory or motor processing) are often modelled using DST-based approaches. Furthermore, at the global level the dynamics of the environment, in which agents are situated, is often described by continuous models (i.e., models based on differential equations); e.g., dynamic models of markets, or natural environmental oscillations. Yet agent-based (logical) languages are often used for describing high-level cognitive processes of agents (e.g., processes related to reasoning) and agent interaction with the environment (e.g., agent actions, execution of tasks).

It is not easy to integrate both types of approaches in one modelling method. On the one hand, it is difficult to incorporate logical aspects in differential equations. For example, qualitative behaviour of an agent that depends on whether the value of a variable is below or above a threshold is difficult to describe by differential equations. On the other hand, quantitative methods based on differential equations are not usable in the context of most logical, agent-based modelling languages, as these languages are not able to handle real numbers and calculations.

This paper shows an integrative approach to simulate and analyse complex systems, integrating quantitative, numerical and qualitative, logical aspects within one expressive temporal specification language. Some initial ideas behind the simulation approach proposed in this paper were described in [5, 6]. The current paper elaborates upon these ideas by proposing more extensive means to design precise, stable, and computationally effective simulation models for hybrid systems (i.e., comprising both quantitative and qualitative aspects). Furthermore, it proposes techniques for analysis of hybrid systems, which were not previously considered elsewhere. The developed simulation and analysis techniques are supported by dedicated tools.

In Section 2, this language (called LEADSTO) is described in detail, and is applied to solve an example differential equation. In Section 3, it is shown how LEADSTO can solve a system of differential equations (for the case of the classical Predator-Prey model), and how it can combine quantitative and qualitative aspects within the same model. Section 4 demonstrates how existing methods for approximation (such as the Runge-Kutta methods) can be incorporated into LEADSTO, and Section 5 shows how existing methods for simulation with dynamic step size can be incorporated. Section 6 demonstrates how interlevel relationships can be established between dynamics of basic mechanisms (described in LEADSTO) and global dynamics of a process (described in a super-language of LEADSTO). Finally, Section 7 is a discussion.

2 Modelling Dynamics in LEADSTO

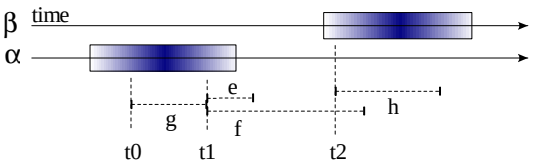
Dynamics can be modelled in different forms. Based on the area within Mathematics called calculus, the Dynamical Systems Theory [23] advocates to model dynamics by continuous state variables and changes of their values over time, which is also assumed continuous. In particular, systems of differential or difference equations are used. This may work well in applications where the world states are modelled in a quantitative manner by real-valued state variables. The world's dynamics in such application show continuous changes in these state variables that can be modelled by mathematical relationships between real-valued variables. However, not for all applications dynamics can be modelled in a quantitative manner as required for DST. Sometimes qualitative changes form an essential aspect of the dynamics of a process. For example, to model the dynamics of reasoning processes usually a quantitative approach will not work. In such processes states are characterised by qualitative state properties, and changes by transitions between such states. For such applications often qualitative, discrete modelling approaches are advocated, such as variants of modal temporal logic, e.g. [20]. However, using such non-quantitative methods, the more precise timing relations are lost too. For the LEADSTO language described in this paper, the choice has been made to consider the timeline as continuous, described by real values, but for state properties both quantitative and qualitative variants can be used. The approach subsumes approaches based on simulation of differential or difference equations, and discrete qualitative modelling approaches. In addition, the approach makes it possible to combines both types of modelling within one model. For example, it is possible to model the exact (real-valued) time interval for which some qualitative property holds. Moreover, the relationships between states over time are described by either logical or mathematical means, or a combination thereof. This will be explained in more detail in Section 2.1. As an illustration, in Section 2.2 it will be shown how the logistic model for population growth in resource-bounded environments [4] can be modelled and simulated in LEADSTO.

2.1 The LEADSTO Language

Dynamics is considered as evolution of states over time. The notion of state as used here is characterised on the basis of an ontology defining a set of properties that do or do not hold at a certain point in time. For a given (order-sorted predicate logic) ontology Ont, the propositional language signature consisting of all *state ground atoms* (or *atomic state properties*) based on Ont is denoted by APROP(Ont). The *state properties* based on a certain ontology Ont are formalised by the propositions that can be made (using conjunction, negation, disjunction, implication) from the ground atoms. A *state* S is an indication of which atomic state properties are true and which are false, i.e., a mapping S: APROP(Ont) $\rightarrow$ {true, false}.

To specify simulation models a temporal language has been developed. This language (the LEADSTO language [7]) enables to model direct temporal dependencies between two state properties in successive states, also called *dynamic properties*. A specification of dynamic properties in LEADSTO format has as advantages that it is executable and that it can often easily be depicted graphically.

The format is defined as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h non-negative real numbers. In the LEADSTO language the notation $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$ (also see Fig. 1), means:



If state property α holds for a certain time interval with duration g, then after some delay (between e and f) state property β will hold for a certain time interval of length h.

Fig. 1. Timing relationships for LEADSTO expressions.

An example dynamic property that uses the LEADSTO format defined above is the following: "observes(agent_A, food_present) $\rightarrow_{2, 3, 1, 1.5}$ beliefs(agent_A, food_present)". Informally, this example expresses the fact that, if agent A observes that food is present during 1 time unit, then after a delay between 2 and 3 time units, agent A will belief that food is present during 1.5 time units. In addition, within the LEADSTO language it is possible to use sorts, variables over sorts, real numbers, and mathematical operations, such as in "has_value(x, v) $\rightarrow_{e, f, g, h}$ has_value(x, v*0.25)". Next, a *trace* or *trajectory* γ over a state ontology Ont is a time-indexed sequence of states over Ont (where the time frame is formalised by the real numbers). A LEADSTO expression $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$, holds for a trace γ if:

$\forall t1\ [\forall t\ [t1–g \leq t < t1 \Rightarrow \alpha \text{ holds in } \gamma \text{ at time } t\ ] \Rightarrow \exists d\ [e \leq d \leq f\ \&\ \forall t'\ [t1+d \leq t' < t1+d+h \Rightarrow \beta \text{ holds in } \gamma \text{ at time } t'\ ]$

To specify the fact that a certain event (i.e., a state property) holds at every state (time point) within a certain time interval a predicate holds_during_interval(event, t1, t2) is introduced. Here event is some state property, t1 is the beginning of the interval and t2 is the end of the interval.

An important use of the LEADSTO language is as a specification language for simulation models. As indicated above, on the one hand LEADSTO expressions can be considered as logical expressions with a declarative, temporal semantics, showing what it means that they hold in a given trace. On the other hand they can be used to specify basic mechanisms of a process and to generate traces, similar to Executable Temporal Logic [3]. More details on the semantics of LEADSTO can be found in [7].

2.2 Solving the Initial Value Problem in LEADSTO: Euler's method

Often behavioural models in the Dynamical Systems Theory are specified by systems of differential equations with given initial conditions for continuous variables and functions. A problem of finding solutions to such equations is known as an initial value problem in the mathematical analysis. One of the approaches for solving this problem is based on discretisation, i.e., replacing a continuous problem by a discrete one, whose solution is known to approximate that of the continuous problem. For this methods of numerical analysis are usually used [22]. The simplest approach for finding approximations of functional solutions for ordinary differential equations is provided by Euler's method. Euler's method for solving a differential equation of the

form $dy/dt = f(y)$ with the initial condition $y(t_0)=y_0$ comprises the difference equation derived from a Taylor series:

$$y(t) = \sum_{n=0}^{\infty} \frac{y^{(n)}(t_0)}{n!} * (t - t_0)^n ,$$

where only the first member is taken into account: $y_{i+1}=y_i+h* f(y_i)$, where $i \geq 0$ is the step number and $h>0$ is the integration step size. This equation can be modelled in the LEADSTO language in the following way:

- Each integration step corresponds to a state, in which an intermediate value of y is calculated.
- The difference equation is modelled by a transition rule to the successive state in the LEADSTO format.
- The duration of an interval between states is defined by a step size h.

Thus, for the considered case the LEADSTO simulation model comprises the rule:

has_value(y, v1) $\rightarrow_{0, 0, h, h}$ has_value(y, v1+h* f(v1))

The initial value for the function y is specified by the following LEADSTO rule:

holds_during_interval(has_value(y, y_0), 0, h)

By performing a simulation of the obtained model in the LEADSTO environment an approximate functional solution to the differential equation can be found.

To illustrate the proposed simulation-based approach based on Euler's method in LEADSTO, the logistic growth model or the Verhulst model [4] which is often used to describe the population growth in resource-bounded environments, is considered: $dP/dt = r*P(1-P/K)$, where P is the population size at time point t; r and K are some constants. This model corresponds to the following LEADSTO simulation model: has_value(y, v1) $\rightarrow_{0, 0, h, h}$ has_value(y, v1+ h*r* v1*(1-v1/K)). The simulation result of this model with the parameters r=0.5 and K=10 and initial value P(0)=1 is given in Figure 2.

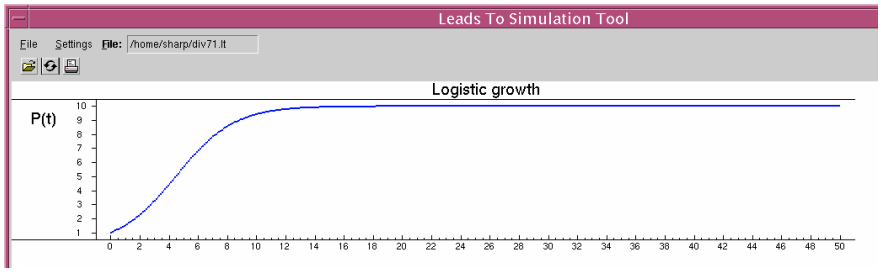


Fig. 2. Logistic growth function modelled in LEADSTO with parameters r=0.5, K=10, P(0)=1.

3 Modelling the Predator-Prey Model in LEADSTO

The proposed simulation-based approach can be applied for solving a system of ordinary differential equations. In order to illustrate this, the classical Lotka-Volterra model (also known as a Predator-Prey model) [21] is considered. The Lotka-Volterra describes interactions between two species in an ecosystem, a predator and a prey.

The model consists of two equations: the first one describes how the prey population changes and the second one describes how the predator population changes. If x(t) and y(t) represent the number of preys and predators respectively, that are alive in the system at time t, then the Lotka-Volterra model is defined by: dx/dt = a*x - b*x*y ; dy/dt = c*b*x*y - e*y where the parameters are defined by: a is the per capita birth rate of the prey, b is a per capita attack rate, c is the conversion efficiency of consumed prey into new predators, and e is the rate at which predators die in the absence of prey. To solve this system, numerical methods derived from a Taylor series up to some order can be used. In the following section it will be shown how Euler's (first-order rough) method can be used for creating a LEADSTO simulation model for finding the approximate solutions for the Predator-Prey problem. After that, in Section 3.2 it will be demonstrated how the generated LEADSTO simulation model can be extended by introducing qualitative behavioural aspects in the standard predator-prey model. Section 3.3 briefly presents a more elaborated example of a LEADSTO simulation model combining quantitative and qualitative aspects of behaviour, addressing simulation of human conditioning processes.

3.1 The LEADSTO language

Using the technique described in Section 2.2, the Lotka-Volterra model is translated into a LEADSTO simulation model as follows:

has_value(x, v1) $\wedge$ has_value(y, v2) $\rightarrow\!\!\!\!\rightarrow_{0, 0, h, h}$ has_value(x, v1+h*(a*v1-b*v1*v2))
has_value(x, v1) $\wedge$ has_value(y, v2) $\rightarrow\!\!\!\!\rightarrow_{0, 0, h, h}$ has_value(y, v2+h*(c*b*v1*v2-e*v2))

The initial values for variables and functions are specified as for the general case. Although Euler's method offers a stable solution to a stable initial value problem, a choice of initial values can significantly influence the model's behaviour. More specifically, the population size of both species will oscillate if perturbed away from the equilibrium. The amplitude of the oscillation depends on how far the initial values of x and y depart from the equilibrium point. The equilibrium point for the considered model is defined by the values x=e/(c*b) and y=a/b. For example, for the parameter settings a=1.5, b=0.2, c=0.1 and e=0.5 the equilibrium is defined by x=25 and y=7.5. Yet a slight deviation from the equilibrium point in the initial values (x_0=25, y_0=8) results in the oscillated (limit cycle) behaviour.

3.2 Extending the Standard Predator-Prey Model with Qualitative Aspects

In this section, an extension of the standard predator-prey model is considered, with some qualitative aspects of behaviour. Assume that the population size of both predators and preys within a certain eco-system is externally monitored and controlled by humans. Furthermore, both prey and predator species in this eco-system are also consumed by humans. A control policy comprises a number of intervention rules that ensure the viability of both species. Among such rules could be following:

- in order to keep a prey species from extinction, a number of predators should be controlled to stay within a certain range (defined by pred_min and pred_max);

- if a number of a prey species falls below a fixed minimum (prey_min), a number of predators should be also enforced to the prescribed minimum (pred_min);
- if the size of the prey population is greater than a certain prescribed bound (prey_max), then the size of the prey species can be reduced by a certain number prey_quota (cf. a quota for fish catch).

These qualitative rules can be encoded into the LEADSTO simulation model for the standard predator-prey case by adding new dynamic properties and changing the existing ones in the following way:

has_value(x, v1) ∧ has_value(y, v2) ∧ v1< prey_max $\rightarrow_{0, 0, h, h}$ has_value(x, v1+h*(a*v1-b*v1*v2))
has_value(x, v1) ∧ has_value(y, v2) ∧ v1 ≥ prey_max $\rightarrow_{0, 0, h, h}$
has_value(x, v1+h*(a*v1-b*v1*v2) - prey_quota)
has_value(x, v1) ∧ has_value(y, v2) ∧ v1 ≥ prey_min ∧ v2 < pred_max $\rightarrow_{0, 0, h, h}$
has_value(y, v2+h* (c*b*v1*v2-e*v2))
has_value(x, v1) ∧ has_value(y, v2) ∧ v2 ≥ pred_max $\rightarrow_{0, 0, h, h}$ has_value(y, pred_min)
has_value(x, v1) ∧ has_value(y, v2) ∧ v1 < prey_min $\rightarrow_{0, 0, h, h}$ has_value(y, pred_min)

The result of simulation of this model using Euler's method with the parameter settings: a=4; b=0.2, c=0.1, e=8, pred_min=10, pred_max=30, prey_min=40, prey_max=100, prey_quota=20, x_0=90, y_0=10 is given in Fig. 3.

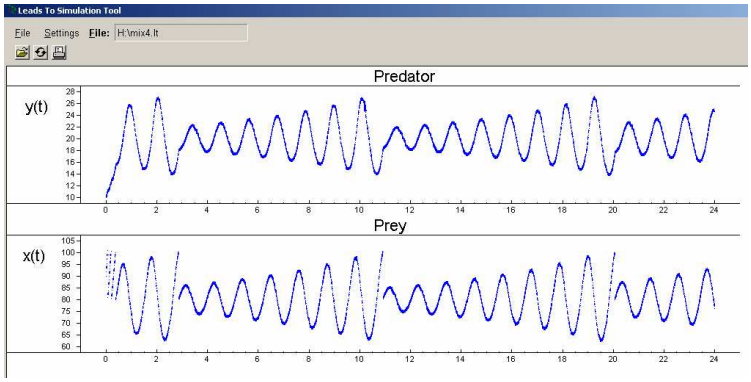


Fig. 3. Simulation results for the Lotka-Volterra model combined some qualitative aspects.

3.3 Example Hybrid LEADSTO Specification - Model for Conditioning

Research into conditioning is aimed at revealing the principles that govern associative learning. An important issue in conditioning processes is the adaptive timing of the conditioned response to the appearance of the unconditioned stimulus. This feature is most apparent in an experimental procedure called *trace conditioning*. In this procedure, a trial starts with the presentation of a *warning stimulus* (S1; comparable to a conditioned stimulus). After a blank interval, called the *foreperiod*, an *imperative stimulus* (S2, comparable to an unconditioned stimulus) is presented to which the participant responds as fast as possible. The *reaction time* to S2 is used as an estimate of the conditioned state of preparation at the moment S2 is presented. In this case, the conditioned response obtains its maximal strength, here called *peak level*, at a moment in time, called *peak time*, that closely corresponds to the moment the unconditioned stimulus occurs.

Machado developed a basic model that describes the dynamics of these conditioning processes in terms of differential equations [18]. The structure of this model is shown in Figure 4. The model posits a layer of *timing nodes* and a single *preparation node*. Each timing node is connected both to the next (and previous) timing node and to the preparation node. The connection between each timing node and the preparation node (called *associative link*) has an adjustable weight associated to it. Upon the presentation of a warning stimulus, a cascade of activation propagates through the timing nodes according to a regular pattern. Owing to this regularity, the timing nodes can be likened to an internal clock or pacemaker. At any moment, each timing node contributes to the activation of the preparation node in accordance with its activation X and its corresponding weight W. The activation of the preparation node reflects the participant's preparatory state, and is as such related to reaction time.

The weights reflect the state of conditioning, and are adjusted by learning rules, of which the main principles are as follows. First, *during* the foreperiod extinction takes place, which involves the decrease of weights in real time in proportion to the activation of their corresponding timing nodes. Second, *after* the presentation of the imperative stimulus a process of reinforcement takes over, which involves an increase of the weights in accordance with the current activation of their timing nodes, to preserve the importance of the imperative moment. Machado describes the more detailed dynamics of the process by a mathematical model (based on linear differential equations), representing the (local) temporal relationships between the variables involved. For example, $d/dt\ X(t,n) = \lambda X(t,n-1) - \lambda X(t,n)$ expresses how the activation level of the n-th timing node $X(t+dt,n)$ at time point t+dt relates to this level $X(t,n)$ at time point t and the activation level $X(t,n-1)$ of the (n-1)-th timing node at time point t. Similarly, as another example, $d/dt\ W(t,n) = -\alpha X(t,n)W(t,n)$ expresses how the n-th weight $W(t+dt,n)$ at time point t+dt relates to this weight $W(t,n)$ at time point t and the activation level $X(t,n)$ of the n-th timing node at time point t.

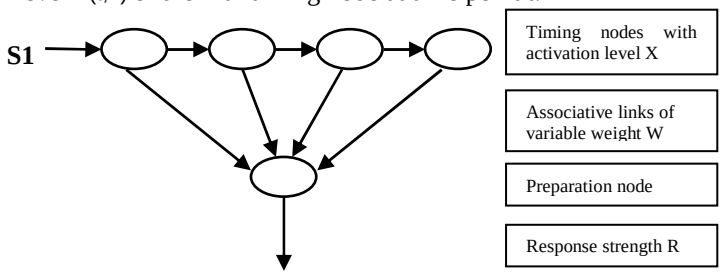


Fig. 4. Structure of Machado's conditioning model (adjusted from [18]).

In [6], LEADSTO has been used to specify Machado's mathematical model in a logical, declarative manner. Some of the dynamic properties used are shown below:

LP5 (Extinction of associative links)
LP5 expresses the adaptation of the associative links during extinction, based on their own previous state and the previous state of the corresponding timing node. Here, α is a learning rate parameter. Formalisation:
$\forall u,v$:REAL $\forall n$:INTEGER
instage(ext) and X(n, u) and W(n, v) $\rightarrow_{0,0,1,1}$ W(n, v*(1-α*u*step))

LP6 (Reinforcement of associative links)
LP6 expresses the adaptation of the associative links during reinforcement, based on their own previous state and the previous state of X. Here, β is a learning rate parameter.
∀u,v:REAL ∀n:INTEGER
instage(reinf) and Xcopy(n, u) and W(n, v) $\rightarrow_{0,0,1,1}$ W(n, v*(1-β*u*step) + β*u*step)

An example simulation trace that has been generated on the basis of this model is shown in Figure 5. The upper part of the figure shows conceptual, qualitative information (e.g., the state properties that indicate the stage of the process); the lower part shows more quantitative concepts, i.e., the state properties involving real numbers with changing values over time (e.g., the preparation level of the person). To limit complexity, only a selection of important state properties was depicted. In the lower part, all instantiations of state property r(X) are shown with different (real) values for X (shown on the vertical axis), indicating the participant's preparation level to respond to a stimulus. For example, from time point 1 to 9, the level of preparation is 0.0, and from time point 9 to 10, the level of preparation is 0.019.

Figure 5 describes the dynamics of a person that is subject to conditioning in an experiment with a foreperiod of 6 time units. As can be seen in the trace, the level of response-related activation increases on each trial. Initially, the subject is not prepared at all: at the moment of the imperative stimulus (S2), the level of response is 0.0. However, already after two trials a peak in response level has developed that coincides exactly with the occurrence of S2. Although this example is relatively simple, it demonstrates the power of LEADSTO to combine (real-valued) quantitative concepts with (conceptual) qualitative concepts.

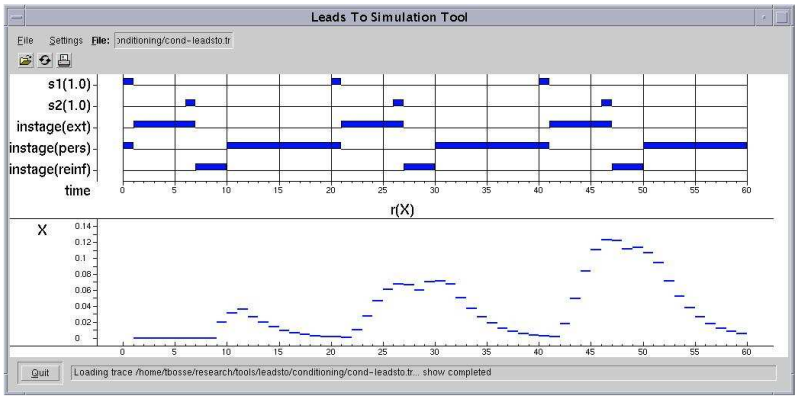


Fig. 5. Example simulation trace of a conditioning process.

4 Simulating the Predator-Prey Model by the Runge-Kutta Method

As shown in [22], within Euler's method the local error at each step (of size h) is $O(h^2)$, while the accumulated error is O(h). However, the accumulated error grows exponentially as the integration step size increases. Therefore, in situations in which

precision of a solution is required, high order numerical methods are used. For the purpose of illustration of high-order numerical approaches the fourth-order Runge-Kutta method is considered. This method is derived from a Taylor expansion up to the fourth order. It is known to be very accurate (the accumulated error is $O(h^4)$) and stable for a wide range of problems. The Runge-Kutta method for solving a differential equation of the form $dx/dt = f(t, x)$ is described by the following formulae:

$x_{i+1} = x_i + h/6 *(k_1 + 2*k_2 + 2*k_3 + k_4)$,

where $i \geq 0$ is the step number, $h>0$ is the integration step size, and

$k_1 = f(t_i, x_i)$, $k_2 = f(t_i + h/2, x_i + h/2 *k_1)$, $k_3 = f(t_i + h/2, x_i + h/2 *k_2)$, $k_4 = f(t_i + h, x_i + h* k_3)$.

Now, using the Runge-Kutta method, the classical Lotka-Volterra model considered in the previous section is described in the LEADSTO format as follows:

$\text{has\_value}(x, v1) \wedge \text{has\_value}(y, v2) \rightarrow_{0, 0, h, h} \text{has\_value}(x, v1 + h/6 *(k_{11} + 2*k_{12} + 2*k_{13} + k_{14}))$
$\text{has\_value}(x, v1) \wedge \text{has\_value}(y, v2) \rightarrow_{0, 0, h, h} \text{has\_value}(y, v2 + h/6 *(k_{21} + 2*k_{22} + 2*k_{23} + k_{24}))$,

where:

$k_{11} = a*v1-b*v1*v2$, $k_{21} = c*b*v1*v2 - e*v2$, $k_{12} = a*(v1 + h/2 *k_{11}) - b*(v1 + h/2 *k_{11})*(v2 + h/2 *k_{21})$, $k_{22} = c*b*(v1 + h/2 *k_{11})*(v2 + h/2 *k_{21}) - e*(v2 + h/2 *k_{21})$, $k_{13} = a*(v1 + h/2 *k_{12}) - b*(v1 + h/2 *k_{12})*(v2 + h/2 *k_{22})$, $k_{23} = c*b*(v1 + h/2 *k_{12})*(v2 + h/2 *k_{22}) - e*(v2 + h/2 *k_{22})$, $k_{14} = a*(v1 + h *k_{13}) - b*(v1 + h *k_{13})*(v2 + h *k_{23})$, $k_{24} = c*b*(v1 + h *k_{13})*(v2 + h *k_{23}) - e*(v2 + h *k_{23})$.

5 Simulation with Dynamic Step Size

Although for most cases the Runge-Kutta method with a small step size provides accurate approximations of required functions, this method can still be computationally expensive and, in some cases, inaccurate. In order to achieve a higher accuracy together with minimum computational efforts, methods that allow the dynamic (adaptive) regulation of an integration step size are used. This section shows how such methods can be incorporated in LEADSTO.

To illustrate the use of methods for dynamic step size control, the biochemical model of [13], summarised in Table 1, is considered.

Table. 1. Glycolysis model by [13].

Variables W: Fructose 6-phosphate X : phosphoenolpyruvate Y : pyruvate N1 : ATP; N2 : ADP; N3 : AMP Differential equations X'[t] == 2*Vpfk - Vxy Y'[t] == Vxy - Vpdc N1'[t] == Vxy + Vak - Vatpase N2'[t] == -Vxy - 2*Vak + Vatpase	Moiety conservation N1[t] + N2[t] + N3 = 20 Initial conditions N1[0] == 10 N2[0] == 9 Y[0] == 0 X[0] == 0 Fixed metabolites W = 0.0001; Z = 0	Rate equations Vxy = 343*N2[t]*X[t]/((0.17 + N2[t])*(0.2 + X[t])) Vak = -(432.9*N3*N1[t] - 133*N2[t]^2) Vatpase = 3.2076*N1[t] Vpdc = 53.1328*Y[t]/(0.3 + Y[t]) (*10.0*Y[t]*) Vpfk = 45.4327*W^2/(0.021*(1 + 0.15*N1[t]^2/N3^2 + W^2))

This model describes the process of glycolysis in *Saccharomyces cerevisiae*, a specific species of yeast. This model is interesting to study, because the concentrations of some of the substances involved (in particular ATP and ADP) are changing at a variable rate: sometimes these concentrations change rapidly, and

sometimes they change very slowly. Using the technique described in Section 2.2 (based on Euler's method), this model can be translated to the following LEADSTO simulation model:

has_value(x, v1) ∧ has_value(y, v2) ∧ has_value(n1, v3) ∧ has_value(n2, v4) →$_{0, 0, h, h}$
has_value(x, v1+ (2* (45.4327*w^2/ (0.021* (1+0.15*v3^2/ (20-v3-v4)^2+w^2)))-343*v4*v1/ ((0.17+v4)* (0.2+v1)))*h)

has_value(x, v1) ∧ has_value(y, v2) ∧ has_value(n1, v3) ∧ has_value(n2, v4) →$_{0, 0, h, h}$
has_value(y, v2+ (343*v4*v1/ ((0.17+v4)* (0.2+v1))-53.1328*v2/ (0.3+v2))*h)

has_value(x, v1) ∧ has_value(y, v2) ∧ has_value(n1, v3) ∧ has_value(n2, v4) →$_{0, 0, h, h}$
has_value(n1, v3+ (343*v4*v1/ ((0.17+v4)* (0.2+v1))+ (- (432.9* (20-v3-v4)*v3-133*v4^2))-3.2076*v3)*h)

has_value(x, v1) ∧ has_value(y, v2) ∧ has_value(n1, v3) ∧ has_value(n2, v4) →$_{0, 0, h, h}$
has_value(n2, v4+ (-343*v4*v1/ ((0.17+v4)* (0.2+v1))-2*
(- (432.9* (20-v3-v4)*v3-133*v4^2))+3.2076*v3)*h)

The simulation results of this model (with a static step size of 0.00001) are shown in Fig. 6. Here the curves for N1 and N2 are initially very steep, but become flat after a while. As demonstrated by Figure 6, for the first part of the simulation, it is necessary to pick a small step size in order to obtain accurate results. However, to reduce computational efforts, for the second part a bigger step size is desirable. To this end, a number of methods exist that allow the dynamic adaptation of the step size in a simulation. Generally, these approaches are based on the fact that the algorithm signals information about its own truncation error. The most straightforward (and most often used) technique for this is *step doubling* and *step halving*, see, e.g. [Gear 1971]. The idea of step doubling is that, whenever a new simulation step should be performed, the algorithm compares the result of applying the current step twice with the result of applying the double step (i.e., the current step * 2) once. If the difference between both solutions is smaller than a certain threshold ε, then the double step is selected. Otherwise, the algorithm determines whether step halving can be applied: it compares the result of applying the current step once with the result of applying the half step (i.e., the current step * 0.5) twice. If the difference between both solutions is smaller than ε, then the current step is selected. Otherwise, the half step is selected.

Since its format allows the modeller to include qualitative aspects, it is not difficult to incorporate step doubling and step halving into LEADSTO. To illustrate this, consider the general LEADSTO rule shown in Section 2.2 for solving a differential equation of the form dy/dt = f(y) using Euler's method:

has_value(y, v1) →$_{0, 0, h, h}$ has_value(y, v1+h* f(v1))

Adding step doubling and step halving to this rule yields the following three rules:

step(h) ∧ has_value(y, v1) ∧ |(v1+2h* f(v1)) - ((v1+h* f(v1))+h* f(v1+h* f(v1)))| ≤ ε
→$_{0, 0, 2h, 2h}$ has_value(y, v1+2h* f(v1)) ∧ step(2h)

step(h) ∧ has_value(y, v1) ∧ |(v1+2h* f(v1)) - ((v1+h* f(v1))+h* f(v1+h* f(v1)))| > ε ∧
|(v1+h* f(v1)) - ((v1+0.5h* f(v1))+0.5h* f(v1+0.5h* f(v1)))| ≤ ε
→$_{0, 0, h, h}$ has_value(y, v1+h* f(v1)) ∧ step(h)

step(h) ∧ has_value(y, v1) ∧ |(v1+h* f(v1)) - ((v1+0.5h* f(v1))+0.5h* f(v1+0.5h* f(v1)))| ≤ ε
→$_{0, 0, 0.5h, 0.5h}$ has_value(y, v1+0.5h* f(v1)) ∧ step(0.5h)

Besides step doubling, many other techniques exist in the literature for dynamically controlling the step size in quantitative simulations. Among these are several

techniques that are especially aimed at the Runge-Kutta methods, see, e.g., [24], Chapter 16 for an overview. Although it is possible to incorporate such techniques into LEADSTO, they are not addressed here because of space limitations.

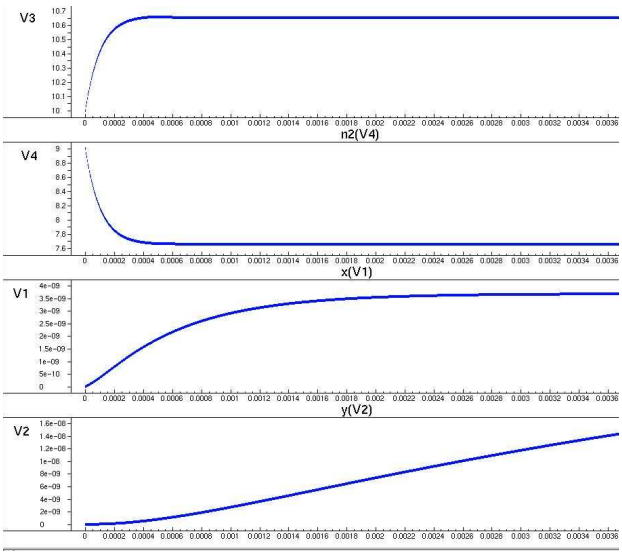


Fig. 6. Simulation results of applying Euler's method to [13]'s glycolysis model.

6 Analysis In Terms of Local-Global Relations

Within the area of agent-based modelling, one of the means to address complexity is by modelling processes at different levels, from the global level of the process as a whole, to the local level of basic elements and their mechanisms. At each of these levels dynamic properties can be specified, and by interlevel relations they can be logically related to each other; e.g., [14], [27]. These relationships can provide an explanation of properties of a process as a whole in terms of properties of its local elements and mechanisms. Such analyses can be done by hand, but also software tools are available to automatically verify the dynamic properties and their interlevel relations. To specify the dynamic properties at different levels and their interlevel relations, a more expressive language is needed than simulation languages based on causal relationships, such as LEADSTO. The reason for this is that, although the latter types of languages are well suited to express the basic mechanisms of a process, for specifying global properties of a process it is often necessary to formulate complex relationships between states at different time points. To this end, the formal language TTL has been introduced as a super-language of LEADSTO; cf. [8]. It is based on order-sorted predicate logic and, therefore, inherits the standard semantics of this variant of predicate logic. That is, the semantics of TTL is defined in a standard way, by interpretation of sorts, constants, functions and predicates, and variable

assignments. Furthermore, TTL allows representing numbers and arithmetical functions. Therefore, most methods used in Calculus are expressible in TTL, including methods based on derivatives and differential equations. In this section, first (in Section 6.1) it is shown how to incorporate differential equations in the predicate-logical language TTL that is used for analysis. Next, in Section 6.2 a number of global dynamic properties are identified, and it is shown how they can be expressed in TTL. In Section 6.3 a number of local dynamic properties are identified and expressed in TTL. Finally, Section 6.4 discusses how the global properties can be logically related to local properties such that a local property implies the global property.

6.1 The LEADSTO language

As mentioned earlier, traditionally, analysis of dynamical systems is often performed using mathematical techniques such as the Dynamical Systems Theory. The question may arise whether or not such modelling techniques can be expressed in the Temporal Trace Language TTL. In this section it is shown how modelling techniques used in the Dynamical Systems approach, such as difference and differential equations, can be represented in TTL. First the discrete case is considered. As an example consider again the logistic growth model: $dP/dt = r*P(1-P/K)$. This equation can be expressed in TTL on the basis of a discrete time frame (e.g., the natural numbers) in a straightforward manner:

$\forall t\ \forall v\ \ state(\gamma, t) \models has\_value(P, v) \ \Rightarrow\ \ state(\gamma, t+1) \models has\_value(P, v + h \bullet r \bullet v \bullet (1 - v/K))$

The traces γ satisfying the above dynamic property are the solutions of the difference equation. However, it is also possible to use the dense time frame of the real numbers, and to express the differential equation directly. To this end, the following relation is introduced, expressing that $x = dy/dt$:

$is\_diff\_of(\gamma, x, y)$:
$\forall t,w\ \ \forall \varepsilon>0\ \exists \delta>0\ \forall t',v,v'\ \ [\ 0 < dist(t',t) < \delta\ \ \&\ \ state(\gamma, t) \models has\_value(x, w)\ \&$
$state(\gamma, t) \models has\_value(y, v)\ \&\ \ state(\gamma, t') \models has\_value(y, v')\ \Rightarrow\ \ \ dist((v'-v)/(t'-t),w) < \varepsilon\ ]$

where γ is the trace that describes the change of values of x and y over time, $dist(u,v)$ is defined as the absolute value of the difference, i.e. $u-v$ if this is ≥ 0, and $v-u$ otherwise. Using this, the differential equation can be expressed by $is\_diff\_of(\gamma, r \bullet P\ (1 - P/K), P)$.

The traces γ for which this statement is true are (or include) solutions for the differential equation. Models consisting of combinations of difference or differential equations can be expressed in a similar manner. This shows how modelling constructs often used in DST can be expressed in TTL. Thus, TTL on the one hand subsumes modelling languages based on differential equations, but on the other hand enables the modeller to express more qualitative, logical concepts as well.

6.2 Mathematical Analysis in TTL: Global Dynamic Properties

Within Dynamical Systems Theory and Calculus, also for global properties of a process more specific analysis methods are known. Examples of such analysis methods include mathematical methods to determine equilibrium points, the

behaviour around equilibrium points, and the existence of limit cycles [10]. Suppose a set of differential equations is given, for example a predator prey model: dx/dt = f(x, y) dy/dt = g(x, y), where f(x, y) and g(x, y) are arithmetical expressions in x and y. Within TTL the following abbreviation is introduced as a definable predicate:

point(γ, t, x, v, y, w) $\Leftrightarrow$ state(γ, t) |= has_value(x, v) $\wedge$ has_value(y, w)

Using this predicate, the following global properties can for example be specified:

Monotonicity

monotic_increase_after(γ, t, x) $\Leftrightarrow$
$\forall$t1, t2 [t $\leq$ t1 < t2 & point(γ, t1, x, v1, y, w1) & point(γ, t2, x, v2, y, w2) $\Rightarrow$ v1<v2]

Bounded

upward_bounded_after_by(γ, t, M) $\Leftrightarrow$ $\forall$t1 [t $\leq$ t1 & point(γ, t1, x, v1, y, w1) $\Rightarrow$ v1$\leq$M]

Equilibrium points

These are points in the (x, y) plane for which, when they are reached by a solution, the state stays at this point in the plane for all future time points. This can be expressed as a global dynamic property in TTL as follows:

has_equilibrium(γ, x, v, y, w) $\Leftrightarrow$ $\forall$t1 [point(γ, t1, x, v, y, w) $\Rightarrow$ $\forall$t2$\geq$t1 point(γ, t2, x, v, y, w)]
occurring_equilibrium(γ, x, v, y, w) $\Leftrightarrow$ $\exists$t point(γ, t, x, v, y, w) & has_equilibrium(γ, x, v, y, w)

Behaviour Around an Equilibrium

attracting(γ, x, v, y, w, ε0) $\Leftrightarrow$ has_equilibrium(γ, x, v, y, w) &
ε0>0 $\wedge$ $\forall$t [point(γ, t, x, v1, y, w1) $\wedge$ dist(v1, w1, v, w) < ε0 $\Rightarrow$
$\forall\varepsilon$>0 $\exists$t1$\geq$t $\forall$t2$\geq$t1 [point(γ, t2, x, v2, y, w2) $\Rightarrow$ dist(v2, w2, v, w) < ε]]

Here, dist(v1, w1, v2, w2) denotes the distance between the points (v1, w1) and (v2, w2) in the (x, y) plane.

Limit cycle

A limit cycle is a set S in the x, y plane such that
$\forall$t, v, w point(γ, t, x, v, y, w) & (v, w) $\in$ S $\Rightarrow$ $\forall$t'$\geq$t, v', w' [point(γ, t', x, v', y, w') $\Rightarrow$ (v', w') $\in$ S]

In specific cases the set can be expressed in an implicit manner by a logical and/or algebraic formula, e.g., an equation, or in an explicit manner by a parameterisation. For these cases it can be logically expressed that a set S is a limit cycle.

(1) When S is defined in an implicit manner by a formula φ(v, w) with S = { (v, w) | φ(v, w) }, then it is defined that S is a limit cycle as follows:

$\forall$t, v, w point(γ, t, x, v, y, w) & φ(v, w) $\Rightarrow$ $\forall$t'$\geq$t, v', w' [point(γ, t', x, v', y, w') $\Rightarrow$ φ(v', w')]

E.g., when S is a circle defined by a formula of the form S = { (v, w) | $v^2 + w^2 = r^2$ }

(2) When a set S in the plane is parameterised by two functions c1, c2: [0, 1] $\rightarrow \Re$, i.e., S = { (c1(u), c2(u)) | u $\in$ [0, 1] }, then S is a limit cycle if

$\forall$t, u point(γ, t, c1(u), c2(u)) $\Rightarrow$ $\forall$t'$\geq$t $\exists$u' point(γ, t', c1(u'), c2(u'))

An example of a parameterising for S in the shape of a circle is as follows:

c1(u) = r cos 2π u, c2(u) = r sin 2π u

In many cases, however, the set S cannot be expressed explicitly in the form of an equation or an explicitly defined parameterisation. What still can be done often is to establish the existence of a limit cycle within a certain area, based on the Poincaré-Bendixson Theorem [16].

6.3 Mathematical Analysis in TTL: Local Dynamic Properties

The global dynamic properties described above can also be addressed from a local perspective. For example, the property of monotonicity (which was expressed above for a whole trace after a certain time point t), can also be expressed for a certain interval (with duration d) around t, as shown below.

Local monotonicity property
monotic_increase_around(γ, t, x, d) ⇔
∀t1, t2 [t-d ≤ t1 < t < t2≤ t+d & point(γ, t1, x, v1, y, w1) & point(γ, t2, x, v2, y, w2) ⇒ v1< v2]

In terms of f and g:
monotic_increase_around(γ, t, x, d) ⇔ point(γ, t, x, v1, y, w1) ⇒ f(v1, w1) > 0

Local bounding property
upward_bounding_around(γ, t, M, δ, d) ⇔
[point(γ, t, x, v1, y, w1) ⇒ ∀t' [t≤t'≤t+d & point(γ, t', x, v2, y, w2) ⇒ M-v2 ≥ (1-δ)*(M-v1)]

In terms of f and g from the equations dx/dt = f(x, y) and dy/dt = g(x, y):
upward_bounding_around(γ, t, M, δ, d) ⇔ point(γ, t, x, v1, y, w1) ⇒ f(v1, w1) ≤ δ/d (M - v1)

Local equilibrium property
From the local perspective of the underlying mechanism, equilibrium points are those points for which dx/dt = dy/dt = 0, i.e., in terms of f and g for this case f(x, y) = g(x, y) = 0.
equilibrium_state(v, w) ⇔ f(v, w) = 0 & g(v, w) = 0

Local property for behaviour around an equilibrium:
attracting(γ, x, v, y, w, δ, ε0, d) ⇔ has_equilibrium(γ, x, v, y, w) &
ε0>0 ∧ 0< δ <1 ∧ d≥0 ∧ ∀t [point(γ, t, x, v1, y, w1) ∧ dist(v1, w1, v, w) < ε0 ⇒
∀t' [t+d≤t'≤t+2d & point(γ, t', x, v2, y, w2) ⇒ dist(v2, w2, v, w) < δ*dist(v1, w1, v, w)]]

In terms of f and g, this can be expressed by relationships for the eigen values of the matrix of derivatives of f and g.

Local limit cycle property
Let a set S in the plane be parameterised by two explicitly given functions c1, c2: [0, 1] → ℜ, i.e., S = { (c1(u), c2(u)) | u ∈ [0, 1] }, and d1(u) = dc1(u)/du, d2(u) = dc2(u)/du. Then S is a limit cycle if:
∀t, u point(γ, t, c1(u), c2(u)) ⇒ d1(u)*g(c1(u), c2(u)) = f(c1(u), c2(u))*d2(u)

6.4 Logical Relations between Local and Global Properties

The properties of local and global level can be logically related to each other by general interlevel relations, for example, the following ones:

∃d>0 ∀t'≥t monotic_increase_around(γ, t', x, d)	⇒ monotic_increase_after(γ, t, x)
∃d>0, δ>0 ∀t'≥t upward_bounding_around(γ, t, M, δ, d)	⇒ upward_bounded_after_by(γ, t, M)
∀t [state(γ, t) |= equilibrium_state(v, w)	⇒ has_equilibrium(γ, x, v, y, w)
∃d>0, δ>0 attracting(γ, x, v, y, w, δ, ε0, d)	⇒ attracting(γ, x, v, y, w, ε0)

These interlevel relations are general properties of dynamic systems, as explained, e.g., in [10]. Full proofs for these relations fall outside the scope of this paper. However, to make them a bit more plausible, the following sketch is given. The first

interlevel relation involving monotonicity can be based on induction on the number of d-intervals of the time axis between two given time points t1 and t2. The second interlevel relation, involving boundedness is based on the fact that local bounding implies that in any d-interval, if the value at the start of the interval is below M, then it will remain below M in that interval. The third interlevel relation, on equilibrium points, is based on the fact that if at no time point the value changes, then at all time points after this value is reached, the value will be the same. For the fourth interlevel relation, notice that local attractiveness implies that for any d-interval the distance of the value to the equilibrium value at the end point is less than δ times the value at the starting point. By induction over the number of d-intervals the limit definition as used for the global property can be obtained.

7 Discussion

The LEADSTO approach discussed in this paper provides means to simulate models of dynamic systems that combine both quantitative and qualitative aspects. A dynamic system, as it is used here, is a system, which is characterised by states and transitions between these states. As such, dynamic systems as considered in [23], which are described by differential equations, constitute a subclass of the dynamic systems considered in this paper. Systems that incorporate both continuous components and discrete components are sometimes called *hybrid systems*. Hybrid systems are studied in both computer science [9], [19] and control engineering [17]. They incorporate both continuous components, whose dynamics is described by differential equations and discrete components, which are often represented by finite-state automata. Both continuous and discrete dynamics of components influence each other. In particular, the input to the continuous dynamics is the result of some function of the discrete state of a system; whereas the input of the discrete dynamics is determined by the value of the continuous state. In the control engineering area, hybrid systems are often considered as switching systems that represent continuous-time systems with isolated and often simplified discrete switching events. Yet in computer science the main interest in hybrid systems lies in investigating aspects of the discrete behaviour, while the continuous dynamics is often kept simple.

Our LEADSTO approach provides as much place for modelling the continuous constituent of a system, as for modelling the discrete one. In contrast to many studies on hybrid systems in computer science (e.g., [25]), in which a state of a system is described by assignment of values to variables, in the proposed approach a state of a system is defined using a rich ontological basis (i.e., typed constants, variables, functions and predicates). This provides better possibilities for conceptualising and formalising different kinds of systems (including those from natural domains). Furthermore, by applying numerical methods for approximation of the continuous behaviour of a system, all variables in a generated model become discrete and are treated equally as finite-state transition system variables. Therefore, it is not needed to specify so-called *control points* [19], at which values of continuous variables are checked and necessary transitions or changes in a mode of a system's functioning are

made. Moreover, using TTL, a super-language of LEADSTO, dynamical systems can be analysed by applying formalised standard techniques from mathematical calculus.

Since LEADSTO has a state-based semantics and allows a high ontological expressivity for defining state properties, many action-based languages (*A*, *B*, *C* [12], *L* [2] and their extensions) can be represented in (or mapped to) the LEADSTO format. In particular, trajectories that define the world evolution in action languages correspond to traces in LEADSTO, fluents evaluated in each state can be represented by state properties, and transitions between states due to actions can be specified by LEADSTO rules that contain the corresponding actions within the antecedents. Furthermore, to represent actions, observations, and goals of agents and facts about the world, the state ontology of LEADSTO includes corresponding sorts, functions and predicates. LEADSTO allows representing both static and dynamic laws as they are defined in [12], and non-deterministic actions with probabilities. To represent and reason about temporal aspects of actions, LEADSTO includes the sort TIME, which is a set of linearly ordered time points.

The expressions of query languages used to reason about actions [2], [12] can be represented in TTL, of which LEADSTO is a sublanguage. TTL formulae can express causality relations of query languages by implications and may include references to multiple states (e.g., histories of temporally ordered sequences of states). Using a dedicated tool [8], TTL formulae can be automatically checked on traces (or trajectories) that represent the temporal development of agent systems.

Concerning other related work, in [26], a logic-based approach to simulation-based modelling of ecological systems is introduced. Using this approach, continuous dynamic processes in ecological systems are conceptualised by system dynamics models (i.e., sets of compartments with flows between them). For formalising these models and performing simulations, the logical programming language Prolog is used. In contrast to this, the LEADSTO approach provides a more abstract (or high-level) logic-based language for knowledge representation.

Also within the area of cognitive modelling, the idea to combine qualitative and quantitative aspects within one modelling approach is not uncommon. A number of architectures have been developed in that area, e.g., ACT-R [1] and SOAR [15]. Such cognitive architectures basically consist of a number of different modules that reflect specific parts of cognition, such as memory, rule-based processes, and communication. They have in common with LEADSTO that they are hybrid approaches, supporting both qualitative (or *symbolic*) and quantitative (or *subsymbolic*) structures. However, in LEADSTO these qualitative and quantitative concepts can be combined within the same expressions, whereas in ACT-R and SOAR separate modules exist to express them. In these cognitive architectures, often the role of the subsymbolic processes is to control the symbolic processes. For example, the subsymbolic part of ACT-R is represented by a large set of parallel processes that can be summarised by a number of mathematical equations, whereas its symbolic part is fulfilled by a production system. Here, the subsymbolic equations control many of the symbolic processes. For instance, if multiple production rules in ACT-R's symbolic part are candidates to be executed, a subsymbolic utility equation may estimate the relative cost and benefit associated with each rule and select the rule with the highest utility for execution.

Accuracy and efficiency of simulation results for hybrid systems provided by the proposed approach to a great extend depend on the choice of a numerical approximation method. Although the proposed approach does not prescribe usage of any specific approximation method (even the most powerful of them can be modelled in LEADSTO), for most of the cases the fourth-order Runge-Kutta method can be recommended, especially when the highest level of precision is not required. For simulating system models, for which high precision is demanded, higher-order numerical methods with an adaptive step size can be applied.

References

1. Anderson, J.R., Lebiere, C. The atomic components of thought. Lawrence Erlbaum Associates, Mahwah, NJ (1998)
2. Baral, C., Gelfond, M., Provetti, A. Representing Actions: Laws, Observation and Hypothesis. Journal of Logic Programming, 31(1-3) (1997) 201-243
3. Barringer, H., Fisher, M., Gabbay, D., Owens, R., Reynolds, M. The Imperative Future: Principles of Executable Temporal Logic, Research Studies Press Ltd. and John Wiley & Sons (1996)
4. Boccara, N. Modeling Complex Systems. Graduate Texts in Contemporary Physics series, Springer-Verlag (2004)
5. Bosse, T., Delfos, M.F., Jonker, C.M., Treur, J. Modelling Adaptive Dynamical Systems to analyse Eating Regulation Disorders. Simulation Journal: Transactions of the Society for Modeling and Simulation International, 82 (2006) 159-171
6. Bosse, T., Jonker, C.M., Los, S.A., Torre, L. van der, Treur, J. Formalisation and Analysis of the Temporal Dynamics of Conditioning. In: Mueller, J.P. and Zambonelli, F. (eds.), Proceedings of the Sixth International Workshop on Agent-Oriented Software Engineering, AOSE'05 (2005) 157-168
7. Bosse, T., Jonker, C.M., Meij, L. van der, Treur, J. LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. In: Eymann, T. et al. (eds.), Proc. MATES'05. LNAI 3550. Springer Verlag (2005) 165-178. Extended version in: International Journal of Artificial Intelligence Tools. To appear, 2007
8. Bosse, T., Jonker, C.M., Meij, L. van der, Sharpanskykh, A., Treur, J. Specification and Verification of Dynamics in Cognitive Agent Models. In: Nishida, T. (ed.), Proceedings of the Sixth International Conference on Intelligent Agent Technology, IAT'06. IEEE Computer Society Press (2006) 247-254
9. Davoren, J.M., Nerode, A. Logics for Hybrid Systems. In Proceedings of the IEEE, 88 (7) (2000) 985-1010
10. Edwards, C.H., Penney, D. L. Calculus with Analytic Geometry. Prentice Hall, London, 5th edition (1998)
11. Gear, C.W. Numerical Initial Value Problems in Ordinary Differential Equations. Englewood Cliffs, NJ: Prentice-Hall (1971)
12. Gelfond, M., Lifschitz, V. Action languages, Electronic Transactions on AI, 3(16) (1998)
13. Hynne F, Dano S, Sorensen PG., Full-scale model of glycolysis in Saccharomyces cerevisiae. Biophys. Chem., 94 (1-2) (2001) 121-63
14. Jonker, C.M., Treur, J. Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. International Journal of Cooperative Information Systems 11 (2002) 51-92.
15. Laird, J.E., Newell, A., and Rosenbloom, P.S. Soar: an architecture for general intelligence. Artificial Intelligence 33 (1) (1987) 1-64.

16. Lefschetz, S. Differential equations: geometric theory. Dover Publications (2005)
17. Liberzon, D., Morse, A. S. Basic problems in stability and design of switched systems, IEEE Control Systems Magazine 19 (5) (1999) 59-70
18. Machado, A. Learning the Temporal Dynamics of Behaviour. Psychological Review, vol. 104 (1997) 241-265
19. Manna, Z., Pnueli, A. Verifying Hybrid Systems. In Hybrid Systems, LNCS 736, Springer-Verlag, (1993) 4-35
20. Meyer, J.J.Ch., Treur, J. (volume eds.). Agent-based Defeasible Control in Dynamic Environments. Series in Defeasible Reasoning and Uncertainty Management Systems (D. Gabbay and Ph. Smets, series eds.) vol. 7, Kluwer Academic Publishers (2002)
21. Morin P.J. Community Ecology. Blackwell Publishing, USA (1999)
22. Pearson, C.E.. Numerical Methods in Engineering and Science. CRC Press (1986)
23. Port, R.F., Gelder, T. van (eds.). Mind as Motion: Explorations in the Dynamics of Cognition. MIT Press, Cambridge, Mass (1995)
24. Press, W. H., Teukolsky, S. A., Vetterling,W. T., Flannery, B. P. Numerical recipes in C: the art of scientific computing. Cambridge university press, second edition (1992)
25. Rajeev, A., Henzinger, T.A., and Wong-Toi, H. Symbolic analysis of hybrid systems. In Proceedings of the 36th Annual Conference on Decision and Control (CDC), IEEE Press (1997) 702-707
26. Robertson, D., Bundy, A., Muetzelfeldt, R., Haggith, M., Ushold, M. Eco-Logic: Logic-Based Approaches to Ecological Modelling. MIT Press, Cambridge, Mass (1991)
27. Sharpanskykh, A., Treur, J. Verifying Interlevel Relations within Multi-Agent Systems. In: Brewka, G., Coradeschi, S., Perini, A., and Traverso, P. (eds.), Proceedings of the 17th European Conference on Artificial Intelligence, ECAI'06, IOS Press (2006) 290-294

Chapter 3

Automated Transformation of Multi-Agent System Behaviour Specifications into Executable Specifications [1]

Abstract. An approach to handle the complex dynamics of a multi-agent system is based on distinguishing aggregation levels. The behaviour at a given aggregation level can specified by a set of dynamic properties at that level, expressed in some (temporal) language. As the behaviour of a system as a whole can be complex, the dynamic properties of higher aggregation levels in principle may involve complex temporal expressions as well. This complexity makes analysis, for example of the logical consequences, difficult. Software tools to support analysis need system specifications describing its basic steps in a simple format. For that reason, for such analyses, often specifications at a lower aggregation level have to be created, describing basic steps in the process. This paper presents a method and tool to support the automated creation of such a specification at a lower aggregation level, as a refinement of a given higher level specification. The generated specification has a simple format which can easily be used for analysis, for example, by simulation, or by verification of logical consequences.

1 Introduction

Often dynamics of a multi-agent system is described by a behavioural temporal specification, which consists of dynamic properties of the multi-agent system. Usually, these properties are expressed as formulae in some (temporal) language. The

[1] Part of this chapter appeared as Sharpanskykh, A., Treur, J.: Verifying Interlevel Relations within Multi-Agent Systems. In: G. Brewka, S. Coradeschi, A. Perini, P. Traverso (eds.), Proceedings of the 17th European Conference on Artificial Intelligence, ECAI'06. IOS Press, 290-294 (2006) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

dynamics of a multi-agent system may be quite complex and difficult to analyze. In order to analyze the behaviour of a complex multi-agent system (e.g., for critical domains such as air traffic control and health care), appropriate approaches for handling the dynamics of the multi-agent system are important. One of the approaches to manage complex dynamics is by distinguishing different aggregation levels; e.g., (Jonker and Treur, 2002). According to this approach, at a higher aggregation level the main overall properties are described, which may have the form of a few temporal expressions of high complexity. At a lower aggregation level a system is described in terms of more basic steps. This usually takes the form of a specification consisting of a large number of temporal expressions in a simpler format.

To formally analyse a given behavioural specification and its logical consequences, dedicated logical analysis techniques (e.g., by simulation or verification) such as described in (Andreka, Benthem and Nemeti, 1998; Bosse, Jonker, Mey and Treur, 2007; Fisher, 2005; Hodkinson, Wolter and Zakharyaschev, 2000; Hustadt, Konev, Riazanov, and Voronkov, 2004) may be of use. The idea is that the logical behavioural specification is used as a system specification, whereas for other temporal properties it is checked whether they are entailed by this system specification. However, such analysis techniques are only appropriate when systems are specified in a simple format, at a low aggregation level, close to a finite state automaton or transition system. In the general case, a given behavioural specification for a multi-agent system may consist of dynamic properties expressed by possibly complex temporal relations, which do not allow direct application of such automated analysis techniques. In order to apply them it is needed to transform the original behavioural specification at a higher aggregation level into a simpler temporal format at a lower aggregation level.

This paper shows how a given behavioural description can be automatically transformed into a specification in a simple temporal logical format, called executable format. A software environment has been developed to automate this process. It takes as input a behavioural specification, which may be at a high aggregation level. Based on this, it constructs as a refinement output in the form of specification at a lower aggregation level in executable temporal logical format. This logical format is close to but yet independent of the formats used in most of the existing verification techniques and tools. The format provides a generic specification without commitment to the type of verification method to be used. It is easily translatable into the various formats often used, in particular those based on some form of logical specification. Examples for which this is shown are: causal-temporal modelling approaches (e.g., LEADSTO; Bosse et al., 2007), modal temporal logic (e.g., MetatheM; Fisher, 1996), monodic first-order logic (Hodkinson and Wolter and Zakharyaschev, 2000), and the guarded fragment of predicate logic (Andreka, and Benthem and Nemeti, 1998).

Section 2 places the modelling issue in a historic context and discusses the main idea. In Section 3 the concepts for formal specification of multi-agent system dynamics and the reified predicate logic temporal language used are briefly introduced. After that, in Section 4 an overview of the transformation procedure from a behavioural specification into an executable specification is given, and illustrated by an example. Sections 5, 6 and 7 describe in more detail the different parts of the transformation: from interaction state (input) to memory state, from memory state to preparation state, from preparation state to interaction state (output). In Section 8 the

different parts of the transformation are combined to obtain the refinement of a behavioural specification into an executable specification. After that, in Section 9 the implementation details of the described transformation procedure are discussed. Section 10 shows how the generated executable specification can easily be used to perform analysis using one of four different available logical techniques and supporting software environments: LEADSTO, Propositional Modal Temporal Logic and MetatheM, Monodic First-Order Temporal Logic, and the Guarded Fragment of Predicate Logic. The paper ends with a discussion in Section 11. More details about proofs are given in Appendix A.

2 Assumptions on Dynamics and Temporal Modelling

Descartes (1633) introduced a perspective on the world that sometimes is called the clockwork universe. This perspective claims that with sufficiently precise understanding of the world's dynamics at some starting time, the future can be predicted by applying a set of laws. He first describes how at some starting time matter came into existence in a diversity of form, size, and motion. From that time on, dynamics continues according to 'laws of nature'.

> 'From the first instant that they are created, He makes some begin to move in one direction and others in another, some faster and others slower (or indeed, if you wish, not at all); thereafter, He makes them continue their motion according to the ordinary laws of nature. For God has so wondrously established these laws that, even if we suppose that He creates nothing more than what I have said, and even if He does not impose any order or proportion on it but makes of it the most confused and most disordered chaos that the poets could describe, the laws are sufficient to make the parts of that chaos untangle themselves and arrange themselves in such right order that they will have the form of a most perfect world, in which one will be able to see not only light, but also all the other things, both general and particular, that appear in this true world.'
>
> (Descartes, The World, 1633, Ch 6: Description of a New World, and on the Qualities of the Matter of Which it is Composed)

> 'Know, then, first that by "nature" I do not here mean some deity or other sort of imaginary power. Rather, I use that word to signify matter itself, insofar as I consider it taken together with all the qualities that I have attributed to it, and under the condition that God continues to preserve it in the same way that He created it. For from that alone (i.e., that He continues thus to preserve it) it follows of necessity that there may be many changes in its parts that cannot, it seems to me, be properly attributed to the action of God (because that action does not change) and hence are to be attributed to nature. The rules according to which these changes take place I call the "laws of nature."'
>
> (Descartes, The World, 1633, Ch 7: On the Laws of Nature of this New World)

Descartes emphasizes that after such a starting time nothing (even no God) except the laws of nature determines the world's dynamics: the role of God is limited to preserving these laws of nature. This view assumes that systematic relationships (laws

of nature) are possible between world states over time, in the sense that (properties of) past world states imply (properties of) future world states. For a temporal modelling perspective, this assumption indicates that dynamics can be described by logical implications from properties of the past to properties of the future:

past properties $\Rightarrow$ future properties

This pattern or special cases thereof can be found in high-level modelling approaches developed in Computer Science and AI, in particular, in the area of Requirements Engineering. Such specifications often are used at a higher aggregation level for the more global properties of a process as a whole, abstracting from the basic steps or mechanisms that realise the process, but also can be used at lower aggregation levels. An example of such a temporal specification 'from past to future', expressed informally is:

> If at some time point in the past agent A received a request for certain information from agent B,
> and at some time point agent A obtains this information,
> then agent A will communicate this information to agent B

The clockwork universe view has been developed further by Newton, Leibniz, Laplace, Ashby and others to what is called Dynamical Systems Theory. Van Gelder and Port (1995) briefly explain what a dynamical system is in the following manner. A system is a set of changing aspects (or state properties) of the world. A state at a given point in time is the way these aspects or state properties are at that time; so a state is characterised by the state properties that hold. The set of all possible states is the state space. A behaviour of the system is the change of these state properties over time, or, in other words, a succession or sequence of states within the state space. Such a sequence in the state space can be indexed, for example, by natural numbers (discrete case) or real numbers (continuous case), and is also called a trace or trajectory. Given these notions, the notion of state-determined system, adopted from Ashby (1952) is taken as the basis to describe what a dynamical system is:

> A system is state-determined only when its current state always determines a unique future behaviour. Three features of such systems are worth noting.
> First, in such systems, the future behaviour cannot depend in any way on whatever states the system might have been in *before* the current state. In other words, past history is irrelevant (or at least, past history only makes a difference insofar as it has left an effect on the current state).
> Second, the fact that the current state determines future behaviour implies the existence of some *rule of evolution* describing the behaviour of the system as a function of its current state. For systems we wish to understand we always hope that this rule can be specified in some reasonable succinct and useful fashion. One source of constant inspiration, of course, has been Newton's formulation of the laws of the solar system.
> Third, the fact that future behaviours are uniquely determined means that state space sequences can never fork. (Gelder and Port, 1995), p. 6.

According to some a dynamical system is just a state-determined system; e.g., Giunti (1995). For others a dynamical system is a state-determined system for which the state properties are described by numerical values; e.g., Van Gelder and Port

(1995). In Ashby (1960), the following claim is expressed, to emphasize the heuristic value of the state-determined system assumption:

> 'Because of its importance, science searches persistently for the state-determined. As a working guide, the scientist has for some centuries followed the hypothesis that, given a set of variables, he can always find a larger set that (1) includes the given variables, and (2) is state-determined. Much research work consists of trying to identify such a larger set, for when it is too small, important variables will be left out of account, and the behaviour of the set will be capricious. The assumption that such a larger set exists is implicit in almost all science, but, being fundamental, it is seldom mentioned explicitly.' (Ashby, 1960), p. 28.

Ashby refers to Temple (1942) and Laplace (1825) for support of his claims. He distinguishes phenomena at a macroscopic level for which his claim is assumed to hold from phenomena at the atomic level, for which the claim turns out not to hold.

> 'Temple, though, refers to '… the fundamental assumption of macrophysics that a complete knowledge of the present state of a system furnishes sufficient data to determine definitely its state at any future time or its response to any future influence.' Laplace made the same assumption about the whole universe when he stated that, given its state at one instant, its future progress should be calculable. The definition given above makes this assumption precise and gives it in a form ready for use in the later chapters. The assumption is now known to be false at the atomic level. We, however, will seldom discuss events at this level; and as the assumption has proved substantially true over great ranges of macroscopic science, we shall use it extensively.' (Ashby, 1960), p. 28.

For the temporal modelling perspective, the state-determined system assumption indicates that dynamics can be described by logical implications from properties of a present state to properties of future states:

present properties $\Rightarrow$ future properties

This format is more limited than the one above, as it does not involve past states. For this format many modelling approaches are known that comply with it, such as dynamical systems theory based on differential equations and causal modelling approaches. Usually such specifications describe a process at a lower aggregation level by the basic steps or mechanisms that realize it, but sometimes also can be used at higher aggregation levels. In continuation of the example specification above, this can be replaced by a specification 'from present to future' of the form in the following manner:

If at some time point agent A received a request for certain information from agent B,
then at a next time point it will have a memory of this request from agent B.

If at some time point agent A has a memory of a request,
then at the next time point it will have a memory of this request.

If at some time point agent A obtains information,
and at that time point it has a memory of a request for this information from agent B
then at a next time point agent A will communicate this information to agent B.

Indeed in this specification additional state ontology is introduced, namely for the (persisting) memory state. Whereas the former specification 'from past to future' as given earlier expresses a property of the process from a more global aggregation level, the latter specification 'from present to future' expresses properties of the same process at a lower, more basic aggregation level. This can be considered a refinement of the former specification. This illustrates Ashby (1960)'s assumption that the ontology for world states can be chosen or extended in such a manner that it is possible to obtain a specification in the more simple and more limited 'present to future' format which determines the behaviour of the system. Carrying over this assumption to the temporal modelling perspective, leads to the question whether and how in general a temporal specification in 'past to future' format can be transformed into one in 'present to future' format, by adding appropriate ontology for state properties. This is the question addressed in this paper.

3 Temporal Specification of Dynamic Properties

From the external perspective, behaviour of a (e.g., multi-agent) system is characterized by a set of dynamic properties, which represent relations over time between its input and output states used for interaction with its environment.

3.1 The Temporal Modelling Approach Adopted

From the philosophical perspective Galton (2003) considers two main streams in temporal logic: modal logic approaches to temporal logic (developed mainly within Computer Science), and predicate logic approaches to temporal logic (developed mainly within AI). In (Galton, 2006) he addresses different approaches in the latter stream in more detail. Two substreams distinguished are the use of temporal arguments within domain predicates, and the reification approach, where state properties are represented not by statements but by terms in the language, and predicates are used to express temporal structure over these term expressions. In this approach part of the model theory is incorporated in the language. This reification approach to predicate logical temporal modelling is the approach adopted here. A basic predicate used in this approach is the holds predicate:

holds_at(p, t)

means that state property p holds at time point t. The model theory notation for this is

$\gamma, t \models p$

where γ is a model representing a possible trace of the process (i.e., a sequence of states indexed by the time frame). The Temporal Trace Language (TTL; Jonker and Treur, 2002; Sharpanskykh and Treur, 2005) is a reified temporal language based on a variant of order-sorted predicate logic (Manzano, 1996). It has some similarities with Situation Calculus (Reiter, 2001) and Event Calculus (Kowalski and Sergot, 1986). Whereas standard many-sorted predicate logic is a language to reason about static

properties, TTL is an extension of such a language by temporal facilities for reasoning about the dynamic properties of dynamical systems. One of the features of TTL is that the trace γ indicated above also can be represented (by a constant or a variable) as a first class citizen in the language. So, as a variant of

holds_at(p, t)

in TTL the expression

holds_at(p, γ, t)

means that state property p holds in the state of trace γ at time point t, also denoted in the language TTL in an infix notation by

state(γ, t) |= p

This feature gives the possibility to quantify over traces and to compare traces, which can be useful and even necessary when adaptive behaviour is analysed. For example, a property such as 'the more exercising, the more skill' compares two traces, one with less and one with more exercising. Another example of such a property is trust monotonicity: 'the better the experiences, the higher the trust'. However, in the current paper these trace-related features of TTL are left out of consideration. The subset of the language TTL considered here does not include quantification over traces; when the argument γ occurs in a formula, it will be considered a fixed constant; thus an expression such as holds_at(p, γ, t) or state(γ, t) |= p is equivalent to (and can be replaced by) holds_at(p, t), which is the more standard expression in reified predicate logic approaches to temporal modelling.

3.2 Brief Overview of the Temporal Trace Language TTL

State properties are expressed in TTL as terms using a standard multi-sorted first-order predicate language with a signature, which consists of a number of sorts, sorted constants, variables, functions and predicates. A system or agent A has assigned an interaction state ontology InteractionOnt(A) for its input and output states. Specifically, within an agent system context, using an ontology InteractionOnt one can define observations of state properties, communications, and actions.

To enable specification of dynamic properties TTL includes special sorts: TIME (a set of linearly ordered time points), STATE (a set of all state names of a system), TRACE (a set of all trace names; a trace or trajectory can be thought of as a timeline with a state for each time point), STATPROP (a set of all state property names), and VALUE (an ordered set of numbers). Furthermore, for every sort S from the state language the following TTL sorts exist: the sort S^{VARS}, which contains all variable names of sort S; the sort S^{GTERMS}, which contains names of all ground terms, constructed using sort S; sorts S^{GTERMS} and S^{VARS} are subsorts of sort S^{TERMS}.

In TTL, formulae of the state language are used as objects. To provide names of state language formulae φ in TTL the operator (*) is used (written as φ*), which maps variable sets, term sets and formula sets of the state language to the elements of TTL sorts S^{GTERMS}, S^{TERMS}, S^{VARS} and STATPROP. As state formulae in a state language occur in a reified form as terms in TTL, a state language and TTL define disjoint sets of

expressions. Therefore, in TTL formulae the same notations for the elements of the state language (i.e, constants, variables, functions, predicates) and for their names in TTL can be used without introducing any ambiguity. Furthermore, t with subscripts and superscripts are used for variables of the sort TIME, and γ with subscripts and superscripts for variables of the sort TRACE.

For the explicit indication of an aspect of a state for a system or agent (called more generally component), to which a state property is related, sorts ASPECT_COMPONENT (a set of the component aspects of a system; i.e., input, output, internal); COMPONENT (a set of all component names of a system); COMPONENT_STATE_ASPECT (a set of all names of aspects of all component states) and a function symbol

comp_aspect: ASPECT_COMPONENT x COMPONENT → COMPONENT_STATE_ASPECT

are used. A state for a component is described by a function symbol state of type TRACE x TIME x COMPONENT_STATE_ASPECT→ STATE.

The set of function symbols of TTL includes ∧, ∨, →, ↔: STATPROP x STATPROP→ STATPROP; not: STATPROP→ STATPROP, **∀, ∃**: S^{VARS} x STATPROP→ STATPROP, which are counterparts of Boolean connectives and quantifiers in the state language. Further we shall use ∧, ∨, →, ↔ in infix notation and **∀, ∃** in prefix notation for better readability.

Notice that also within states statements about time can be made (e.g., in state properties representing memory). To relate time within a state property (sort LTIME) to time external to states (sort TIME) a function present_time: $LTIME^{TERMS}$→ STATPROP is used. Here time is assumed to have the properties of correctness and uniqueness:

Uniqueness of time

This expresses that present_time(t) is true for at most one time point t:

∀t, t'' state(γ, t) |= present_time(t'') ⇒ ∀t', t'≠t'' ¬state(γ, t) |= present_time(t')

Correctness of time

This expresses that present_time(t) is true for the current time point t:

∀t state(γ, t) |= present_time(t)

Furthermore, for the purposes of this paper it is assumed that $LTIME^{GTERMS}$=TIME and $LVALUE^{GTERMS}$=VALUE (LVALUE is a sort of the state language, which is a set of numbers). We shall use u with subscripts and superscripts to denote constants of sort $LTIME^{VARS}$. For formalising relations between sorts VALUE and TIME function symbols –, +, /, •: TIME x VALUE→ TIME are introduced. And for sorts $LVALUE^{TERMS}$ and $LTIME^{TERMS}$ the function symbols –, +, /, • are overloaded: $LTIME^{TERMS}$ x $LVALUE^{TERMS}$ → STATPROP.

The states of a component are related to names of state properties via the formally defined satisfaction relation denoted by the infix predicate |= (or denoted by the prefix predicate holds): state(γ,t,output(A))|= p (or holds(state(γ,t,output(A))), which denotes that the state property with a name p holds in trace γ at time point t at the output state of component A. Sometimes, when the indication of a component aspect is not necessary, this relation will be used without the third argument: state(γ,t) |= p. Both state(γ,t,output(A)) and p are terms of TTL. In general, TTL terms are constructed by induction in a standard way from variables, constants and function symbols typed with all before mentioned TTL sorts.

Temporal relations between states at different points in time are described by dynamic properties, which are expressed by TTL-formulae. The set of *atomic TTL-formulae* is defined as:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds(v_1,u_1) is an atomic TTL formula.
(2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1=\tau_2$ is an atomic TTL formula.
(3) If t_1, t_2 are terms of sort TIME, then $t_1<t_2$ is an atomic TTL formula.

The set of *well-formed TTL-formulae* is defined inductively in a standard way using Boolean connectives and quantifiers. TTL has semantics mainly based on the semantics of order-sorted predicate logic.

Dynamic properties to model a behavioural specification are TTL formulae that are assumed to be specified in the form of a logical implication from a temporal input pattern to a temporal output pattern. The consequent parts of dynamic properties do not contain any disjunctions in order to prevent non-determinism in behaviour. It is a necessary assumption to enable analysis of a system using existing checking techniques and tools. Dynamic properties are expressed using past, interval and future statements, which are defined as follows:

a) A *past statement* for a trace γ and a time point t over state ontology Ont is a temporal statement $\varphi_p(\gamma,t)$ in TTL, such that each time variable s different from t is restricted to the time interval before t: for every time quantifier for a time variable s a restriction of the form $s \leq t$, or $s<t$ is required within the statement.

b) A *future statement* for a trace γ and a time point t over state ontology Ont is a temporal statement $\varphi_f(\gamma,t)$ in TTL, such that for every quantified time variable s, different from t a restriction of the form $s \geq t$, or $s>t$ is required within the statement.

c) An *interval statement* for a trace γ and time points t_1 and t_2 over state ontology Ont is a temporal statement $\varphi(\gamma,t_1,t_2)$ in TTL, that is a past statement for t_2 and a future statement for t_1.

4 Overview of the Transformation Process

The procedure described in a nutshell in this section achieves the transformation of an external behavioural specification for a multi-agent system into executable format. An external behavioural specification of a multi-agent system is defined as follows.

Definition 4.1 (External Behavioural Specification)
An *external behavioural specification* for a multi-agent system consists of dynamic properties $\varphi(\gamma, t)$ expressed in TTL of the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)]$, where $\varphi_p(\gamma, t)$ is a past statement over the interaction ontology and $\varphi_f(\gamma, t)$ is a future statement. The future statement is represented in the form of a conditional behaviour: $\varphi_f(\gamma, t) \Leftrightarrow \forall t_1>t$ $[\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$, where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement over the interaction ontology, which describes a condition for some specified action(s) and/or communication(s), and $\varphi_{bh}(\gamma,t_1)$ is a (conjunction of) future statement(s) for t_1 over the output ontology of the form state(γ,t_1+c) |= output(a), for some integer constant c and action or communication a.

When a past formula $\varphi_p(\gamma,t)$ is true for γ at time t, a potential to perform one or more action(s) and/or communication(s) exists. This potential is realized at time t_1 when the condition formula $\varphi_{cond}(\gamma,t,t_1)$ becomes true, which leads to the action(s) and/or communication(s) being performed at the time point(s) t_1+c indicated in $\varphi_{bh}(\gamma,t_1)$ (this is illustrated in Figure 1).

The term 'external' refers to the fact that such a specification is merely based on the interaction state ontology, no other (e.g., no internal or hidden) state ontology is assumed. An external behavioural specification can include arbitrarily complex temporal relationships. In contrast, an executable specification consists of a set of dynamic properties in a more simple executable temporal language, representing transition-like temporal relations between pairs of states.

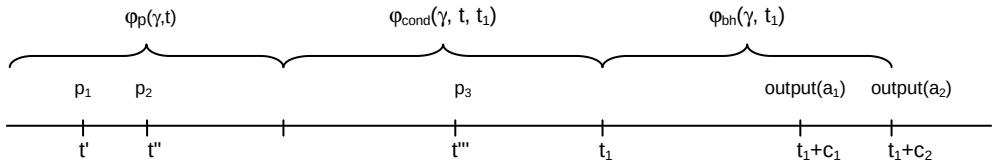


Fig. 1. Graphical illustration of the structure of a formula from an external behavioural specification

Definition 4.2 (Executable Format)
A temporal formula is in *executable format* if it has one of the following forms, for certain state properties , X and Y with $X \neq Y$, and integer constant c.

(1) $\forall t\ state(\gamma, t) \models X \Rightarrow state(\gamma, t+c) \models Y$ (states relation property)
(2) $\forall t\ state(\gamma, t) \models X \Rightarrow state(\gamma, t+1) \models X$ (persistency property)
(3) $\forall t\ state(\gamma, t) \models X \Rightarrow state(\gamma, t) \models Y$ (state relation property)

The next step is to define when a specification in executable format is a refinement of a given external behavioural specification. First the following Definition is needed.

Definition 4.3 (Coinciding Traces)
Two traces γ_1, γ_2 *coincide* on ontology Ont (denoted by a predicate symbol coincide_on: TRACE x TRACE x ONTOLOGY (ONTOLOGY is a sort that contains all names of ontologies)) iff

$$\forall t\ \forall a \in STATATOM_{Ont}\quad [\ state(\gamma_1, t) \models a \ \Leftrightarrow\ state(\gamma_2, t) \models a\ ]$$

where $STATATOM_{Ont} \subseteq STATPROP_{Ont}$ is the sort, which contains all names of ground atoms expressed in terms of Ont.

The notion of refinement as expressed in the following Definition plays a central role in this paper.

Definition 4.4 (Refinement of an External Dynamic Property)
Let $\varphi(\gamma, t)$ be an externally observable dynamic property for. An executable specification $\pi(\gamma, t)$ *refines* $\varphi(\gamma, t)$ iff

(1) $\forall \gamma, t\ \ \pi(\gamma, t) \ \Rightarrow\ \varphi(\gamma, t)$

(2) $\forall \gamma_1, t\ [\ \varphi(\gamma_1, t) \Rightarrow [\ \exists \gamma_2\ \ \text{coincide\_on}(\gamma_1, \gamma_2, \text{InteractionOnt}(A)) \ \&\ \pi(\gamma_2, t)\ ]\ ]$

Note that this Definition achieves that if $\pi(\gamma, t)$ refines $\varphi(\gamma, t)$ and γ is a trace generated in accordance with $\pi(\gamma, t)$ then by (1) it follows that this trace satisfies $\varphi(\gamma, t)$. This means that simulation traces generated on the basis of specification $\pi(\gamma, t)$ are simulation traces for $\varphi(\gamma, t)$. Moreover, (2) guarantees that every trace for $\varphi(\gamma, t)$ can be obtained in this manner. This shows that analysis by simulation of $\varphi(\gamma, t)$ can be done based on $\pi(\gamma, t)$. For another type of analysis, namely verification of logical consequences, first a theorem is needed. This theorem needs the following Lemma. [1]

Lemma 4.1 (Coinciding Traces)
Let $\varphi(\gamma, t)$ be a dynamic property expressed using the state ontology Ont. Then the following hold:

(1) $\text{coincide\_on}(\gamma_1, \gamma_2, \text{Ont})\ \&\ \text{coincide\_on}(\gamma_2, \gamma_3, \text{Ont}) \Rightarrow \text{coincide\_on}(\gamma_1, \gamma_3, \text{Ont})$
(2) $\text{coincide\_on}(\gamma_1, \gamma_2, \text{Ont}) \Rightarrow [\ \varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)\ ].$

Note that for any past interaction statement $\varphi_p(\gamma, t)$ and future interaction statement $\varphi_f(\gamma, t)$ the following holds:

$\forall \gamma 1, \gamma 2\ [\ \text{coincide\_on}(\gamma 1, \gamma 2, \text{InteractionOnt}) \Rightarrow [\ \varphi_p(\gamma 1, t) \Leftrightarrow \varphi_p(\gamma 2, t) \ \&\ \varphi_f(\gamma 1, t) \Leftrightarrow \varphi_f(\gamma 2, t)\ ]]$

Theorem 4.1 (Refinement Implies the Same Consequences)
If the executable specification $\pi(\gamma, t)$ refines the external behavioural specification $\varphi(\gamma, t)$ of a multi-agent system, and $\psi(\gamma, t)$ is a dynamic interaction property of the multi-agent system in its environment, expressed using the interaction ontology, then

$$[\forall \gamma\ [\ \pi(\gamma, t) \Rightarrow \psi(\gamma, t)\ ]\ ] \Leftrightarrow [\forall \gamma\ [\ \varphi(\gamma, t) \Rightarrow \psi(\gamma, t)\ ]\ ]$$

This Theorem 4.1 shows that when $\pi(\gamma, t)$ refines $\varphi(\gamma, t)$, verification of logical consequences of $\varphi(\gamma, t)$ can be done by verification of logical consequences of $\pi(\gamma, t)$. Therefore, summarizing, when a refinement of $\varphi(\gamma, t)$ has been obtained, analysis is supported of $\varphi(\gamma, t)$ both by simulation and by verification of logical consequences.

In Section 8 it will be proven that every external behavioural specification can be refined into an executable specification. To obtain this refinement, an automated transformation procedure can be used as described below and in Sections 5 to 7.

For transformation of an external behavioural specification into executable format, postulated internal states of the system are used. Internal states of a component or system A are described using a postulated internal state ontology InternalOnt(A). In Cognitive Science, which has been used as a source of inspiration, it is often assumed that an agent maintains a memory in the form of some internal model of the history; e.g., (Dennett, 1991; Damasio, 2000). Furthermore, we assume that internal states are formed on the basis of (input) observations (sensory representations) or communications. For this the function symbol memory: $\text{LTIME}^{\text{TERMS}} \times \text{STATPROP} \rightarrow \text{STATROP}$ is used. For example, memory(t, observed(a)) expresses that the component has memory that it observed a state property a at time point t. Before performing an action or communication it is postulated that a component creates an internal

[1] Proofs of lemmas, propositions and theorems given in this paper are provided in Appendix A.

preparation state. For example, preparation_for(b) represents a preparation of a component to perform an action or a communication b. Each dynamic property in the internal behavioural specification is specified in *executable* form.

The Transformation Procedure: Brief Outline

Let $\varphi(\gamma,t)$ be a non-executable dynamic property from an external behavioural specification for the multi-agent system, for which an executable representation should be found.

(1) Identify the set $Th_{o\to m}$ of executable temporal properties, which describe transitions from interaction states to memory states (Section 5) (for a graphical representation of relations between the states considered in this procedure see Figure 2).
(2) Identify the set $Th_{m\to p}$ of executable temporal properties, which describe transitions from memory states to preparation states for output (Section 6).
(3) Identify the set $Th_{p\to o}$ of executable properties, which describe the transition from preparation states to the corresponding output states (Section 7).
(4) From the sets of executable properties, identified during steps 1-3, construct the specification $\pi(\gamma,t) = Th_{o\to m} \cup Th_{m\to p} \cup Th_{p\to o}$ (considered as conjunction), which describes a refinement of $\varphi(\gamma,t)$ (Section 8).

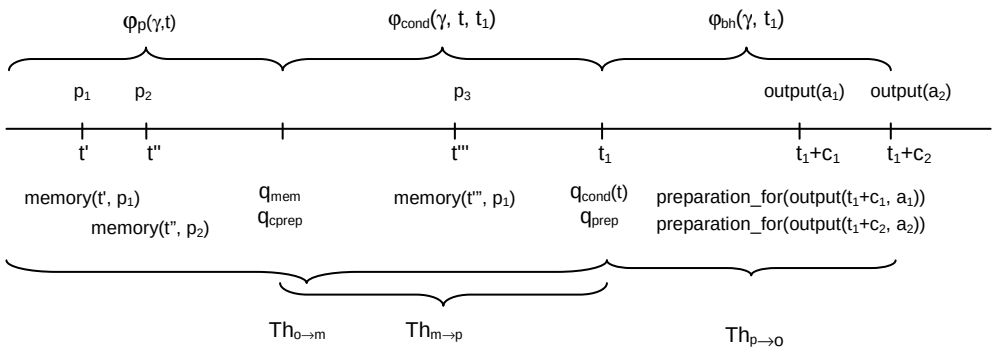


Fig. 2. A graphical representation of relations between interaction states described by a non-executable dynamic property and internal states described by rules from the executable theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to o}$.

The details of the proposed procedure are described in the next three sections by means of an example, in which a multi-agent system for co-operative information gathering is considered. The multi-agent system consists of four interacting components: two information gathering agents A and B, agent C, and environment component E representing the external world. Each of the agents is able to acquire partial information from an external source (component E) by initiated observations. Each agent can be reactive or proactive with respect to the information acquisition process. An agent is proactive if it is able to start information acquisition independently of requests of any other agents, and an agent is reactive if it requires a request from some other agent to perform information acquisition.

Observations of any agent taken separately are insufficient to draw conclusions of a desired type; however, the combined information of both agents is sufficient. Therefore, the agents need to co-operate to be able to draw conclusions. Each agent

can be proactive with respect to the conclusion generation, i.e., after receiving both observation results an agent is capable to generate and communicate a conclusion to agent C. Moreover, an agent can be request pro-active to ask information from another agent, and an agent can be pro-active or reactive in provision of (already acquired) information to the other agent.

For the components of the multi-agent system, a number of dynamic properties were identified and formalized in TTL as it is shown below. In the formalization the variables A1 and A2 are defined over the sort $\text{AGENT}^{\text{TERMS}}$, the constant E belongs to the sort $\text{ENVIRONMENTAL\_COMPONENT}^{\text{GTERMS}}$, the variable IC is defined over the sort $\text{INFORMATION\_CHUNK}^{\text{TERMS}}$, the constants IC1, IC2 and IC3 belong to the sort $\text{INFORMATION\_CHUNK}^{\text{GTERMS}}$ and the constant C belongs to the sort $\text{AGENT}^{\text{TERMS}}$.

DP1(A1, A2) (Effectiveness of information request transfer between agents)
∀IC∀t1 [state(γ, t1, output(A1)) |= output(communicated(request_from_to_for(A1, A2, IC)))
⇒ state(γ, t1+c, input(A2)) |= communicated(request_from_to_for(A1, A2 , IC))]

DP2(A1, A2) (Effectiveness of information transfer between agents)
∀IC∀t1 [state(γ, t1, output(A1)) |= output(communicated(message_from_to(A1, A2, IC)))
⇒ state(γ, t1+c, input(A2)) |= communicated(message_from_to(A1, A2, IC))]

DP3(A1, E) (Effectiveness of information transfer between an agent and environment)
∀IC∀t1 [state(γ, t1, output(A1)) |= output(obs_focus_from_to_for(A1, E, IC))
⇒ state(γ, t1+c, input(E)) |= observed(obs_focus_from_to_for(A1, E, IC))]

DP4(A1, E) (Information provision effectiveness)
∀IC∀t1 [state(γ, t1, input(E)) |= observed(obs_focus_from_to_for(A1, E, IC))
⇒ state(γ, t1+c, output(E)) |= observed(provide_result_from_to(E, A1, IC))]

DP5(E, A1) (Effectiveness of information transfer between environment and an agent)
∀IC∀t1 [state(γ, t1, output(E)) |= observed(provide_result_from_to(E, A1, IC))
⇒ state(γ, t1+c, input(A1)) |= observed(provided_result_from_to(E, A1, IC))]

DP6(A1, A2) (Information acquisition reactiveness)
∀IC∀t1 [state(γ, t1, input(A2)) |= communicated(request_from_to_for(A1, A2, IC))
⇒ state(γ, t1+c, output(A2)) |= output(obs_focus_from_to_for(A2, E, IC))]

DP7(A1, A2) (Information provision reactiveness)
∀IC [∃t1 [t1<t & state(γ, t1, input(A2)) |= communicated(request_from_to_for(A1, A2, IC))]]
⇒ ∀t2 [t<t2 & state(γ, t2, input(A2)) |= observed(provided_result_from_to(E, A2, IC) ⇒
state(γ, t2+c, output(A2)) |= output(communicated(message_from_to(A2, A1, IC)))]]

DP8(A1, A2) (Conclusion proactiveness)
∀IC1, IC2 [∀t1, t2 t1<t & t2<t & state(γ, t1, input(A1)) |= observed(provided_result_from_to(E, A1, IC1)) & state(γ, t2, input(A1)) |= communicated(message_from_to(A2, A1, IC2))
⇒ ∃IC3, t4>t [state(γ, t4, output(A1)) |=output(communicated(message_from_to(A1, C, IC3)))]]

DP9(A1, E) (Information acquisition proactiveness)
∃IC state(γ, c, output(A1)) |= output(obs_focus_from_to_for(A1, E, IC))

DP10(A1, A2) (Information request proactiveness)
∃IC state(γ, c, output(A1)) |= output(communicated(request_from_to_for(A1, A2, IC)))

Notice that most of the properties in the behavioural specification above (e.g., DP1, DP2) are already specified in executable format. Therefore, as an illustration the transformation procedure is applied to properties such as DP7 and DP8 which are non-executable. To illustrate the required transformation the dynamic property that describes an information provision reactiveness of the agent B has been chosen (DP7(A1, A2)). Informally this property expresses that the agent A2 generates an

information chunk (the constant IC of sort INFORMATION_CHUNK^{GTERMS}) for the agent A1 if the agent A2 observes the IC at its input from the environment and at some point in the past A2 received a request for the IC from the agent A1. According to the definition of an external behavioural specification the considered property can be represented in the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)]$, where $\varphi_p(\gamma, t)$ is a formula

$\exists t2 \leq t$ state(γ, t2, input(B)) |= communicated(request_from_to_for(A, B, IC))

and $\varphi_f(\gamma, t)$ is a formula

$\forall t1 > t$ [state(γ, t1, input(B)) |= observed(provided_result_from_to(E, B, IC)) $\Rightarrow$
state(γ, t1+c, output(B)) |= output(communicated(message_from_to(B, A, IC)))]

with $\varphi_{cond}(\gamma, t, t1)$ is

state(γ, t1, input(B)) |= observed(provided_result_from_to(E, B, IC))

and $\varphi_{bh}(\gamma, t1)$ is

state(γ, t1+c, output(B)) |= output(communicated(message_from_to(B, A, IC)))],

where t is the present time point with respect to which the formulae are evaluated and c is some natural number.

5 From Interaction States to Memory States

In this section the part of the executable specification describing the basic steps from interaction states to memory states is addressed. Here the past part $\varphi_p(\gamma, t)$ of the behavioural specification is taken and encoded in a memory state. Memory states are represented by memory formulae in the following form.

Definition 5.1 (Memory formula)
The formula $\varphi_{mem}(\gamma, t)$ obtained by replacing all occurrences in $\varphi_p(\gamma, t)$ of subformulae of the form state(γ, t')|= p by state(γ, t)|= memory(t', p) is called the memory formula for $\varphi_p(\gamma, t)$.

Thus, a memory formula defines a sequence of past events (i.e., a history of observations of an external world and actions) for the present time point t. The memory formula is no state formula yet. To obtain a memory state formula, normalization of the memory formula for $\varphi_p(\gamma, t)$ is performed by using Lemma 5.1 below. This Lemma will also be used to obtain other types of state formulae in Sections 6 and 7.

Lemma 5.1 (Normalization to State Formula)
Let t be a given time point. If a formula $\delta(\gamma, t)$ only contains temporal relations such as $t' < t''$ and $t' \leq t''$, and atoms of the form state(γ, t) |= p for some state formula p, and the given time point t, then some state formula q(t) can be constructed such that $\delta(\gamma, t)$ is equivalent to the formula $\delta^*(\gamma, t)$ of the form state(γ, t) |= q(t).

Definition 5.2 (Normalized Memory State Formula)
The state formula constructed by Lemma 5.1 for a memory formula $\varphi_{mem}(\gamma, t)$ is called *the (normalized) memory state formula for* $\varphi_{mem}(\gamma, t)$ and denoted by $q_{mem}(t)$. Moreover, q_{mem} is the state formula $\forall u'$ [present_time(u') $\rightarrow q_{mem}(u')$].

The normalized memory state formula for $\varphi_{mem}(\gamma, t)$ uniquely describes the present state at the time point t by a certain history of events. For the considered example $q_{mem}(t)$ for $\varphi_{mem}(\gamma, t)$ is specified as:

$$\exists u2 \leq t \; memory(u2, communicated(request\_from\_to\_for(A, B, IC)))$$

Lemma 5.2 (Memory Formula and Memory State Formula)
If time has the properties correctness and uniqueness, then

$$\varphi_{mem}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem}$$

Additionally, memory state persistency properties are composed for all memory atoms. Rules that describe creation and persistence of memory atoms are given in the executable theory from observation states to memory states $Th_{o\rightarrow m}$ described in Definition 2.3.

Definition 5.3 (Executable Theory from Interaction to Memory $Th_{o\rightarrow m}$)
For a given $\varphi(\gamma, t)$ the executable theory from observation states to memory states $Th_{o\rightarrow m}$ consists of the following formulae.

For any atom p occurring in $\varphi_p(\gamma, t)$, expressed in the InteractionOnt(A):

$$\forall t' \; state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models memory(t', p)$$
$$\forall t'' \; state(\gamma, t'') \models memory(t', p) \Rightarrow state(\gamma, t''+1) \models memory(t', p)$$
$$state(\gamma, 0) \models present\_time(0)$$
$$\forall t \; state(\gamma, t) \models present\_time(t) \Rightarrow state(\gamma, t+1) \models present\_time(t+1)$$

The last two rules are assumed to be included into the two theories $Th_{m\rightarrow p}$ and $Th_{p\rightarrow o}$ defined in subsequent sections as well.

For the example the rules for creation and persistence of memory atoms are specified as follows:

$$\forall t' \; state(\gamma, t', input(B)) \models communicated(request\_from\_to\_for(A, B, IC)) \Rightarrow$$
$$state(\gamma, t', internal(B)) \models memory(t', communicated(request\_from\_to\_for(A, B, IC)))$$
$$\forall t'' \; state(\gamma, t'', internal(B)) \models memory(t', communicated(request\_from\_to\_for(A, B, IC))) \Rightarrow$$
$$state(\gamma, t''+1, internal(B)) \models memory(t', communicated(request\_from\_to\_for(A, B, IC)))$$

The following Proposition expresses in what sense the executable theory guarantees that memory states are created that are correct.

Proposition 5.1 (Relating Past Formula and Memory State)
Let $\varphi_p(\gamma, t)$ be a past statement for a given t, $\varphi_{mem}(\gamma, t)$ the memory formula for $\varphi_p(\gamma, t)$, $q_{mem}(t)$ the normalized memory state formula for $\varphi_{mem}(\gamma, t)$, and $Th_{o\rightarrow m}$ the executable theory from the interaction states for $\varphi_p(\gamma, t)$ to the memory states. Then,

$$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$$

and

$$Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \;\; \& \;\; state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem}].$$

6 From Memory States to Preparation States

This section describes the executable theory for the basic steps from memory states to preparation states. First the $\varphi_{cond}(\gamma, t, t_1)$ part of the future formula in the behavioural specification is taken and encoded in memory state in a similar manner as the past formula was handled in Section 5.

Definition 6.1 (Condition Memory Formula)
Obtain the *condition memory state formula* $\varphi_{cmem}(\gamma, t, t_1)$ by replacing all occurrences in $\varphi_{cond}(\gamma, t, t_1)$ of state(γ, t') |= p by state(γ, t_1) |= memory(t', p).

The condition memory formula $\varphi_{cmem}(\gamma, t, t_1)$ describes a history of events, between the time point t, when $\varphi_p(\gamma, t)$ is true and the time point t_1, when the formula $\varphi_{cond}(\gamma, t, t_1)$ becomes true.

Definition 6.2 (Normalized Condition State Formula)
The state formula constructed by Lemma 5.1 for the condition memory formula $\varphi_{cmem}(\gamma, t, t_1)$ is called *the (normalized) condition state formula for* $\varphi_{cmem}(\gamma, t, t_1)$ and denoted by $q_{cond}(t, t_1)$. Moreover, $q_{cond}(t)$ is the state formula $\forall u'$ [present_time(u') $\rightarrow q_{cond}(t, u')$].

Lemma 6.1 (Condition Memory Formula and Condition State Formula)
If time has the properties correctness and uniqueness, then

$$\varphi_{cmem}(\gamma, t, t_1) \Leftrightarrow \text{state}(\gamma, t_1) \models q_{cond}(t, t_1) \;\&\; \text{state}(\gamma, t_1) \models q_{cond}(t, t_1) \Leftrightarrow \text{state}(\gamma, t_1) \models q_{cond}(t)$$

For the considered example $q_{cond}(t, t1)$ for $\varphi_{cmem}(\gamma, t)$ is obtained as: memory(t1, observed(provided_result_from_to(E, B, IC))) and $q_{cond}(t)$: $\forall u'$ [present_time(u') $\rightarrow$ memory(u', observed(provided_result_from_to(E, B, IC)))].

Next the $\varphi_{bh}(\gamma, t_1)$ part of the future formula is considered.

Definition 6.3 (Preparation Formula)
Obtain the *preparation formula* $\varphi_{prep}(\gamma, t_1)$ by replacing in $\varphi_{bh}(\gamma, t_1)$ any occurrence of state(γ, t_1+c) |= output(a) for some number c and output a by state(γ, t_1) |= preparation_for(output(t_1+c, a)).

The preparation state is created at the same time point t_1, when the condition $\varphi_{cond}(\gamma, t, t_1)$ for an output is true.

Definition 6.4 (Normalized Preparation State Formula)
The state formula constructed by Lemma 5.1 for the preparation formula $\varphi_{prep}(\gamma, t_1)$ is called *the (normalized) preparation state formula for* $\varphi_{prep}(\gamma, t_1)$ and denoted by $q_{prep}(t_1)$. Moreover, q_{prep} is the state formula $\forall u'$ [present_time(u')] $\rightarrow q_{prep}(u')$]

For the considered example $q_{prep}(t_1)$ is composed as preparation_for(output(t1+c, communicated(message_from_to(B, A, IC)))).

Lemma 6.2 (Preparation Formula and Preparation State Formula)
If time has the properties correctness and uniqueness, then

$$\varphi_{prep}(\gamma, t_1) \Leftrightarrow \text{state}(\gamma, t_1) \models q_{prep}(t_1) \;\&\; \text{state}(\gamma, t_1) \models q_{prep}(t_1) \Leftrightarrow \text{state}(\gamma, t_1) \models q_{prep}$$

Definition 6.5 (Conditional Preparation Formula)
Let $q_{cond}(t, t_1)$ be the normalized condition state formula for $\varphi_{cmem}(\gamma, t, t1)$ and $q_{prep}(t_1)$ the normalized preparation state formula for $\varphi_{prep}(\gamma, t_1)$. The formula $\varphi_{cprep}(\gamma, t)$ of the form

$$state(\gamma, t) \models \forall u_1 > t\ [q_{cond}(t, u_1) \rightarrow q_{prep}(u_1)]$$

is called the *conditional preparation formula* for $\varphi_f(\gamma, t)$.

Definition 6.6 (Normalized Conditional Preparation State Formula)
The state formula $\forall u_1 > t\ [\ q_{cond}(t, u_1) \rightarrow q_{prep}(u_1)\ ]$ is called *the normalized conditional preparation state formula for* $\varphi_{cprep}(\gamma, t)$ and denoted by $q_{cprep}(t)$. Moreover, q_{cprep} is the formula $\forall u'\ [\ present\_time(u')\ \rightarrow\ q_{cprep}(u')\ ]$.

Lemma 6.3 (Conditional Preparation and Conditional Preparation State Formula)
If time has the properties correctness and uniqueness, then

$$\varphi_{cprep}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t)\ \ \&\ \ state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}$$

Rules, which describe generation and persistence of condition memory states, a transition from the condition to the preparation state, and the preparation state generation and persistence, are given in *the executable theory from memory states to preparation states* $Th_{m\rightarrow p}$.

Definition 6.7 (Executable Theory From Memory to Preparation $Th_{m\rightarrow p}$)
For any state atom p occurring in $\varphi_{cond}(\gamma, t, t_1)$, expressed in the $InteractionOnt(A)$[1]:

$\forall t'\ state(\gamma, t') \models p\ \Rightarrow\ state(\gamma, t')\ \models [\ memory(t', p) \wedge stimulus\_reaction(p)\ ]$
$\forall t'', t'\ state(\gamma, t'')\ \models memory(t', p) \Rightarrow\ state(\gamma, t''+1)\ \models memory(t', p)$
$\forall t'\ state(\gamma, t')\ \models q_{mem} \Rightarrow\ state(\gamma, t')\ \models q_{cprep}$
$\forall t', t\ state(\gamma, t')\ \models [q_{cprep} \wedge q_{cond}(t) \wedge \wedge_p stimulus\_reaction(p)\ ] \Rightarrow\ state(\gamma, t')\ \models q_{prep}$
$\forall t'\ state(\gamma, t')\ \models [\ stimulus\_reaction(p) \wedge \neg\ preparation\_for(output(t'+c, a))\ ] \Rightarrow\ state(\gamma, t'+1)\ \models stimulus\_reaction(p)$
$\forall t'\ state(\gamma, t')\ \models [\ preparation\_for(output(t'+c, a)) \wedge \neg\ output(a)\ ] \Rightarrow\ state(\gamma, t'+1)\ \models preparation\_for(output(t'+c, a))$
$\forall t'\ state(\gamma, t') \models [\ present\_time(t') \wedge [\ present\_time(u') \rightarrow preparation\_for(output(u'+c), a)\ ]\ ] \Rightarrow$
$state(\gamma, t')\ \models preparation\_for(output(t'+c), a)$

where a is an action or a communication for which $state(\gamma, t'+c)\ \models output(a)$ occurs in $\varphi_f(\gamma, t)$.

Note that the last rule in the theory can be derived from other rules of the theory and lemmas, and was introduced only for convenience purposes to support the following proofs.

The auxiliary functions stimulus_reaction(a) are used for reactivation of agent preparation states for generating recurring actions or communications.

For the considered example:

$\forall t'\ [\ state(\gamma,t',input(B))\models observed(provided\_result\_from\_to(E,B,IC)) \Rightarrow$
$state(\gamma,t',internal(B))\models [\ memory(t',observed(provided\_result\_from\_to(E,B,IC))) \wedge$
$stimulus\_reaction(observed(provided\_result\_from\_to(E,B,IC)))]\ ]$

[1] If a future formula does not contain a condition, then stimulus_reaction atoms are generated from the corresponding past formula

$\forall t''$ state(γ,t'',internal(B)) |= memory(t',observed(provided_result_from_to(E,B,IC))) $\Rightarrow$
state(γ, t''+1, internal(B)) |= memory(t', observed(provided_result_from_to(E,B,IC)))

$\forall t'$ state(γ,t') |= $\forall u''$ [present_time(u'')$\rightarrow \exists u2$ [
memory(u2,communicated(request_from_to_for(A,B,IC)))]] $\Rightarrow$
state(γ, t')|= $\forall u'''$[present_time(u''')$\rightarrow$ [$\forall u1>u'''$ [memory(u1,
observed(provided_result_from_to(E,B,IC)))$\rightarrow$
preparation_for(output(u1+c,communicated(message_from_to(B,A,IC))))]]]

$\forall t',t$ state(γ,t')|=[$\forall u'''$ [present_time(u''')$\rightarrow$ [$\forall u1>u'''$ [
memory(u1,observed(provided_result_from_to(E,B,IC))) $\rightarrow$
preparation_for(output(u1+c,communicated(message_from_to(B,A,IC))))]]] $\wedge$
$\forall u''$ [present_time(u'') $\rightarrow$ memory(u'',observed(provided_result_from_to(E,B,IC)))] $\wedge$
stimulus_reaction(observed(provided_result_from_to(E,B,IC)))] $\Rightarrow$
state(γ,t',internal(B)) |= $\forall u1$ [present_time(u1)$\rightarrow$
preparation_for(output(u1+c,communicated(message_from_to(B,A,IC))))]

$\forall t'$ state(γ,t') |= [stimulus_reaction(observed(provided_result_from_to(E,B,IC))) $\wedge$
not(preparation_for(output(t'+c,communicated(message_from_to(B,A,IC))))] $\Rightarrow$
state(γ,t'+1) |= stimulus_reaction(observed(provided_result_from_to(E, B, IC)))

$\forall t'$ state(γ,t',internal(B)) |=
[preparation_for(output(t'+c,communicated(message_from_to(B,A,IC)))) $\wedge$
not(output(communicated(message_from_to(B,A,IC))))] $\Rightarrow$
state(γ,t'+1,internal(B)) |=
preparation_for(output(t'+c,communicated(message_from_to(B,A,IC)))).

Proposition 6.1 (Relating Preparation Formula and Preparation State Formula)
Let $\varphi_f(\gamma, t)$ be a future statement for t of the form $\forall t_1>t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$, where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement, which describes a condition for one or more actions and/or communications and $\varphi_{bh}(\gamma, t_1)$ is a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed; let $\varphi_{prep}(\gamma, t_1)$ be the preparation formula, $\varphi_{cprep}(\gamma, t)$ be the conditional preparation formula for $\varphi_f(\gamma, t)$, $q_{cprep}(t)$ be the normalized conditional preparation state formula for $\varphi_{cprep}(\gamma, t)$, and $Th_{m\rightarrow p}$ the executable theory for $\varphi(\gamma, t)$ from memory states to preparation states. Then,

$Th_{m\rightarrow p} \models \forall t_1>t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow \varphi_{cprep}(\gamma, t)]$

and

$Th_{m\rightarrow p} \models [\forall t_1>t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow$ state(γ, t) |= $q_{cprep}(t)$ &
state(γ, t) |= $q_{cprep}(t) \Leftrightarrow$ state(γ, t) |= q_{cprep}].

7 From Preparation States to Output States

The preparation state preparation_for(output(t_1+c, a)) is followed by the output state, created at time point t_1+c. Rules that describe a transition from preparation to output state(s) are given in *the executable theory from the preparation to the output state(s)* $Th_{p\rightarrow o}$.

Definition 7.2 (Executable Theory from Preparation to Output $Th_{p\rightarrow o}$)
For a given $\varphi_f(\gamma, t)$ the executable theory from the preparation to the output state(s) consists of the formula

$$\forall t'\ state(\gamma, t') \models preparation\_for(output(t'+c, a)) \Rightarrow state(\gamma, t'+c) \models output(a)$$

where c is a number and a an action or a communication for which $state(\gamma, t'+c) \models output(a)$ occurs in $\varphi_f(\gamma, t)$.

For the considered example the following rule is generated:

$$\forall t'\ state(\gamma, t', internal(B)) \models preparation\_for(output(t'+c, communicated(message\_from\_to(B, A, IC)))) \Rightarrow state(\gamma, t'+c, output(B)) \models output(communicated(message\_from\_to(B, A, IC))).$$

Proposition 7.1 (Relating Preparation Formula and Behaviour Formula)
Let $\varphi_{bh}(\gamma, t_1)$ be a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed, $\varphi_{prep}(\gamma, t_1)$ be the preparation formula and $Th_{p\to o}$ the executable theory from preparation states to output states. Then,

$$Th_{p\to o} \models [\ \varphi_{prep}(\gamma, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)\ ]$$

8 Combining the Executable Theories to Obtain the Refinement

In this section the sets of executable properties, identified in Sections 5 to 7 are combined to construct the specification $\pi(\gamma,t) = Th_{o\to m} \cup Th_{m\to p} \cup Th_{p\to o}$ (considered as conjunction), which describes a refinement of $\varphi(\gamma,t)$. The following theorem proves the existence of such a refinement for every external behavioural specification.

Theorem 8.1 (Existence of Executable Refinement)
Every external behavioural specification can be refined into an executable specification. To obtain such a refinement, the automated transformation procedure can be used as described in Sections 5 to 7.

9 Some Implementation Details

To automate the proposed procedure the software tool was developed in Java™. A model that describes dynamics of a system using an interaction ontology should be provided as an ASCII text file with name input.txt in the directory with the translation tool. All dynamic properties should be specified in the format [past formula] implies [future formula], where [future formula] is of the form [conditional formula] implies [action formula]. If the future formula contains a trivial condition (which is always true), then this condition can be omitted, and the corresponding dynamic property should be specified as [past formula] implies [action formula].

In order to enter a new dynamic property into the input file, first the past formula should be entered, then <new line symbol> and the future formula should be entered. If the specified property is not the last one in the specification, then <new line symbol> followed by a combination of symbols "---" and one more <new line symbol> has to be added. More specific technical details for specifying dynamic properties are given below.

- All time variables should be named as t[index], where [index] is a natural number.
- Time variables of both past and future formulae should be related to t, which is a standard variable and should not be additionally introduced (the present time point, with respect to which the formula is being evaluated).
- Names of state atoms should not contain blanks; no white spaces are allowed in predicate expressions.
- There are a number of standard predicates defined:
  - world(t,a): denotes an event a in the external world at a time point t
  - observed(t,a): denotes an observation a of an agent at a time point t
  - communicated(t, a): denotes a communication act a of an agent performed at a time point t
  - output(t,a): denotes an output a at a time point t
- Formulae are built using the following logical connectives and quantifiers:
  - AND: denotes the logical "and"
  - THEN: denotes the logical implication
  - not_a: denotes the negation of an atom a (note that negations can be also applied to the predicates, e.g., not_world(t,a), not_observation(t,not_a))
  - '[',']': denote brackets for formulas (note that brackets should be always separated by a single blank from the literals, which stand before and after them)
  - At1: denotes a universally quantified variable t1
  - Et1: denotes an existentially quantified variable t1
  - , : denotes a coma (make notice that there should be no blanks between a coma and literals before and after it).
- Other logical connectives can be expressed by means of already mentioned ones.

For example,

Past formula: Et3 t3<t observation(t3,a)

Future formula: At1 t1>=t observation(t1,b) THEN action(t1+2,c)

The transformation algorithm searches in the input file for the standard predicate names and the predefined structures, then performs string transformations that correspond precisely to the described steps of the translation procedure, and adds executable rules to the output specification file. In particular, for every observed atom from past and condition state formulae corresponding memory state generation and memory state persistence rules are formed. During the transformation of dynamic properties into corresponding rules of the executable theory $Th_{o\to m}$ in the expressions for q_{mem} and q_{cprep}, time variables t[index] are replaced by local time variables u[index], where [index] is a natural number. Additionally, for every observed atom from condition state formulae, a rule for generating stimulus_reaction atom and a stimulus reaction state persistence rule are created. Furthermore, for every output atom the preparation state and the output state generation rules are created. When transformation is finished, the output.txt file with the resulted executable specification is generated.

The transformation tool works on any platform running JRE 1.4 or higher. The processor capacity and the amount of RAM do not bear considerable influence on the time to generate an executable specification. In particular, on a computer system with the Intel Pentium III 850 MHz and 128 Mb RAM, the executable specification for the example considered in this paper was generated in 0.53 seconds.

10 Applications: Translating the Executable Format into Various Formats

Although the executable format obtained is very general, it has much in common with a number of particular executable languages and logics. In this paper we shall consider a number of them: the LEADSTO language, propositional modal temporal logic, monodic first-order temporal logic, and the loosely guarded fragment of first-order predicate logic. These logics have as advantages good computational properties or decidability. For all of these languages and logics dedicated techniques and tools for performing different types of analysis (e.g., by simulation or by verification) are available. The executable format obtained here is very close to these languages, but nevertheless is generic in the sense that it does not commit to one of them. Therefore, it is easy to make use of these techniques and tools, by simple translations of executable specifications into the considered languages and logics. First, some general translation principles will be described, applicable to all considered languages and logics. Then, more specific translation techniques for the particular languages and logics will be presented and illustrated by examples.

Since most of the considered languages do not allow function symbols, first all functions $f: y\ x\ z \rightarrow v$ in executable specifications and in state properties in particular are replaced by predicates combined_of(f, v, y, z). Furthermore, the holds-relation (|=) in TTL expressions is used as a predicate holds_at(X, γ, t) without function symbols in the arguments, which denotes that the state property X holds at time point t in the trace γ. Moreover, when occurring, a universal quantification (over a finite domain) in a state property is replaced by a conjunction of propositions and similarly an existential quantification is replaced by a disjunction of propositions. More specifically,

$\forall x: \text{SORT}\ P(x)$ is replaced by $\bigwedge_{a \in \text{SORT}} P(a)$

$\exists x: \text{SORT}\ P(x)$ is replaced by $\bigvee_{a \in \text{SORT}} P(a)$.

10.1 Translation Into LEADSTO Format

The LEADSTO language (Bosse *et al.*, 2007) is an executable fragment of order-sorted logic and a sublanguage of TTL. It models direct temporal or causal dependencies between two state properties in states at different points in time as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h real or integer numbers (constants of sort VALUE). A LEADSTO expression $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$, holds for a trace γ if:

$\forall t1\ [\forall t\ [t1–g \leq t < t1 \Rightarrow \alpha \text{ holds in } \gamma \text{ at time } t\ ] \Rightarrow \exists d\ [e \leq d \leq f\ \&\ \forall t'\ [t1+d \leq t' < t1+d+h \Rightarrow \beta \text{ holds in } \gamma \text{ at time } t'\ ]$

The introduced earlier types of executable TTL formulae can easily be translated into the LEADSTO format as shown in Table 1.

Table 1. Translation of executable formulae into LEADSTO format

Executable TTL formulae	Corresponding LEADSTO translation
$\forall t$ state(γ,t) |= X $\Rightarrow$ state(γ,t+c) |= Y	X $\rightarrow_{c-1, c-1, 1, 1}$ Y
$\forall t$ state(γ,t) |= X $\Rightarrow$ state(γ,t+1) |= X	X $\rightarrow_{0, 0, 1, 1}$ X
$\forall t$ state(γ,t) |= X $\Rightarrow$ state(γ,t) |= Y	X $\rightarrow_{-1, -1, 1, 1}$ Y

As an illustration, consider the following two examples of translation of properties from executable theories into the LEADSTO format.

1. The transition property from the a preparation to an action state

$\forall t'$ state(γ, t') \|= preparation_for(output(t'+c, a)) $\Rightarrow$ state(γ, t'+c) \|= output(a)

is translated into:

preparation_for(y2) & combined_of(output, y2, y1, a) & combined_of(plus, y1, t', c) $\twoheadrightarrow_{c-1, c-1, 1, 1}$ output(a)

2. The persistence property for the stimulus_reaction atom

$\forall t'$ state(γ, t') \|= [stimulus_reaction(p) $\wedge$ $\neg$ preparation_for(output(t'+c, a))] $\Rightarrow$ state(γ, t'+1) \|= stimulus_reaction(p)

is translated into the LEADSTO expression:

stimulus_reaction(p) & $\neg$preparation_for(y2) & combined_of(output, y2, y1, a) & combined_of(**plus**, y1, t', c)]
$\rightarrow_{0, 0, 1, 1}$ stimulus_reaction(p)

A specification in LEADSTO format has as an advantage that it can be easily depicted graphically, in a causal graph or system dynamics style. Furthermore, based on specifications in LEADSTO format, using the dedicated software environment, simulations of different scenarios can be performed and predicate logical dynamic properties can be automatically checked with respect to the generated simulation traces.

10.2 Translation Into Propositional Modal Temporal Logic

Propositional modal temporal logic (Benthem, 1995; Fisher, 1996, 2005) has been extensively used in the area of computer science to formalize the temporal development of a system. This logic can be seen as an extension of classical propositional logic by temporal operators, for a linear discrete time frame (e.g., '$\circ$', meaning "at the next moment in time", '$\Box$' meaning "at every future moment", '$\Diamond$' meaning "at some future moment"). The executable formulae of three types are translated into the propositional modal temporal logic as shown in Table 2.

Table 2. Translation of executable formulae into propositional modal temporal logic

Executable TTL formulae	Corresponding propositional modal temporal logic translation
$\forall t\ state(\gamma,t) \models X \Rightarrow state(\gamma,t+c) \models Y$	$\Box (X \Rightarrow \circ_c Y)^*$
$\forall t\ state(\gamma,t) \models X \Rightarrow state(\gamma,t+1) \models X$	$\Box (X \Rightarrow \circ X)$
$\forall t\ state(\gamma,t) \models X \Rightarrow state(\gamma,t) \models Y$	$\Box (X \Rightarrow Y)$

* $\circ_c$ is the contracted form that denotes c executable rules in form $X' \Rightarrow \circ Y'$, which describe c intermediate transitions between the state in which X holds and the state in which Y becomes true; notice that this requires that c-1 intermediate state properties are added to the state ontology to represent these intermediate states.

As a first example, the executable property

$$\forall t'\ state(\gamma, t') \models preparation\_for(output(t'+c, a)) \Rightarrow state(\gamma, t'+c) \models output(a)$$

with domain $D_{ACTION} = \{a1, a2\}$ is translated into two propositional modal temporal logic formulae:

$$\Box\ (preparation\_for(output(a1)) \Rightarrow \bigcirc_c\ output(a1))$$

$$\Box\ (preparation\_for(output(a2)) \Rightarrow \bigcirc_c\ output(a2))$$

Note that some state properties contain variables (e.g., in memory functions) over sort LTIME, whereas in modal temporal logic time is not explicitly available. To allow translation of such properties into propositional modal temporal logic, the predicate present_time is added to the state ontology and the domain for sort LTIME is explicitly defined. For example, the memory state generation property

$$\forall t'\ state(\gamma, t') \models p \Rightarrow state(\gamma, t') \models [\ memory(t', p) \wedge stimulus\_reaction(p)\ ]$$

with the domains $D_{LTIME} = \{1,2,3\}$, $D_{EVENT} = \{p1, p2\}$ is translated into a propositional modal temporal logic formula as follows:

$$present\_time(1)\ \&\ p1 \Rightarrow memory(1, p1) \wedge stimulus\_reaction(p1)$$
$$present\_time(2)\ \&\ p1 \Rightarrow memory(2, p1) \wedge stimulus\_reaction(p1)$$
$$present\_time(3)\ \&\ p1 \Rightarrow memory(3, p1) \wedge stimulus\_reaction(p1)$$

Similar for the domain instance p2.

Although the obtained specification may look quite cumbersome, nevertheless, by applying automated verification techniques based on efficient temporal resolution methods, such specifications can be effectively processed and analyzed. Furthermore, executable properties translated into propositional modal temporal logic can be naturally represented in MetateM, a modelling language based on the direct execution of modal temporal logic statements. By means of the dedicated software tools simulation and analysis of MetateM specifications can be performed. However, the expressivity of propositional modal temporal logic is still limited. For practical purposes more compact and expressive representations are needed, such as, for example, suggested by first-order variants of temporal logic.

10.3 Translation into Monodic First-Order Temporal Logic

The first-order temporal logic (FOTL) is an extension of classical first-order logic with modal operators for a linear discrete time frame; e.g., (Hodkinson *et al*., 2000; Hustadt *et al*, 2005). The monodic fragment of FOTL consists of all formulae, in which quantifiers over the domain variables are applied to formulae with at most one

free temporal variable. The three types of formulae defined in the executable format comply with this requirement. In general, translation into this first order temporal logic is similar to one given in Table 2. Furthermore, since the monodic fragment does not include function symbols, the functions used for building state properties in TTL are to be replaced by the corresponding predicates, as shown earlier. Consider the following two examples.

1. The transition property from the a preparation to an action state

$$\forall t'\ state(\gamma, t') \models preparation\_for(output(t'+c, a)) \Rightarrow state(\gamma, t'+c) \models output(a)$$

is translated into

$$\Box\ [\ preparation\_for(y2)\ \&\ combined\_of(output, y2, y1, a)\ \&\ combined\_of(\mathbf{plus}, y1, t', c) \Rightarrow \circ_c\ output(a)]$$

2. The persistence property for the stimulus_reaction atom

$$\forall t'\ state(\gamma, t') \models [\ stimulus\_reaction(p) \wedge \neg\ preparation\_for(output(t'+c, a))\ ] \Rightarrow state(\gamma, t'+1) \models stimulus\_reaction(p)$$

is translated into

$$\Box\ [stimulus\_reaction(p)\ \&\ \neg preparation\_for(y2)\ \&\ combined\_of(output, y2, y1, a)\ \&\ combined\_of(\mathbf{plus}, y1, t', c) \Rightarrow \circ\ stimulus\_reaction(p)]$$

Specifications in monodic first-order temporal logic can be automatically verified using the dedicated theorem prover TeMP (Hustadt *et al*, 2005) that implements the resolution-based calculus for monodic first-order temporal logic.

10.4 Translation Into the Loosely Guarded Fragment of Predicate Logic

The loosely guarded fragment (Andreka, Benthem and Nemeti, 1998) is decidable and has good computational properties. Formulae in the loosely guarded fragment are specified in the form:

$$\exists y\ ((\alpha_1 \wedge \ldots \wedge \alpha_m) \wedge \psi(x, y))\ \text{ or }\ \forall y\ ((\alpha_1 \wedge \ldots \wedge \alpha_m) \rightarrow \psi(x, y))$$

where x and y are tuples of variables, $\alpha_1 \ldots \alpha_m$ are atoms that relativize a quantifier (the guard of the quantifier), and $\psi(x, y)$ is an inductively defined formula in the guarded fragment, such that each free variable of the formula is in the set of free variables of the guard. The formulae defined in the executable format are also formulae of the loosely guarded fragment of the first-order predicate logic with atomic guards specified by predicates holds_at(X, γ, t). For example, the transition property from the a preparation to an action state

$$\forall t'\ state(\gamma, t') \models preparation\_for(output(t'+c, a)) \Rightarrow state(\gamma, t'+c) \models output(a)$$

is translated into the loosely guarded fragment as follows

$$\forall t'\ [\ [holds(y3, \gamma, t')\ \&\ combined\_of(preparation\_for, y3, y2)\ \&\ combined\_of(output, y2, y1, a)\ \&\ combined\_of(\mathbf{plus}, y1, t', c)\ ] \rightarrow [\ holds\_at(y4, \gamma, y1)\ \&\ combined\_of(output, y4, a)\ ]]$$

Specifications in terms of the loosely guarded fragment can be effectively analyzed by resolution techniques implemented by theorem provers such as Bliksem (Nivelle, 1999).

11 Discussion

The approach to analyzing behaviour of a multi-agent system proposed in this paper is based on distinguishing dynamic properties of different aggregation levels. The behaviour at a given aggregation level can specified in some temporal logical language by a set of dynamic properties. As the behaviour of a system can be complex, specifications at higher aggregation levels in principle may involve complex temporal expressions. Analysis is often performed by simulation or by verification of (possible) logical consequences of a specification. However, performing simulations and determining logical consequences of complex temporal formulae is not easy in general. This complexity makes analysis on the basis of a higher level specification difficult. Software tools to support analysis need system specifications in a simple format describing the system's basic steps at a lower aggregation level. For that reason, to make analysis possible, often specifications at a lower aggregation level have to be created, which may be a tedious task. For example, to express one complex temporal relation, usually a large number of simpler specifications are needed.

To support analysis on the basis of a higher level specification, an automated procedure has been developed, which allows transformation of a behavioural specification of a certain aggregation level into an executable temporal specification at a lower aggregation level, as a refinement of the given higher level specification. Specification of multi-agent system behaviour at a higher aggregation level is much easier. The order-sorted logic based reified temporal language TTL provides an intuitive way of creating a specification of system dynamics, which by the proposed transformation process still can be automatically translated into a lower level specification, as shown here.

The complexity of the representation of the obtained executable model is linear in size of the behavioural specification. More specifically, the non-executable specification is related to the executable specification in the following linear way:

(1) For every communicated and observed function from a past and a conditional formulae from dynamic properties, a corresponding memory state creation and a memory state persistence rule are introduced;

(2) For every dynamic property being translated a conditional preparation generation rule is created. This rule contains q_{mem} and q_{cprep} formulae, for each of which a variable over sort LTIME is introduced. This variable is used as argument of the present_time function. Furthermore, all time variables from φ_{mem} are replaced in the corresponding q_{mem} by their local time counterparts (i.e., time variables over sort LTIME); the same applies for q_{cprep};

(3) For every dynamic property being translated a preparation state creation rule is generated. This rule contains $q_{cond}(t)$ and q_{prep} formulae, for which transformations with time variables similar to ones in (2) are applied;

(4) For every output atom (i.e., action or communication) specified in $\varphi_{bh}(\gamma, t_1)$ a preparation state persistence rule and an output state creation rule are introduced;

(5) For reactivation of agent preparation states the auxiliary variables and the update rules corresponding to communicated and observed functions from $\varphi_{prep}(\gamma, t_1)$ are introduced.

This shows that no serious representational complexity is added by the transformation.

Finally, one other observation can be made. The transformation can also be seen and used as a way to eliminate past aspects from temporal formulae. It is sometimes a point of discussion in how far the possibility to incorporate references to the past adds expressivity to a temporal language; e.g., (Hodkinson and Reynolds, 2005). The transformation given here shows on the one hand that the past elements can be eliminated, so one can say that no essential expressivity is added by using past elements. On the other hand, however, this elimination is not for free: the state ontology has to be extended seriously to achieve it, in line with Ashby (1960)'s remarks discussed in Section 2.

REFERENCES

Ashby, R. (1952/1960). *Design for a Brain*. Chapman & Hall, London. First edition 1952, second edition 1960.

Andreka, H., Benthem, J. van, and Nemeti, I. (1998) Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic* 27, pp. 217-274.

Benthem, J. van (1995). *Temporal Logic*. In: D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4, Oxford: Clarendon Press, pp. 241-350.

Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J.: A Language and Environment for Analysis of Dynamics by Simulation. International Journal of Artificial Intelligence Tools, 16: 435-464 (2007)

Damasio, A. (2000). The Feeling of What Happens: Body, Emotion and the Making of Consciousness. MIT Press, 2000.

Dennett, D.C. (1991). *Consciousness Explained*, Penguin Press, 1991.

Descartes, R. (1633). *The World or Treatise on Light*. Withdrawn from publication, 1633. In: Descartes, R., The World and Other Writings. (S. Gaukroger ed.) Cambridge Texts in the History of Philosophy. Cambridge University Press, 1998. See also translated version by M.S. Mahoney: URL: http://www.princeton.edu/~hos/mike/texts/descartes/world/world.htm.

Fisher, M. (2005). Temporal Development Methods for Agent-Based Systems, *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 10, 2005, pp. 41-66.

Fisher, M. (1996). A Temporal Semantics for Concurrent METATEM, *Journal of Symbolic Computation (Special Issue on Executable Temporal Logics)* 22(5):627-648, November/December 1996, Academic Press.

Galton, A. (2003). Temporal Logic. *Stanford Encyclopedia of Philosophy*, URL: http://plato.stanford.edu/entries/logic-temporal/#2.

Galton, A. (2006). Operators vs Arguments: The Ins and Outs of Reification. *Synthese*, vol. 150, 2006, pp. 415-441.

Gelder, T.J. van, and Port, R.F., (1995). It's About Time: An Overview of the Dynamical Approach to Cognition. In: Port, R.F., Gelder, T. van (eds.) (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass., pp. 1-43.

Giunti, M. (1995). Dynamical Models of Cognition. In: Port, R.F., Gelder, T. van (eds.). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass., 1995, pp. 549-572.

Hodkinson, I., and Reynolds, M. (2005). Separation - Past, Present and Future. In: We Will Show Them: Essays in Honour of Dov Gabbay, Vol 2. S. Artemov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb, and J. Woods (eds.), College Publications, 2005, pp. 117-142.

Hodkinson, I., Wolter, F., and Zakharyaschev, M. (2000). Decidable fragments of first-order temporal logics. *Annals of Pure and Applied Logic* 106, pp. 85–134.

Hustadt, U., Konev, B., Riazanov, A., and Voronkov, A. (2004). TeMP: A Temporal Monodic Prover. In: Basin, D. A., and Rusinowitch, M. (eds), *Proceedings of the Second International Joint Conference on Automated Reasoning IJCAR 2004*, LNAI 3097, Springer, pp. 326-330.

Jonker, C.M., and J. Treur. (2002). Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness, *International Journal of Cooperative Information Systems*, vol. 11, 2002, 51-92.

Kowalski, R., and M.A. Sergot. (1986). A logic-based calculus of events, *New Generation Computing*, vol. 4, 1986, pp. 67-95,

Laplace, P.S. (1825). *Philosophical Essays on Probabilities*. Springer-Verlag, New York, 1995. Translated by A.I. Dale from the 5th French edition of 1825.

Manzano, M., (1996). *Extensions of First Order Logic*, Cambridge University Press, 1996.

Marcio Cysneiros, L., and E. Yu. (2002). Requirements Engineering for Large-Scale Multi-agent Systems. In: *Proceedings of the 1st International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS*), 2002, pp. 39-56.

Nivelle, H. de. (1999). The Bliksem Theorem Prover, Version 1.12. Max-Planck-Institut, Saarbruecken, Germany, 1999. (http://www.mpi-sb.mpg.de/~bliksem/manual.ps)

Port, R.F., Gelder, T. van (eds.) (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass.

Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, Cambridge MA: MIT Press, 2001.

Sharpanskykh, A., and Treur, J. (2005). Syntax and Semantics of the Temporal Trace Language, Technical Report No. TR-1801AI, Artificial Intelligence Department, Vrije Universiteit Amsterdam. http://www.few.vu.nl/~sharp/tr1801ai.pdf

Temple, G. (1942). *General principles of quantum theory*. 2nd ed. London: Methuen.

Appendix A

Lemma 4.1
Let $\varphi(\gamma, t)$ be a dynamic property expressed using the state ontology Ont. Then the following holds:

(1) coincide_on(γ_1, γ_2, Ont) & coincide_on(γ_2, γ_3, Ont) $\Rightarrow$ coincide_on(γ_1, γ_3, Ont)
(2) coincide_on(γ_1, γ_2, Ont) $\Rightarrow$ [$\varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)$].

Proof sketch.
The transitivity property (1) follows directly from the definition of coinciding traces for coincide_on(γ_1, γ_2, Ont) and coincide_on(γ_2, γ_3, Ont):

$\forall a \in STATATOM_{Ont}$ $\forall t'$ [state(γ_1, t') |= a $\Leftrightarrow$ state(γ_3, t') |= a] $\Rightarrow$ coincide_on(γ_1, γ_3, Ont)

From

$\forall \gamma_1, \gamma_2$ [coincide_on(γ_1, γ_2, Ont) $\Rightarrow$ $\forall t'$ [$\varphi_p(\gamma_1, t') \Leftrightarrow \varphi_p(\gamma_2, t')$ & $\varphi_f(\gamma_1, t') \Leftrightarrow \varphi_f(\gamma_2, t')$]]

follows that $\varphi(\gamma_1, t) \Leftrightarrow \varphi(\gamma_2, t)$.

Theorem 4.1
If the executable specification $\pi_A(\gamma, t)$ refines the external behavioural specification $\varphi_A(\gamma, t)$ of component A, and $\psi(\gamma, t)$ is a dynamic interaction property of component A in its environment, expressed using the interaction ontology InteractionOnt(A), then

$$[\forall \gamma \ [\pi_A(\gamma, t) \Rightarrow \psi(\gamma, t)]] \Leftrightarrow [\forall \gamma \ [\varphi_A(\gamma, t) \Rightarrow \psi(\gamma, t)]]$$

Proof sketch.
$\Leftarrow$ is direct:
from $\pi_i(\gamma, t) \Rightarrow \varphi_i(\gamma, t)$ and $\wedge\varphi_i(\gamma, t) \Rightarrow \psi(\gamma, t)$ it follows $\wedge\pi_i(\gamma, t) \Rightarrow \psi(\gamma, t)$.
$\Rightarrow$ runs as follows:
Suppose $\varphi_i(\gamma, t)$ holds for all i, then since $\pi_1(\gamma)$ refines $\varphi_1(\gamma, t)$, then according to the definition of refinement of an externally observable property exists such a γ_1 that $\pi_1(\gamma_1)$ and coincide_on(γ, γ_1, InteractionOnt (A)).
Due to Lemma 4.1, this γ_1 still satisfies all $\varphi_i(\gamma_1, t)$ (i.e., $\varphi_i(\gamma_1, t)$ holds for all i).
Proceed with γ_1 to obtain a γ_2 and further for all i to reach a trace γ_n, for which
$\pi_i(\gamma_n)$ holds for all i,
and
coincide_on(γ, γ_n, InteractionOnt(A)),
and
$\varphi_i(\gamma_n)$ holds for all i.

From
$\forall \gamma \forall i [\pi_i(\gamma) \Rightarrow \varphi_i(\gamma)]$,
and
$\forall \gamma$ [$\wedge\pi_i(\gamma) \Rightarrow \psi(\gamma, t)$]
it follows that $\forall \gamma \ \wedge\varphi_i(\gamma) \Rightarrow \psi(\gamma)$.
So it has been proven that $\forall \gamma \ \wedge\varphi_i(\gamma) \Rightarrow \psi(\gamma)$. ■

Lemma 5.1 (Normalization lemma)
Let t be a given time point. If a formula $\delta(\gamma, t)$ only contains temporal relations such as $t' < t''$ and $t' \leq t''$, and atoms of the form state(γ, t) |= p for some name of a state formula p, then some state

formula q(t) can be constructed such that $\delta(\gamma, t)$ is equivalent to the formula $\delta^*(\gamma, t)$ of the form $state(\gamma, t) \models q(t)$.

Proof sketch for Lemma 5.1.
First in the formula $\delta(\gamma, t)$ replace all temporal relations such as $t' < t''$ and $t' \le t''$ by $state(\gamma, t) \models t' < t''$ and $state(\gamma, t) \models t' \le t''$ respectively. Then proceed by induction on the composition of the formula $\delta(\gamma, t)$. Treat the logical connectives &, |, $\neg$, $\Rightarrow$, $\forall s$, $\exists s$.
1) conjunction: $\delta(\gamma, t)$ is $\delta 1(\gamma, t)$ & $\delta 2(\gamma, t)$
By induction hypothesis
$\delta 1(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$ (which is $\delta 1^*(\gamma, t)$)
$\delta 2(\gamma, t) \Leftrightarrow state(\gamma, t) \models p2$ (which is $\delta 2^*(\gamma, t)$)
Then
$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$ & $state(\gamma, t) \models p2 \Leftrightarrow state(\gamma, t) \models [\, p1 \wedge p2 \,]$ (which becomes $\delta^*(\gamma, t)$)

2) disjunction: $\delta(\gamma, t)$ is $\delta 1(\gamma, t)$ | $\delta 2(\gamma, t)$
Again by induction hypothesis
$\delta 1(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$ (which is $\delta 1^*(\gamma, t)$)
$\delta 2(\gamma, t) \Leftrightarrow state(\gamma, t) \models p2$ (which is $\delta 2^*(\gamma, t)$)
Then
$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1 \mid state(\gamma, t) \models p2 \Leftrightarrow state(\gamma, t) \models [\, p1 \vee p2 \,]$ (which becomes $\delta^*(\gamma, t)$)

3) negation: $\delta(\gamma, t)$ is $\neg\delta 1(\gamma, t)$
$\delta 1(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$
$\delta(\gamma, t) \Leftrightarrow \neg state(\gamma, t) \models p1$
$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models not(p1)$ (which is $\delta^*(\gamma, t)$)

4) implication: $\delta(\gamma, t)$ is $\delta 1(\gamma, t) \Rightarrow \delta 2(\gamma, t)$
Again by induction hypothesis
$\delta 1(\gamma, t) \Leftrightarrow state(\gamma, t) \models p1$ (which is $\delta 1^*(\gamma, t)$)
$\delta 2(\gamma, t) \Leftrightarrow state(\gamma, t) \models p2$ (which is $\delta 2^*(\gamma, t)$)
Then
$\delta(\gamma, t) \Leftrightarrow [state(\gamma, t) \models p1 \Rightarrow state(\gamma, t) \models p2] \Leftrightarrow state(\gamma, t) \models [p1 \rightarrow p2]$ (which becomes $\delta^*(\gamma, t)$)

5) universal quantifier:
$\delta(\gamma, t) \Leftrightarrow \forall t'\ state(\gamma, t) \models p1(t')$
$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models \forall u'\ p1(u')$ (which is $\delta^*(\gamma, t)$)

6) existential quantifier:
$\delta(\gamma, t) \Leftrightarrow \exists t'\ state(\gamma, t) \models p1(t')$
$\delta(\gamma, t) \Leftrightarrow state(\gamma, t) \models \exists u'\ p1(u')$ (which becomes $\delta^*(\gamma, t)$)

Lemma 5.2
If time has properties of correctness and uniqueness, then
$$\varphi_{mem}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem} \quad (1)$$
Proof.
The proof follows directly from Lemma 5.1, definitions of correctness and uniqueness of time and the definition of the formula q_{mem}.

Proposition 5.1
Let $\varphi_p(\gamma, t)$ be a past statement for a given t, $\varphi_{mem}(\gamma, t)$ the memory formula for $\varphi_p(\gamma, t)$, $q_{mem}(t)$ the normalized memory state formula for $\varphi_{mem}(\gamma, t)$, and $Th_{o\rightarrow m}$ the executable theory from the interaction states for $\varphi_p(\gamma, t)$ to the memory states. Then,

$Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$
and
$Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t) \,\&\, state(\gamma, t) \models q_{mem}(t) \Leftrightarrow state(\gamma, t) \models q_{mem}]$.

Proof.
From the definitions of $q_{mem}(t)$ and of $Th_{o\to m}$ follows
$Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow \varphi_{mem}(\gamma, t)]$
Further by Lemma 5.2
$Th_{o\to m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}(t)]$ ■

Lemma 6.1
If time has properties of correctness and uniqueness, then
$\varphi_{cmem}(\gamma, t, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{cond}(t, t_1) \,\&\, state(\gamma, t_1) \models q_{cond}(t, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{cond}(t)$ (2)

Proof.
The lemma can be proven in the same manner as Lemma 5.2.

Lemma 6.2
If time has properties of correctness and uniqueness, then
$\varphi_{prep}(\gamma, t_1) \Leftrightarrow state(\gamma, t_1) \models q_{prep}(t_1) \,\&\, state(\gamma, t_1) \models q_{prep}(t_1) \Leftrightarrow state(\gamma, t_1) \models q_{prep}$ (3)

Proof.
The lemma can be proven in the same manner as Lemma 5.2.

Lemma 6.3
If time has properties of correctness and uniqueness, then
$\varphi_{cprep}(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{cprep}(t) \,\&\, state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}$ (4)

Proof.
The lemma can be proven in the same manner as Lemma 5.2.

Proposition 6.1
Let $\varphi_f(\gamma, t)$ be a future statement for t of the form $\forall t_1{>}t\, [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$, where $\varphi_{cond}(\gamma, t, t_1)$ is an interval statement, which describes a condition for one or more actions and/or communications and $\varphi_{bh}(\gamma, t_1)$ is a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed; let $\varphi_{prep}(\gamma, t_1)$ be the preparation formula, $\varphi_{cprep}(\gamma, t)$ be the conditional preparation formula for $\varphi_f(\gamma, t)$, $q_{cprep}(t)$ be the normalized conditional preparation state formula for $\varphi_{cprep}(\gamma, t)$, and $Th_{m\to p}$ the executable theory for $\varphi(\gamma, t)$ from memory states to preparation states. Then,

$$Th_{m\to p} \models \forall t_1{>}t\, [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow \varphi_{cprep}(\gamma, t)]$$

and

$$Th_{m\to p} \models [\forall t_1{>}t\, [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow state(\gamma, t) \models q_{cprep}(t) \,\&\, state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}].$$

Proof.
From the definition of $Th_{m\to p}$, Lemmas 6.1 and 6.2, Definition 6.6 it follows that
$Th_{m\to p} \models \forall t_1{>}t\, [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow \varphi_{cprep}(\gamma, t)]$

Then, by Lemma 6.3

$Th_{m\to p} \models [\forall t_1{>}t\, [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)] \Leftrightarrow state(\gamma, t) \models q_{cprep}(t)$

and

$state(\gamma, t) \models q_{cprep}(t) \Leftrightarrow state(\gamma, t) \models q_{cprep}$
■

Proposition 7.1
Let $\varphi_{bh}(\gamma, t_1)$ be a (conjunction of) future statement(s) for t_1, which describes action(s) and/or communications that are to be performed, $\varphi_{prep}(\gamma, t_1)$ be the preparation formula and $Th_{p\rightarrow o}$ the executable theory from preparation states to output states. Then,
$Th_{p\rightarrow o} \models [\ \varphi_{prep}(\gamma, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)\ ]$

Proof.
Follows directly from the definition of $Th_{p\rightarrow o}$ ■

Theorem 8.1
Every external behavioural specification can be refined into an executable specification.

Proof.
According to the Definition 4.4 an executable specification $\pi(\gamma, t)$ refines an externally observable dynamic property $\varphi(\gamma, t)$ iff
(1) $\forall \gamma, t\ \ \pi(\gamma, t) \Rightarrow \varphi(\gamma, t)$
(2) $\forall \gamma_1, t\ [\ \varphi(\gamma_1, t) \Rightarrow [\ \exists \gamma_2\ \ coincide\_on(\gamma_1, \gamma_2, InteractionOnt(A)) \ \&\ \pi(\gamma_2, t)\ ]\ ]$

The first condition can be reformulated as $Th_{o\rightarrow m} \cup Th_{m\rightarrow p} \cup Th_{p\rightarrow o} \models \varphi(\gamma,t)$

Here $\varphi(\gamma,t)$ is of the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)\ ]$ with $\varphi_f(\gamma, t)$ future statement for t of the form $\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$. As a basis we use Propositions 5.1, 6.1 and 7.1. These propositions relate the past formula to the memory state formula, the preparation formula to the preparation state formula, and the preparation formula to the behaviour formula, respectively:

(1) $Th_{o\rightarrow m} \models [\varphi_p(\gamma, t) \Leftrightarrow state(\gamma, t) \models q_{mem}\ ]$
(2) $Th_{m\rightarrow p} \models [\ [\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)]\ \Leftrightarrow\ state(\gamma, t) \models q_{cprep}\ ]$
(3) $Th_{p\rightarrow o} \models [\ \varphi_{prep}(\gamma, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)\ ]$

Moreover, as this executable rule is included in $Th_{m\rightarrow p}$ (see Definition 6.7), it holds:

(4) $Th_{m\rightarrow p} \models \forall t'\ state(\gamma, t') \models q_{mem} \Rightarrow state(\gamma, t') \models q_{cprep}$

Based on these four lines, it can be seen that $\pi(\gamma,t)$ indeed satisfies the first criterion for refinement, by the following steps in the theory $\pi(\gamma,t) = Th_{o\rightarrow m} \cup Th_{m\rightarrow p} \cup Th_{p\rightarrow o}$:

- when $\varphi_p(\gamma, t)$ holds, also $state(\gamma, t) \models q_{mem}$ holds (1)
- when $state(\gamma, t) \models q_{mem}$ holds, also $state(\gamma, t) \models q_{cprep}$ holds (4)
- when $state(\gamma, t) \models q_{cprep}$ holds, also $\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)]$ holds (2)
- when $\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{prep}(\gamma, t_1)]$ holds, also $\forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$ holds (3)
- hence $\varphi_p(\gamma, t) \Rightarrow \forall t_1{>}t\ [\varphi_{cond}(\gamma, t, t_1) \Rightarrow \varphi_{bh}(\gamma, t_1)]$ holds, which is $\varphi(\gamma, t)$.

For the second criterion of refinement, first of all notice that:

- the consequents of the executable rules in $\pi(\gamma,t)$ always are atoms, never negations
- these consequent atoms are always internal atoms, except the consequent of the executable rule in $Th_{p\rightarrow o}$, which is the interaction atom $state(\gamma, t'{+}c) \models output(a)$.

For any trace γ, let

$diag(\gamma, Ont) = \{\ state(\gamma, t_1) \models a\ |\ \gamma(t_1) \models a\ \ \&\ \ a \text{ literal in } Ont\ \}$

Given these observations, let any trace γ be given such that $\varphi(\gamma, t)$ holds. Construct the trace γ' such that γ' is equal to γ for the interaction ontology, and complies to the executable rules for the internal atoms, as follows:

$\gamma'(t_1) \models b \Leftrightarrow \gamma(t_1) \models b$ for any interaction literal b

$\gamma'(t_1) \models a$ if a is an internal atom and

$$diag(\gamma, InteractionOnt) \cup \pi(\gamma,t) \models [\, state(\gamma, t_1) \models a \,]$$

$\gamma'(t_1) \models \neg a$ if a is an internal atom and

$$not\ diag(\gamma, InteractionOnt) \cup \pi(\gamma,t) \models [\, state(\gamma, t_1) \models a \,]$$

By this construction all executable rules hold for γ', except possibly the rule from $Th_{p\to o}$. This last rule is the remaining issue to be addressed. Suppose this rule does not hold for γ'. Then a t_1 exists such that the antecedent holds, but not the consequent:

$state(\gamma', t_1) \models preparation\_for(output(t_1+c, a))$ &
$not\ \ state(\gamma', t_1+c) \models output(a)$

From the construction of the trace γ' and Definition 6.3 it follows that the preparation atom $preparation\_for(output(t_1+c, a))$ is based on the occurrence of $state(\gamma, t_1+c) \models output(a)$ in $\varphi_{bh}(\gamma, t_1)$. Moreover, the preparation atom is derivable from $\pi(\gamma, t)$ so it originates from a condition formula and a memory state formula that both hold for γ'. By Propositions 5.1 and 6.1 it holds that

$\varphi_p(\gamma', t)$

$\varphi_{cond}(\gamma', t, t_1)$

Since these are formula based on InteractionOnt, and γ and γ' coincide on InteractionOnt, by Lemma 4.1(2) also

$\varphi_p(\gamma, t)$

$\varphi_{cond}(\gamma, t, t_1)$
hold.
As $\varphi(\gamma, t)$ holds, this implies that $\varphi_{bh}(\gamma, t_1)$ holds. Moreover, it was found that $state(\gamma, t_1+c) \models output(a)$ occurs in $\varphi_{bh}(\gamma, t_1)$. Therefore, $state(\gamma, t_1+c) \models output(a)$ holds, and again, since γ and γ' coincide on InteractionOnt, this implies that $state(\gamma', t_1+c) \models output(a)$ holds, which is a contradiction. This shows that the second criterion of refinement is fulfilled, which completes the proof of Theorem 8.1 ■

Chapter 4

Formal Modeling and Analysis of Cognitive Agent Behavior [1]

Abstract. From an external perspective, cognitive agent behavior can be described by specifying (temporal) correlations of a certain complexity between stimuli (input states) and (re)actions (output states) of the agent. From an internal perspective the agent's dynamics can be characterized by direct (causal) temporal relations between internal, mental states of the agent. The latter type of specifications can be represented in a relatively simple, executable format, which enables different types of analysis of the agent's behavior. In particular, simulations of the agent's behavior under different (environmental) circumstances can be explored. Furthermore, by applying verification techniques, automated analysis of the consequences of the agent's behavior can be carried out. To enable such types of analysis when only given an external behavioral specification, this has to be transformed first into some type of executable format. An automated procedure for such a transformation is proposed in this paper. The application of the transformation procedure is demonstrated for a number of cases, showing examples of the types of analysis as mentioned for different forms of behavior.

1 Introduction

The behavior of a cognitive agent can be considered both from an external and an internal perspective. From the external perspective, behavior of the agent can be

[1] Part of this chapter appeared as Sharpanskykh, A., Treur, J.: Modeling of Agent Behavior Using Behavioral Specifications. In: Fum, D., Missier, F. del, Stocco, A. (eds.): Proceedings of the 7th International Conference on Cognitive Modelling, ICCM'06, 280-286 (2006) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

described by temporal relationships of a certain complexity between its input (stimuli) and output (actions) states over time, expressed in some (temporal) language, without any reference to internal or mental states of the agent. Such relationships are called input-output correlations by Kim (1996, pp. 87-91). Within Philosophy of Mind such a view is considered within the perspective of behaviorism (Kim, 1996). The states of the agent are required to be publicly observable and the statements that describe these states should be intersubjectively verifiable (Heil, 2000). According to the apologists of behaviorism Watson (1913) and Skinner (1953), internal states of the agent (mental or inner states) are considered to be methodologically intractable and unnecessary, since they are based on a personal subjective experience and evaluations and can not be used for analysis and predictions of the agent behavior. Descriptions from an external perspective can be successfully used for modeling relatively simple types of behavior (e.g., stimulus-response behavior (Skinner, 1935)). For less simple types of behavior (e.g., adaptive behavior based on conditioning (Balkenius & Moren, 1999)) an external behavioral specification often consists of more complex temporal relations, relating behavior at a certain point in time to a possibly large number of inputs in the past (e.g., a training program), that can not be directly used for simulations or other types of analysis.

From the internal perspective the behavior of the agent can be characterized by a specification of more direct (causal) temporal relations between mental states of the agent, based on which an externally observable behavioral pattern is generated. Such a perspective is taken within functionalism (Kim, 1996). From this perspective mental states are described by their functional or causal roles. These can be specified in simple, executable formats. A mental state is characterized by its direct temporal or causal relations with input, output and other mental states. Functionalism was originally formulated by Putnam in terms of a 'Turing machine' (Putman, 1975), an abstract machine to give a mathematically precise definition of an algorithm or an automatic procedure. However, in general other executable (temporal) languages can be applied to specify functional roles.

From the viewpoint of analysis, executability is an important advantage of an internal specification over an external one. By means of executable specifications it is possible to perform automated support of the analysis of an agent's behavior, for example, by simulations of different scenarios of the agent's behavior or by verifying certain global properties of an agent in its environment. To enable automated analysis of an external behavioral specification, the possibly complex temporal relationships between input states and output stats over time have to be reformulated in terms of a simpler executable format. In practice, such a reformulation process is by no means trivial. For example, it may involve a certain creativity concerning additional intermediate states that have to be postulated and direct temporal relations between such states that have to be hypothesized. Moreover, to obtain certainty that the reformulated specification is equivalent in a certain sense to the original one (and does not describe just another process), is hardly possible by human activity only. To provide more support for this is the main problem on which this paper is focused.

The challenge addressed is to obtain a standard method for this reformulation process and to provide automated support for this method, with guaranteed outcome equivalent to the original specification. As a solution a standard procedure for (automated) transformation of the external behavioral specification first into a

synthetic executable specification using postulated intermediate states, and subsequently into a general state transition system format is proposed. The executable specification is based on direct executable temporal relations between certain (postulated) states. These states play roles comparable to sensory representation memory states and preparation states of an agent. The type of internal memory states of an agent used (and shown to suffice) are memory states based on the agent sensing (observations) of objects and processes in his/her environment and of his/her own behavior (e.g., actions). Furthermore, it is postulated that before performing an action an agent creates an internal preparation state. While simple types of agent behavior (e.g., variants of stimulus-response behavior) are based on a limited number of (unrelated) internal states, more complex types (e.g., motivation-based, goal-directed, adaptive) require more complex patterns of temporal relations between (multiple) internal states. In the approach presented this is addressed by allowing the memory states to represent complex temporal relations. So, they are used not only to represent world states, but also temporal patterns that occurred in the past. In this way reasoning of an agent about his/her previous experience enables to generate proactive, motivation-based or goal-directed behavior as well.

The justification that the proposed transformation method indeed provides an executable specification which is equivalent to the original specification, is based on the theorem (see Section 4) that an external behavioral specification entails any dynamic property if and only if the generated executable internal specification entails the same property.

Based on the generated executable specification, indeed different types of (automated) analysis can be performed. First, a developed simulation software tool applied to the generated transition system specification of the agent's behavior can be used to generate traces representing changes of internal (mental) states and actions of the agent over time, according to different environmental scenarios. Second, the generated transition system specification is also useful to analyze the consequences of the agent's behavior under such environmental scenarios.

For a given specification of externally observable behavior, an interesting but not easily solvable problem is how to determine the (logical) consequences of this behavior in different environmental circumstances. For example, if an animal has a certain behavioral repertoire with respect to different food-related circumstances in the environment, in how far will this repertoire be adequate in the sense that it entails the animal's well-being, i.e., in how far the animal will become satisfied and healthy due to this behavioral repertoire. Within Computer Science, quite useful and efficient model checking techniques have been developed to determine consequences of a given system specification; e.g., (Clarke & Grumberg & Peled, 1999). By performing model checking it is possible to determine automatically if a system model, usually specified in a transition system format, entails some dynamic property, specified by more complex temporal formulae. Using model checking techniques this paper contributes an automated approach for analyzing the consequences of a behavioral specification of an agent in its environment. To be able to use model checking techniques, a behavioral specification has to be given in a simple, executable format (as a transition system). To address this issue the proposed approach includes an automated procedure for the transformation of executable specifications of agent

behavior into the input format of the SMV model checking tool (McMillan, 1993) that is used for the analysis of logical consequences of the agent's behavior.

In the next section using the language for formal modeling of agent behavior introduced in (Sharpanskykh & Treur 2005) the transformation procedure from an external into an executable internal specification and subsequently into a general description of a finite state transition system is described in some detail. The explanation of the procedure is illustrated by a running example. After that the proposed approach is applied for a number of cases concerning different types of analysis of agent behavior. More specifically, in Section 3 simulation of different scenarios of agent behavior is considered, and in Section 4 an automated approach for the analysis of the consequences of an agent's behavior is described. The paper ends with a discussion.

2 Transformation into Executable Format

The procedure described in this section achieves the transformation of an external behavioral specification for an agent into executable format and subsequently into the representation of a finite state transition system.

Let $\varphi(\gamma, t)$ be a non-executable dynamic property from an external behavioral specification for agent A, expressed using ontology InteractionOnt(A), for which an executable representation should be found, then the transformation procedure is specified as follows.

The Transformation Procedure

(1) Identify executable temporal properties, which describe transitions from interaction states to memory states.
(2) Identify executable temporal properties, which describe transitions from memory states to preparation states for performing an action.
(3) Specify executable properties, which describe the transition from preparation states to the corresponding action performance states.
(4) From the executable properties, identified during the steps 1-3, construct a part of the specification $\pi(\gamma, t)$, which describes the internal dynamics of agent A, corresponding to the property $\varphi(\gamma, t)$.
(5) Apply the steps 1-4 to all properties in the external behavioral specification of the agent A. In the end add to the executable specification the dynamic properties, which were initially specified in executable form using an ontology, different than InteractOnt(A).
(6) Translate the identified during the steps 1-5 executable rules into the transition system representation.

The details of the described procedure are explained by means of an example, in which delayed-response behavior of a laboratory mouse is analyzed; e.g., Hunter (1912); Allen & Bekoff (1997).

Initial situation

The initial situation for the conducted experiment is as follows: the mouse is placed in front of a transparent screen that separates it from a piece of food that is put behind the screen. The mouse is able to observe the position of food and of the screen. At some moment after food has been put, a cup is placed covering the food, which makes food invisible for the mouse. After some time the screen is raised and the animal is free to go to any position. If the mouse comes to the position, where the food is hidden, then it will be capable to lift up the cup and get the food.

The behavioral specification for the conducted experiment consists of environmental properties and externally observable behavioral properties of the mouse. For the purposes of illustration of the proposed transformation procedure the dynamic property that describes the delayed-response behavior of the mouse has been chosen. Informally this property expresses that the mouse goes to the position with food if it observes that there is no screen and at some point in the past the mouse observed food and since then did not observe the absence of food. According to the definition of an external behavioral specification the considered property can be represented in the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)]$, where $\varphi_p(\gamma, t)$ is a formula

$\exists t2<t$ [state(γ, t2, input(mouse)) |= observed(food) $\wedge$
$\forall t3, t \geq t3 > t2$ state(γ, t3, input(mouse))|= not(observed(not(food)))]

and $\varphi_f(\gamma, t)$ is a formula

$\forall t4 > t$ [state(γ, t4, input(mouse)) |= observed(not(screen)) $\Rightarrow$
state(γ, t4+c, output(mouse)) |= performing_action(goto_food)]

with $\varphi_{cond}(\gamma, t, t4)$ is

state(γ, t4, input(mouse)) |= observed(not(screen))

and $\varphi_{act}(\gamma, t4)$ is

state(γ, t4+c, output(mouse)) |= performing_action(goto_food)

where t is the present time point with respect to which the formulae are evaluated.

Step 1. ***From interaction states to memory states***

General idea

The formula $\varphi_{mem}(\gamma, t)$ obtained by replacing all occurrences in $\varphi_p(\gamma, t)$ of subformulae of the form state(γ, t') |= p by state(γ, t) |= memory(t', p) is called the *memory formula for* $\varphi_p(\gamma, t)$.

Thus, a memory formula defines a sequence of past events (i.e., a history) (e.g., observations of an external world, actions) for the present time point t. The time interval for generation of an internal memory state of an agent from its observation is assumed to be incommensurably smaller than time intervals between external events (i.e., stimuli). Therefore, in the proposed model both an observation state and a corresponding memory state are created at the same time point.

By a rewriting process (for the formal details for the considered procedure we refer to Sharpanskykh. & Treur (2005)) $\varphi_{mem}(\gamma, t)$ is equivalent to some formula $\delta^*(\gamma, t)$ of the form state(γ, t) |= $q_{mem}(t)$, where $q_{mem}(t)$ is called *the normalized memory state formula for* $\varphi_{mem}(\gamma, t)$, which uniquely describes the present state at the time point t by a certain history of events. Moreover, q_{mem} is the state formula $\forall t'$ [present_time(t') $\Rightarrow$ $q_{mem}(t')$].

Example

For the considered example $q_{mem}(t)$ for $\varphi_{mem}(\gamma, t)$ is specified as:

$\exists t2$ [memory(t2, observed(food)) $\wedge$
$\forall t3, t \geq t3 > t2$ memory(t3, not(observed(not(food))))]

Additionally, memory state persistency properties are composed for all memory atoms. For example, for the atom memory(t2, observed(food)) the corresponding persistency property is defined as:

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', observed(food)) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', observed(food))

Rules that describe creation and persistence of memory atoms are given in *the executable theory from observation states to memory states* $Th_{o\to m}$. For the considered example:

$\forall t'$ state(γ, t', input(mouse)) |= observed(food) $\Rightarrow$
state(γ, t', internal(mouse)) |= memory(t', observed(food))

$\forall t'$ state(γ, t', input(mouse)) |= not(observed(not(food))) $\Rightarrow$
state(γ, t', internal(mouse)) |= memory(t', not(observed(not(food))))

$\forall t'$ state(γ, t', input(mouse)) |= observed(not(food)) $\Rightarrow$
state(γ, t', internal(mouse)) |= memory(t', observed(not(food)))

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', observed(food)) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', observed(food))

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', not(observed(not(food)))) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', not(observed(not(food))))

$\forall t''$ state(γ, t'', internal(mouse)) |= memory(t', observed(not(food))) $\Rightarrow$
state(γ, t''+1, internal(mouse)) |= memory(t', observed(not(food)))

Step 2. ***From memory states to preparation states***

General idea

Obtain $\varphi_{cmem}(\gamma, t, t_1)$ by replacing all occurrences in $\varphi_{cond}(\gamma, t, t_1)$ of state(γ, t') |= p by state(γ, t_1) |= memory(t', p). The condition memory formula $\varphi_{cmem}(\gamma, t, t_1)$ contains a history of events, between the time point t, when $\varphi_p(\gamma, t)$ is true and the time point t_1, when the formula $\varphi_{cond}(\gamma, t, t_1)$ becomes true. Again by a rewriting process $\varphi_{cmem}(\gamma, t, t_1)$ is equivalent to the formula state(γ, t_1) |= $q_{cond}(t, t_1)$, where $q_{cond}(t, t_1)$ is called *the normalized condition state formula for* $\varphi_{cmem}(\gamma, t, t_1)$. Moreover, $q_{cond}(t)$ is the state formula $\forall t'$ [present_time(t') $\Rightarrow$ $q_{cond}(t, t')$].

Example

For the considered example $q_{cond}(t, t4)$ for $\varphi_{cmem}(\gamma, t)$ is obtained as: memory(t4, observed(not(screen))) and $q_{cond}(t)$: $\forall t'$ [present_time(t') $\Rightarrow$ memory(t', observed(not(screen)))].

Obtain $\varphi_{prep}(\gamma, t_1)$ by replacing in $\varphi_{act}(\gamma, t_1)$ any occurrence of state(γ, t_1+c) |= performing_action(a) by state(γ, t_1) |= preparation_for(action(t_1+c, a)), for some number c and action a. The preparation state is created at the same time point t_1, when the condition for an action $\varphi_{cond}(\gamma, t, t_1)$ is true. By Lemma 1 $\varphi_{prep}(\gamma, t_1)$ is equivalent to the state formula state(γ, t_1) |= $q_{prep}(t_1)$, where $q_{prep}(t_1)$ is called *the normalized preparation state formula for* $\varphi_{cond}(\gamma, t_1)$. Moreover, q_{prep} is the state formula $\forall t'$ [present_time(t')] $\Rightarrow$ $q_{prep}(t')$]. For the considered example $q_{prep}(t4)$ is composed as preparation_for(action(t4+c, goto_food)).

Rules, which describe generation and persistence of condition memory states, a transition from the condition to the preparation state, and the preparation state generation and persistence, are given in *the executable theory from memory states to preparation states* $Th_{m\to p}$. For the considered example:

$\forall t'$ state(γ, t', input(mouse)) |= observed(not(screen)) $\Rightarrow$

state(γ, t', internal(mouse)) |= [memory(t', observed(not(screen))) ∧ stimulus_reaction(observed(not(screen)))]

∀t'' state(γ, t'', internal(mouse)) |= memory(t', observed(not(screen))) ⇒
state(γ, t''+1, internal(mouse)) |= memory(t', observed(not(screen)))

∀t' state(γ, t') |= ∀t'' [present_time(t'') → ∃t2 [memory(t2, observed(food)) ∧ ∀t3, t'' ≥ t3 > t2 memory(t3, not(observed(not(food))))]] ⇒
state(γ, t') |= ∀t''' [present_time(t''') → [∀t4 > t''' [memory(t4, observed(not(screen))) → preparation_for(action(t4+c, goto_food))]]]

∀t', t state(γ, t') |= [∀t''' [present_time(t''') → [∀t4 > t''' [memory(t4, observed(not(screen))) → preparation_for(action(t4+c, goto_food))]]] ∧
∀t'' [present_time(t'') → memory(t'', observed(not(screen)))] ∧ stimulus_reaction(observed(not(screen)))] ⇒
state(γ, t', internal(mouse)) |= ∀t4 [present_time(t4) → preparation_for(action(t4+c, goto_food))]

∀t' state(γ, t') |= [stimulus_reaction(observed(not(screen))) ∧ not(preparation_for(action(t'+c, goto_food)))] ⇒
state(γ, t'+1) |= stimulus_reaction(observed(not(screen)))

∀t' state(γ, t', internal(mouse)) |= [preparation_for(action(t'+c, goto_food)) ∧ not(performing_action(goto_food))] ⇒
state(γ, t'+1, internal(mouse)) |= preparation_for(action(t'+c, goto_food)).

The auxiliary atoms stimulus_reaction(a) are used to reactivate agent preparation states for generating recurring actions.

Step 3. ***From preparation states to action states***

General idea

The preparation state preparation_for(action(t_1+c, a)) is followed by the action state, created at the time point t_1+c. Rules that describe a transition from preparation to action states are given in *the executable theory from the preparation to the action state(s)* $Th_{p\to a}$.

Example

For the considered example the following rule holds:

∀t' state(γ, t', internal(mouse)) |= preparation_for(action(t'+c, goto_food)) ⇒
state(γ, t'+c, output(mouse)) |= performing_action(goto_food).

Step 4. ***Constructing an executable specification***

An executable specification π(γ, t) for agent A is defined by a union of the dynamic properties from the executable theories $Th_{o\to m}$, $Th_{m\to p}$ and $Th_{p\to a}$, identified during the steps 1-3. For the purposes of simulations of agent behavior the non-executable external behavioral specification is replaced by the executable behavioral specification.

Step 5. ***Constructing an executable specification for the whole external behavioral specification of an agent***

Other non-executable dynamic properties from the agent behavioral specification are substituted by executable ones by applying the same sequence of steps 1-4. In the end the executable properties for generating observation states from the states of the external world are added:

∀t' state(γ, t', world) |= [food ∧ not(cup)] ⇒
state(γ, t', input(mouse)) |= observed(food)

$\forall$t' state(γ, t', world) |= [not(food) $\wedge$ not(cup)] $\Rightarrow$
state(γ, t', input(mouse)) |= observed(not(food))

$\forall$t' state(γ, t', world) |= not(screen) $\Rightarrow$
state(γ, t', input(mouse)) |= observed(not(screen))

$\forall$t' state(γ, t', world) |= screen $\Rightarrow$
state(γ, t', input(mouse)) |= observed(screen)

It is assumed that an observation state is generated at the same time point, when a corresponding state of the external world is active.

Step 6. ***Translation of an executable specification into a description of a transition system***

General idea

For the purposes of practical analysis (e.g., by performing simulations and verification) a specification based on executable temporal logical properties generated by the procedure described in the previous section is translated into a finite state transition system model. The translation is based on the fact that a computation (in our case the execution of temporal logical properties) is essentially an (infinite) sequence of states (Vardi, 1996). Therefore, similarly to Vardi (1996), given an executable temporal specification one can construct a finite state transition system that generates the set of traces (by all possible executions of transition rules) equivalent to the set produced by all possible execution of temporal logical properties from the specification.

In computer science a finite state transition system is often described by a tuple $\langle Q, Q_0, \Sigma, \rightarrow \rangle$, where Q is a finite set of states of an agent, $Q_0 \subseteq Q$ is a set of initial states, Σ is a set of labels or events, which trigger the transition and $\rightarrow \subseteq Q \times \Sigma \times Q$ is a set of transitions. Such a representation often assumes an explicit denotation for every state in a transition system, which can be very numerous. However, a more compact representation, close to the production systems style, in the form of a set of transition rules with variables is possible (Arnold, 1994).

Definition (General Representation of a Finite State Transition System)

Let Ont be a state ontlogy consisting of sorts, constants, functions and predicates. Let At(Ont) be the set of (many-sorted predicate logic) atoms over Ont (possibly with variables). A general representation for a finite state transition system over Ont consists of transition rules of the form [P $\twoheadrightarrow$ N], where P is a proposition based on atoms from At(Ont), and N is a conjunction of atoms from At(Ont). The meaning is that when a certain instance of P by a certain variable assignment is true in a state, then the instance of N by the same variable assignment will be true in the next state; here $\twoheadrightarrow$ is a symbol for the transition between the two states.

Such a general representation for a finite state transition system has as an advantage that it does not depend on any particular implementation (e.g., verification or simulation tools). However, as this generic format describes states and transitions between them, it can be relatively easy translated into specialized languages of existing tools, based on the finite state transition system representation (e.g., the input format of the SMV model checker).

To translate the executable specification constructed at Step 5 from the theories $Th_{o \rightarrow m}$, $Th_{m \rightarrow p}$ and $Th_{p \rightarrow o}$ into the finite state transition system format, for each rule

from the executable specification the corresponding transition rule should be created. Let us first consider the formulae from the theory $Th_{o\to m}$. To relate states of a transition system to the timeline used in these rules the unary predicate present_time is used. The atom present_time(t) being true in a given state indicates that t is the time in this state. Furthermore, the assumption from $Th_{o\to m}$ that an observation state and a corresponding memory state are created at the same time point should be preserved. Thus, the time increment rules are defined as:

present_time(0) $\wedge \neg p \twoheadrightarrow$ present_time(1)
present_time(t) $\wedge \neg q_{mem} \wedge \neg p \twoheadrightarrow$ present_time(t+1)

Now, when a relation between states and time points is established, the rules defined in the $Th_{o\to m}$ can be easily translated into the transition system format as it is shown in Table 1.

Table 1. Translation of the formulae from the executable theory $Th_{o\to m}$ into the corresponding finite state transition rules

Rule from the executable theory $Th_{o\to m}$	**Corresponding transition rules**
Memory state creation rule $\forall t'$ state(γ, t') |= p $\Rightarrow$ state(γ, t') |= memory(t', p)	present_time(t) $\wedge$ p $\twoheadrightarrow$ memory(t, p)
Memory persistence rule $\forall t''$ state(γ, t'') |= memory(t', p) $\Rightarrow$ state(γ, t''+1) |= memory(t', p)	memory(t, p) $\twoheadrightarrow$ memory(t, p)

Next, let us translate the properties from $Th_{m\to p}$. The time increment rules are created similarly to the $Th_{o\to m}$ case based on the assumption from $Th_{m\to p}$ that a preparation state is generated at the same time point, when the condition for an output is true.

present_time(t) $\wedge$ $q_{cprep} \wedge \neg q_{cond}$(t) $\wedge \neg p \twoheadrightarrow$ present_time(t+1)
present_time(t) $\wedge$ $q_{prep} \twoheadrightarrow$ present_time(t+1)

Then, the rules defined in the $Th_{m\to p}$ are translated into the transition system format in a straightforward manner as it is shown in Table 2.

Table 2. Translation of the rules from the executable theory $Th_{m\to p}$ into the corresponding finite state transition rules

Rule from the executable theory $Th_{m\to p}$	**Corresponding transition rules**
Memory state creation rule $\forall t'$ state(γ, t') |= p $\Rightarrow$ state(γ, t') |= [memory(t', p) $\wedge$ stimulus_reaction(p)]	present_time(t) $\wedge$ p $\twoheadrightarrow$ [memory(t, p) $\wedge$ stimulus_reaction(p)]
Memory persistence rule $\forall t''$ state(γ, t'') |= memory(t', p) $\Rightarrow$ state(γ, t''+1) |= memory(t', p)	memory(t, p) $\twoheadrightarrow$ memory(t, p)
Conditional preparation generation rule $\forall t'$ state(γ, t') |= q_{mem} $\Rightarrow$ state(γ, t') |= q_{cprep}	$q_{mem} \twoheadrightarrow q_{cprep}$
Preparation state creation rule	present_time(t') $\wedge$ $q_{cprep} \wedge q_{cond}$(t) $\wedge$

$\forall$t', t state(γ, t') |= [$q_{cprep} \wedge q_{cond}$(t) $\wedge$ $\bigwedge$ stimulus_reaction(p)] p $\Rightarrow$ state(γ, t') |= q_{prep}	$\bigwedge$ stimulus_reaction(p) $\twoheadrightarrow$ q_{prep} p
Preparation state persistence rule $\forall$t' state(γ, t') |= [preparation_for(output(t'+c, a)) $\wedge \neg$ output(a)] $\Rightarrow$ state(γ, t'+1) |= preparation_for(output(t'+c, a))	preparation_for(output(t+c, a)) $\wedge \neg$output(a) $\twoheadrightarrow$ preparation_for(output(t+c, a))
Stimulus reaction state persistence rule $\forall$t' state(γ, t') |= [stimulus_reaction(p) $\wedge \neg$preparation_for(output(t'+c, a))] $\Rightarrow$ state(γ, t'+1) |= stimulus_reaction(p)	present_time(t') $\wedge$ stimulus_reaction(p) $\wedge \neg$preparation_for(output(t'+c, a)) $\twoheadrightarrow$ stimulus_reaction(p)

The executable theory from preparation to output $Th_{p\rightarrow o}$ contains only one formula that relates a preparation state at the time point t' to an output state at the time point t'+c; its translation is given in Table 3.

Table 3. Translation of the rule from the executable theory $Th_{p\rightarrow o}$ into the corresponding finite state transition rule

Rule from the executable theory $Th_{p\rightarrow o}$	**Corresponding transition rules**
Output generation rule $\forall$t' state(γ, t') |= preparation_for(output(t'+c, a)) $\Rightarrow$ state(γ, t'+c) |= output(a)	preparation_for(output(t+c, a)) $\wedge$ present_time(t+c-1) $\twoheadrightarrow$ output(a)

Example

The executable properties from the executable specification, translated into the transition rules for the considered example are given below:

food $\wedge$ not(cup) $\twoheadrightarrow$ observed(food)

not(food) $\wedge$ not(cup) $\rightarrow$ observed(not(food))

screen $\twoheadrightarrow$ observed(screen)

not(screen) $\twoheadrightarrow$ observed(not(screen))

present_time(t) $\wedge$ observed(food) $\twoheadrightarrow$ memory(t, observed(food))

present_time(t) $\wedge$ not(observed(not(food))) $\twoheadrightarrow$ memory(t, not(observed(not(food))))

present_time(t) $\wedge$ observed(not(food)) $\twoheadrightarrow$ memory(t, observed(not(food)))

present_time(t) $\wedge$ observed(not(screen)) $\twoheadrightarrow$ memory(t, observed(not(screen))) $\wedge$ stimulus_reaction(observed(not(screen)))

memory(t, observed(food)) $\twoheadrightarrow$ memory(t, observed(food))

memory(t, not(observed(not(food)))) $\twoheadrightarrow$ memory(t, not(observed(not(food))))

memory(t, observed(not(food))) $\twoheadrightarrow$ memory(t, observed(not(food)))

memory(t, observed(not(screen))) $\twoheadrightarrow$ memory(t, observed(not(screen)))

present_time(t) $\wedge$ $\exists$t2 [memory(t2, observed(food)) $\wedge$ $\forall$t3, t $\geq$ t3 > t2 memory(t3, not(observed(not(food))))] $\twoheadrightarrow$ conditional_preparation_for(action(goto_food))

present_time(t) $\wedge$ conditional_preparation_for(action(goto_food)) $\wedge$ memory(t, observed(not(screen))) $\wedge$ stimulus_reaction(observed(not(screen))) $\twoheadrightarrow$ preparation_for(action(t+c, goto_food))

present_time(t) ∧ stimulus_reaction(observed(not(screen))) ∧ not(preparation_for(action(t+c, goto_food))) ⇻ stimulus_reaction(observed(not(screen)))

preparation_for(action(t+c, goto_food)) ∧ not(performing_action(goto_food)) ⇻ preparation_for(action(t+c, goto_food))

preparation_for(action(t+c, goto_food)) ∧ present_time(t+c-1) ⇻ performing_action(goto_food).

The described transformation procedure was implemented in Java™, with an input (an external behavioral specification) and an output (an executable specification and finite transition system descriptions) files specified in textual format.

The generated finite state transition system representation is used in this paper for performing two types of analysis: by running simulations of different types of agent behavior, and by analyzing the consequences of the agent behavior by model checking techniques.

3 Analysis of Agent Behavior by Simulation

In this Section the proposed transformation procedure is applied for simulating the delayed response and the adaptive behavior of an agent. For performing simulations a special software tool has been developed. Based on a specification of agent behavior in form of a transition system and using a sequence of external events (i.e., stimuli) as input, the program generates a trace (i.e., a sequence of agent states over time). The generated in such way traces can be used for analysis of external and internal dynamics of the agent in different experimental settings.

3.1 Simulation of the Delayed-response Behavior of the Agent

First, let us consider in more detail the example of the delayed-response behavior of the agent (a laboratory mouse), briefly introduced in Section 2. The behavioral specification for this case is given below; it consists of environmental properties and externally observable behavioral properties of the agent.

Environmental properties:

EP1: At some time point food has been put at the position p1, after some time a cup has been placed upon food and after that the screen is raised

∃t1, t2, t3 t2>t1 & t2<t3 state(γ, t2) |= cup_at(p1) & state(γ, t1) |= food_at(p1) & state(γ, t3) |= not_screen

EP2: Food stays at the position where it has been put until it has been taken away or the agent is satisfied

∀t4 state(γ, t4) |= [food_at(X) & not(mouse_sat) & not(food_taken_away_from(X))] ⇒ state(γ, t4+1) |= food_at(X), where X ∈ {p1, p2}

EP3: After the screen has been raised, it will never be drawn down again

∀t5 state(γ, t5) |= not_screen ⇒ state(γ, t5+1) |= not_screen

EP4: After placing the cup it will not be removed

$\forall$t6 state(γ, t6) |= cup_at(X) $\Rightarrow$ state(γ, t6+1) |= cup_at(X), where X $\in$ {p1, p2}

Properties that define the externally observable behavior of the mouse:

BP1: The mouse is able to observe presence (absence) of screen.

$\forall$t7 state(γ, t7) |= X $\Rightarrow$ $\exists$t8 t8>t7 state(γ, t8, input(mouse)) |= observed(X),
where X $\in$ {not_screen, screen}

BP2: The mouse is always able to observe presence or absence of food if the cup is not covering it.

$\forall$t9 state(γ, t9) |= X & not(cup_at(Y))$\Rightarrow$ $\exists$t10 t10>t9 state(γ, t10, input(mouse)) |= observed(X),
where X$\in$ {food_at(Y), not(food_at(Y))} and Y $\in$ {p1, p2}

BP3: The mouse is able to observe that food is taken away if the cup is not covering it.

$\forall$t11 state(γ, t11) |= food_taken_away_from(X) & not(cup_at(X)) $\Rightarrow$
$\exists$t12 t12>t11 state(γ, t12, input(mouse)) |= observed(food_taken_away_from(X)),
where X $\in$ {p1, p2}

BP4: The mouse always arrives at the position where it goes.

$\forall$t13 state(γ, t13, output(mouse)) |= performing_action(goto(X)) $\Rightarrow$
$\exists$t14 t14>t13 state(γ, t14) |= mouse_at(X),
where X$\in$ {p1, p2}

BP5: If the mouse is at the position with food, then it will be eventually satisfied (after consuming food).

$\forall$t15 state(γ, t15) |= mouse_at(X) & food_at(X) $\Rightarrow$ $\exists$t16 t16>t15 state(γ, t16) |= mouse_sat,
where X $\in$ {p1, p2}

BP6: The mouse consumes food completely.

$\forall$t17 state(γ, t17) |= mouse_sat & mouse_at(X) $\Rightarrow$ state(γ, t17+1) |= not(food_at(X))

BP7: If mouse found the position with food, it stays there.

$\forall$t18 state(γ, t18) |= mouse_at(X) & food_at(X) $\Rightarrow$ $\forall$t19 t19>t18 mouse_at(X)

BP8: Delayed-response behavior of the mouse

The mouse goes to the position with food if and only if it observes that there is no screen and at some point in the past the mouse observed food and since then did not observe the absence of food.

$\forall$t20 [state(γ, t20, input(mouse)) |= observed(not_screen) &
$\exists$t21< t20 state(γ, t21, input(mouse)) |= observed(food_at(X)) &
$\forall$t22, t20 $\geq$ t22 > t21 state(γ, t22, input(mouse))|= not(observed(not(food_at(X))))] $\Rightarrow$
$\exists$t23, t23>t20 state(γ, t23, output(mouse)) |= performing_action(goto(X)),
where X$\in$ {p1, p2}

The complete specification of the finite state transition system generated for this specification is given in (Sharpanskykh & Treur, 2005). This transition system was used to simulate a scenario of the animal's behavior with the following events in the environment: food is put at position p1 (time point 0), the screen separating the animal from food is present (time points 0-3), a cup is put at position p1, covering the food (time point 2), and the screen is removed (time point 4). The results of the simulation in the form of a trace (i.e., a sequence of states) are given in Table 4.

Table 4. Simulation trace illustrating delayed-response of the agent

0: present_time(0) world(0,food_at(p1)) world(0,screen)	9: present_time(4) world(4,not(screen))
1: observed(0,food_at(p1)) observed(0,screen)	10: observed(4,not(screen))
2: memory(observed(0,food_at(p1))) memory(observed(0,screen))	11: memory(observed(4,not(screen))) stimulus_reaction(observed(not(screen)))
3: conditional_preparation_for(action(goto(p1)))	12: preparation_for(action(5,goto(p1)))
4: present_time(1)	13: not(stimulus_reaction(observed(not(screen))))
5: present_time(2) world(2,cup_at(p1))	14: present_time(5) performing_action(goto(p1))
6: observed(2,cup_at(p1))	15: not(preparation_for(action(5,goto(p1))))
7: memory(observed(2,cup_at(p1))	16: present_time(6) world(6,mouse_at(p1))
8: present_time(3)	17: present_time(7) world(7,mouse_sat)

Furthermore, a transition system representation can be used for construction of graphical models of agent dynamics. A graphical model for the considered example is shown in Figure 1. The state description literals with names started with a capital letter denote variables, which allow a concise representation of sets of states. For example, the label mem(T, obs(not(screen))) represents the set that includes every state corresponding to some time point t, in which the state property mem(t, obs(not(screen))) holds. An AND-relation between states requires all state properties of the states in the relation to be true in order to carry out the corresponding transition. A persistent state once activated, remains active at every time point in the future, i.e. the state properties of a persistent state hold for every time point in the future. The model in Figure 1 has been built manually. However, there exist tools, as one described in van Ham, van de Wetering, and van Wijk (2002), which allow for automatic visualization of finite state transition systems and can be used for graphical analysis of executable models. The graphical representation is particularly useful for the analysis of large transition systems. Usually such systems comprise a large number of transition rules specified without any particular order that do not provide a clear and ordered overview on the dynamics of a system. A graphical counterpart of a transition system makes temporal and causal relations between states of a system explicit and allows tracking different development paths of the system. Furthermore, the existing tools allow zooming into particular parts of a transition system to investigate relations between particular states.

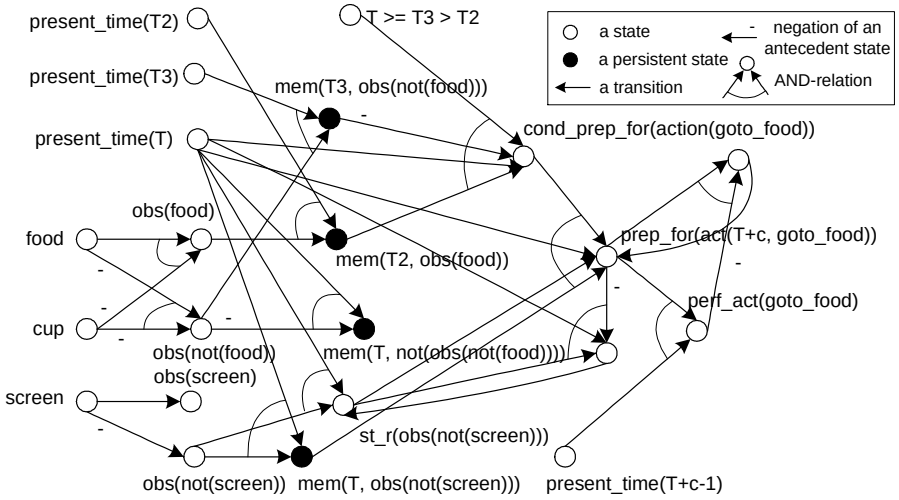


Fig. 1. Graphical model, which describes delayed-response behavior in executable form

3.2 Simulation of Adaptive Agent Behavior

In the second simulation example the adaptive behavior of *Aplysia Californica* (a sea hare) is considered. In neurobiology *Aplysia* has been often used for investigating classical and operant conditioning (Carew & Walters & Kandel, 1981). Consider a slightly simplified classical conditioning experiment of the *Aplysia*'s defensive withdrawal reflex. Before a learning phase a strong noxious stimulus (an electric shock) on the *Aplysia*'s tail produces a defensive reflex (a contraction), while a light tactile stimulus on *Aplysia*'s siphon does not lead to contraction. Formally:

$\forall t9 \le t$ state(γ, t9, input(aplysia)) |= observed(tail_shock) $\Rightarrow$
state(γ, t9+c, output(aplysia)) |= performing_action(contraction)

During the learning phase a light tactile stimulus on the Aplysia's siphon is repeatedly paired with an electric shock on its tail. After a few trials (for this example three temporal pairings are assumed) the animal reacts by contraction to the light tactile stimulus. The property that describes the learning process of the animal from the external perspective can be represented in the form $[\varphi_p(\gamma, t) \Rightarrow \varphi_f(\gamma, t)]$, where $\varphi_p(\gamma, t)$ is the formula:

$\exists t2, t3, t4, t5, t6, t7$ [t2 < t3 & t3 < t4 & t4 < t5 & t5 < t6 & t6 < t7 & t7 < t &
state(γ, t2, input(aplysia)) |= observed(touch_siphon) &
state(γ, t3, input(aplysia)) |= observed(tail_shock) &
state(γ, t4, input(aplysia)) |= observed(touch_siphon) &
state(γ, t5, input(aplysia)) |= observed(tail_shock) &
state(γ, t6, input(aplysia)) |= observed(touch_siphon) &
state(γ, t7, input(aplysia)) |= observed(tail_shock)]

and $\varphi_f(\gamma, t)$ is the formula

$\forall t8 \ge t$ [state(γ, t8, input(aplysia)) |= observed(touch_siphon) $\Rightarrow$
state(γ, t8+c, output(aplysia)) |= performing_action(contraction)]

with $\varphi_{cond}(\gamma, t, t8)$ is

$\forall t8 \geq t$ state(γ, t8, input(aplysia)) |= observed(touch_siphon)

and $\varphi_{act}(\gamma, t8)$ is

state(γ, t8+c, output(aplysia)) |= performing_action(contracts).

For this experiment c is assumed to be equal to two time units in a relative time scale.

Using the automated procedure, from the external behavioral specification of *Aplysia* a transition system was generated. This transition system was used to simulate a scenario of the animal's behavior with the following stimuli: touch the siphon (time points 0, 5, 9 and 15) and shock on the tail (time points 1, 6 and 10). The results of the simulation in form of a partial trace are given in Table 5.

Table 5. Partial simulation trace illustrating adaptive behavior of *Aplysia Californica*

0: present_time(0) world(0,touch_siphon)	11: not(preparation_for(action(3,contracts)))
1: not(world(0,touch_siphon)) observed(0,touch_siphon)	……
2: memory(observed(0,touch_siphon)) not(observed(0,touch_siphon)) stimulus_reaction(observed(touch_siphon))	39: present_time(15) world(15,touch_siphon)
3: present_time(1) world(1,tail_shock)	40: not(world(15,touch_siphon)) observed(15,touch_siphon)
4: not(world(1,tail_shock)) observed(1,tail_shock)	41: memory(observed(15,touch_siphon)) not(observed(15,touch_siphon)) stimulus_reaction(observed(touch_siphon))
5: memory(observed(1,tail_shock)) not(observed(1,tail_shock)) stimulus_reaction(observed(tail_shock))	42: preparation_for(action(17,contracts))
6: conditional_preparation_for(action(contracts))	43: not(stimulus_reaction(observed(touch_siphon))) not(stimulus_reaction(observed(tail_shock)))
7: preparation_for(action(3,contracts))	44: present_time(16)
8: not(stimulus_reaction(observed(touch_siphon))) not(stimulus_reaction(observed(tail_shock)))	45: present_time(17) performing_action(contracts)
9: present_time(2)	46: not(preparation_for(action(17,contracts)))
10: present_time(3) performing_action(contracts)	47: present_time(18)

In the given trace the process of conditioning starts at the state 0 (time point 0) and finishes at the state 36 (time point 12). After that the animal reacts to a light tactile stimulus (state 39) by producing a defensive reflex (states 42-45).

A graphical model for the example of classical conditioning for Aplysia Californica's defensive withdrawal reflex is shown in Figure 2.

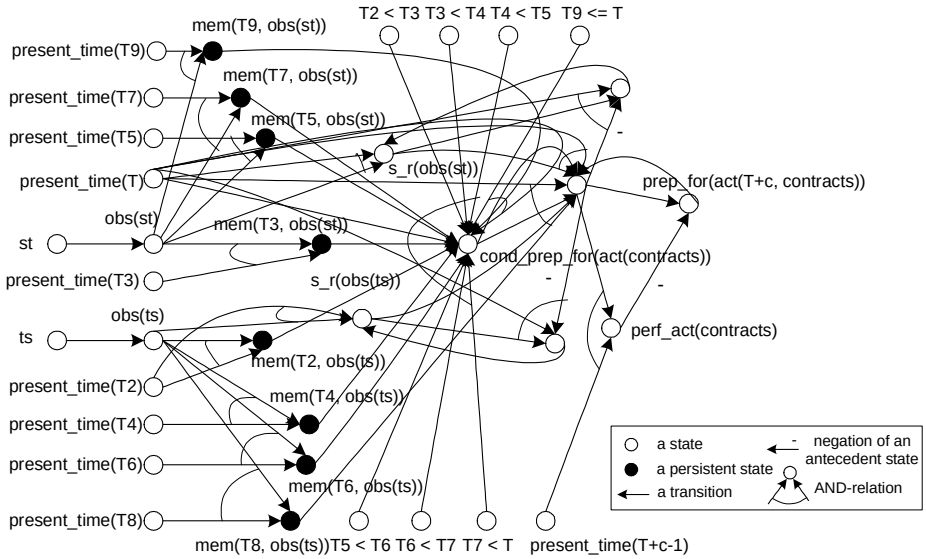


Fig. 2. A graphical model for the example of classical conditioning for Aplysia Californica's defensive withdrawal reflex

4. Analysis of the Consequences of Agent Behavior by Model Checking

The proposed approach for analysis of the consequences of the agent behavior is based on the statement that the logical consequences of a certain external behavior specification are the logical consequences of the corresponding internal executable specification. This statement is supported by the following theorem.

Theorem[1]
If the internal dynamics specification $\pi(\gamma, t)$ corresponds (by the transformation above) to the external behavioral specification $\varphi(\gamma, t)$, and $\psi(\gamma, t)$ is a dynamic property of the agent in its environment, then $\psi(\gamma, t)$ is entailed by $\varphi(\gamma, t)$ if and only if $\psi(\gamma, t)$ is entailed by $\pi(\gamma, t)$:

$$\forall\gamma\ [\pi(\gamma, t) \Rightarrow \psi(\gamma, t)] \quad \Leftrightarrow \quad \forall\gamma\ [\varphi(\gamma, t) \Rightarrow \psi(\gamma, t)]$$

The consequences of the generated executable specification are easier to determine because of the simpler format of the internal dynamics specification. Furthermore, the process of analysis of such consequences can be automated by model checking techniques. For this purpose the SMV model checking tool is used in this paper. The SMV uses efficient algorithms to analyze a model of an agent system and the Computational Tree Logic (CTL) (McMillan, 1993) is used for properties (e.g.,

[1] The proof for this theorem is given in Sharpanskykh & Treur (2005)

properties concerning well-being) to check. CTL is branching-time logic, meaning that its model of time is a tree-like structure in which different paths in the future are possible, any one of which might be actually realized. A particular use of CTL will be demonstrated by an example in this Section.

Moreover, the language for model specification in the SMV is similar to the executable format of agent behavioral specifications, which facilitates the automatic translation of the description of a finite state transition system, generated by the procedure introduced in Section 2, into the SMV input format. A specification in SMV is a plain text file that consists of two main parts: (1) a specification of a transition system and (2) a set of properties to be checked on the transition system specification expressed in CTL.

A transition system (or model) specification in SMV consists of a number of sections. In the section labeled VAR the names and types of the variables used in the model are defined. The type associated with a variable is either Boolean, scalar, or an array. In the second section labeled ASSIGN the initial values of variables are defined (i.e., the values that the variables have in the initial state) and the transition rules between states are specified. The transition rules are specified by case-expressions that define the change of values of the variables of the transition system as follows:

```
next (var) := case
                        boolean_expression: val;
              esac
```

All case-expressions are evaluated in every state. When boolean_expression on the left-hand side of ":" of some transition rule is evaluated to true in some state, then the corresponding variable var will receive the value val in the next state.

For the translation of an executable specification of agent behavior into a SMV specification a dedicated procedure has been developed and implemented. This procedure is applied for every dynamic property in an executable behavioral specification as follows: First, the normalized memory state formula $q_{mem}(t)$ and the normalized condition state formula $q_{cond}(t, t_1)$ are processed by applying the steps 1-3 described below. After that conditional preparation generation rules are added by performing the step 4. Finally, the preparation and output state creation rules are generated by performing the step 5.

Step 1. For each occurrence of an existential quantifier of the form ∃t1 P(t1), where t1 is a time variable name and P(t1) is some function of the form memory(observed(t1, obs_event)), ¬memory(observed(t1, obs_event)), memory(output(t1, act_event)), or ¬memory(output(t1, act_event)), where obs_event and act_event are some atoms and for each occurrence of a universal quantifier of the form ∀t1 P(t1), create an atom (a label) t1 and add to the SMV specification the corresponding initialization rules.

Step 2. For each occurrence of the expression Q t1, t2 R t1 memory(observed(t1, obs_event)), where Q is either an existential or a universal quantifier, R is the comparison relation for the linear ordered time line: R∈{<, ≤}; t1 and t2 are time variables, add to the specification the following rule:

```
next(t1):= case
            t2 & obs_event: 1; //memory state creation
            !t2: 0;
            1: t1;  //persistence of memory
```

```
esac;
```

Similar rules should be added for the expressions Q t1, t2 R t1 memory(output(t1, act_event), Q t1, t2 R t1 ¬memory(observed(t1, obs_event)) and Q t1, t2 R t1 ¬memory(output(t1, act_event)).

Step 3. For each expression of the form ∃t1, t2 ∀t3 [t3 R t2 AND t1 R t3 AND memory(observed(t1, obs_event1)) AND memory(observed(t2, obs_event2)) & P3(t3)] if P3(t) is of the form memory(observed(t3, obs_event))

i. For t3 < t2 and t1< t3 add to the specification the following rules:

```
t3t1_eq: boolean ;
init(t3t1_eq):=0;
next(t3t1_eq):= case
            t1: 1;
             1: 0;
esac;
next(t1):= case
          !obs_event2 & !t2 &
t3t1_eq & !obs_event3: 0;
            1: t1;
esac;
```

```
next(t3):= case
            !t1: 0;
            !obs_event2 & !t2 &
!obs_event3: 0;
            !obs_event2 & !t2 &
obs_event3: 1;
            1: t3;
esac;
```

The cases (ii) t3 < t2 and t1≤ t3; (iii) t3 ≤ t2 and t1< t3 and (iiii) t3 ≤ t2 and t1≤ t3 are dealt similarly.

Step 4. Add conditional preparation generation rules to the specification:

```
next(fmemN):= case //N is a number of a dynamic property in the input specification
            ∧ti: 1; // conjunction of all labels, created based on φp(γ, t)
            i
            1: 0;
esac;
```

Step 5. For each action and communication a function output(act_event) in a formula $q_{bt}(t)$ add to the specification the following rules:

```
next(fprep_act):= case
            fmemN & ∧tj: 1;
                    j
                     1: 0;
esac;
```

```
next(act_event):= case
                  fprep_act: 1;
                          1: 0;
esac;
```

When an executable specification is translated into the SMV input format, the checking of a CTL property(ies) on this specification can be automatically performed using the SMV. As a result the tool generates an answer, if the specified property(ies) are satisfied by the model. If the property is not satisfied, a counterexample is provided. A counter-example shows a sequence of states that resulted in a state, in which the checked property is not satisfied. In such a way, the reason for the checking failure can be determined.

In this section the proposed analysis method is described and illustrated by an example, in which next to the delayed-response behavior (considered in Section 3.1) also the motivation-based behavior of the agent is analyzed. The specification for the delayed-response behavior (denoted here by φ_1) is given in Section 3.1. The specification for the motivation-based behavior (denoted here by φ_2) is constructed from the properties BP1-BP7, which are defined in Section 3.1, and additional properties BP9-BP12 given below.

BP9: Motivation-based behavior of the mouse (start at position p1)
If the mouse observes no screen and it is not satisfied, and at some time point in the past it observed food at position p1 and since then did not observe food at position p2, then the mouse will go to position p1.

∀t24 [state(γ, t24, input(mouse)) |= observed(not_screen) & state(γ, t24) |= not(mouse_sat) &

∃t25, t25<t24 state(γ, t25, input(mouse)) |= observed(food_at(p1)) &

∀t26, t26 ≤ t24 & t26 > t25 state(γ, t26, input(mouse)) |= not(observed(food_at(p2)))] ⇒

∃t27, t27>t24 state(γ, t27, output(mouse)) |= performing_action(goto(p1))

BP10: Motivation-based behavior of the mouse (start at position p2)
If the mouse observes no screen and it is not satisfied, and at some time point in the past it observed food at position p2 and since then did not observe food at position p1, then the mouse will go to position p2.

∀t24 [state(γ, t24, input(mouse)) |= observed(not_screen) &

state(γ, t24) |= not(mouse_sat) & ∃t25, t25<t24 state(γ, t25, input(mouse)) |= observed(food_at(p2)) &

∀t26, t26 ≤ t24 & t26 > t25 state(γ, t26, input(mouse)) |= not(observed(food_at(p1)))] ⇒

∃t27, t27>t24 state(γ, t27, output(mouse)) |= performing_action(goto(p2))

BP11: Motivation-based behavior of the mouse (continue at position p2)
If the mouse is at position p1 and there is no food at p1 and the mouse is still not satisfied, then it will go to position p2 to continue its search for food

∀t28 state(γ, t28) |= mouse_at(p1) & not(food_at(p1)) & not(mouse_sat) ⇒

∃t29, t29>t28 state(γ, t29, output(mouse)) |= performing_action(goto(p2))

BP12: Motivation-based behavior of the mouse (continue at position p1)
If the mouse is at position p2 and there is no food at p2 and the mouse is still not satisfied, then it will go to position p1 to continue its search for food

∀t30 state(γ, t30) |= mouse_at(p2) & not(food_at(p2)) & not(mouse_sat) ⇒

∃t31, t31>t30 state(γ, t31, output(mouse)) |= performing_action(goto(p1))

Such a specification of behavior can be attributed, for example, to an animal that feels hunger.

Both types of behavior of the agent are analyzed in two different environmental experimental settings (E, resp. E') with an identical initial situation, described as follows:

The mouse is placed in front of a transparent screen that separates it from a piece of food that is put behind the screen. The mouse is able to observe the position of food and of the screen. At some moment after food has been put, a cup is placed covering the food, which makes food invisible for the mouse. After some time the screen is raised and the animal is free to go to any position. If the mouse comes to the position, where the food is hidden, then it will be capable to lift up the cup and get the food.

By means of the analysis method described below it is determined for each of the environmental settings and each type of behavior whether the combination will bring the agent well-being.

The Analysis Method

(a) By means of the translation procedure described in Section 2, each external behavioral specification φ_i is automatically translated into its corresponding executable internal dynamics specification π_i and its related state transition system representation τ_i.
(b) The well-being properties ψ and ψ' to be checked are specified in CTL
(c) Using the state transition system representations τ_i, verification of each of the agent models with respect to properties ψ and ψ' is performed in the SMV model checker, resulting in confirmed or rejected entailment relations between the π_i and ψ and ψ'.
(d) Based on the theorem introduced at the beginning of this section the confirmed or rejected entailment relations between the π_i and ψ and ψ' imply corresponding confirmed or rejected entailment relations between the φ_i and ψ and ψ'.

The environmental conditions E are defined by dynamic properties EP1-EP4 listed in Section 3.1. For this example, ψ is the following conditional well-being property (which is expressed conditionally for environmental conditions E):

> for all traces, if the screen is removed and food is hidden under the cup, then the mouse will eventually be satisfied.

This property ψ can be expressed in Computation Tree Logic (CTL) (Clarke & Grumberg & Peled, 1999) required for verification in the SMV model checking tool as follows:

$$\boldsymbol{AG}\ (\text{not\_screen} \ \&\ \text{food} \ \&\ \text{cup} \rightarrow \boldsymbol{AF}\ \text{mouse\_sat})$$

where ***A*** is a path quantifier defined in CTL, meaning “for all computational paths”, ***G*** and ***F*** are temporal quantifiers that correspond to “globally” and “eventually” respectively.

The automatic verification in the SMV model checking tool showed that the property ψ expressing well-being under environmental conditions E is entailed by the model of the agent delayed-response behavior expressed by φ_1. The model of agent motivation-based behavior φ_2 also turns out to entail the general property ψ.

In the second experimental setting, described by environmental conditions E', the mouse observed food for some time at the position p1, after that one cup is put covering the food and another cup is put at the position p2, which is also behind the transparent screen. Thereafter, invisibly for the mouse, food is removed from position p1 and put under the cup at position p2. Later the screen is raised and the animal is

free to go to any position. The environmental conditions E' are formalized by dynamic properties BP2-BP4, and by the property BP5:

EP5: At some time point food had been put at the position p1, after some time one cup had been placed upon food and another cup had been placed at the position p2; thereafter food has been taken away from p1 and has been put at p2 behind the cup, after that the screen is raised

∃t32, t33, t34, t35, t33>t32 & t33<t34 & t35>t34 state(γ, t33) |= [cup_at(p1) & cup_at(p2)] &
state(γ, t32) |= food_at(p1) & state(γ, t34) |= [food_taken_away_from(p1) & food_at(p2)] &
state(γ, t35) |= not_screen

The global property ψ' to be verified in this case expresses well-being under these environmental conditions E':

for all traces if the screen is removed and food is hidden behind the cup at position p2, then the mouse will eventually be satisfied,

or, in CTL:

AG (not_screen & food_at(p2) & cup_at(p2) → ***AF*** mouse_sat)

The automated verification in SMV showed that the model of the agent behavior φ_1 for the delayed-response case does not entail property ψ' expressing well-being under environmental conditions E'. From the counter-example generated by the model checker it is visible that the animal went to the position p1, and did not find food there, and after that did not go anywhere else, which caused the failure of the property.

Unlike the external behavior specification φ_1 that describes the delayed-response behavior of the agent, the specification φ_2 for the motivation-based behavior includes behavioral repertoire to deal with invisible food, expressed in the form of properties that turn out to ensure the entailment of global property ψ'. More specifically, φ_2 expresses the behavior that if the agent could not find food at the position where it has seen it before, and the agent is still not satisfied, then the agent will search for food at another position p2. Formally this is expressed by:

∀t5 state(γ, t5) |= mouse_at(p1) & not(food_at(p1)) & not(mouse_sat) ⇒
  ∃t6, t6>t5 state(γ, t6, output(mouse)) |= performing_action(goto(p2))

∀t7 state(γ, t7) |= mouse_at(p2) & not(food_at(p2)) & not(mouse_sat) ⇒
  ∃t8, t8>t7 state(γ, t8, output(mouse)) |= performing_action(goto(p1))

The automated verification in SMV confirmed that the external behavioral specification φ_2 for the case of motivation-based behavior entails property ψ'.

Table 6. Outcomes of the Example Analysis

		well-being under different environmental conditions	
		ψ	ψ'
behavior type	delayed response φ_1	+	-
	motivation-based φ_2	+	+

From the results of verification of the external behavioral specifications φ_1 and φ_2 for both types of behavior in both experimental settings with respect to the entailment

of properties ψ and ψ' (see Table 6) we draw the conclusion that the agent that manifests motivation-based behavior φ_2 fits more for surviving in the world, described by the two types of experimental conditions than the agent that has the delayed-response behavior φ_1.

5 Discussion

Behavior of organisms comes in a variety of forms and complexities. Simple forms of behavior such as stimulus-response patterns can be formalized in relatively simple terms, based on direct stimulus-action associations that can be considered as associations between an input state and a subsequent output state of the organism. A description of an organism's behavior in terms of such stimulus-action associations can directly be used as a basis to model and analyze this behavior. For more complex behavior, however, the picture is not so simple. To describe behavior from the external perspective, in general, an input-output correlation (cf. Kim, 1996) has to be specified which indicates how a pattern of input states over time relates to a pattern of output states over time. With increasing complexity of the behavior considered, specification of such an input-output correlation will become more complex, and not take the form of direct stimulus-action associations anymore. The question arises how such more complex descriptions of behavior can be expressed and handled, and, in particular, how such behavior can be analyzed, for example, by simulation and verification. The answer on this question developed in this paper is twofold. First, a formal language is put forward that allows specifying behavior from an external perspective in terms of dynamic properties involving input states and output states over time. Secondly, it is shown how an external behavior specification expressed in such a language can be automatically transformed into an equivalent executable specification that easily can be used to perform different types of analysis of agent behavior. This transformation creates a specification based on postulated internal states (in particular memory states and preparation states), and their direct temporal relationships.

Such a transformation in principle can be done in different manners, making use of different types of internal states. For the approach chosen here a main role is played by internal memory states of an agent that are based on the agent sensing (i.e., observations) of not only his/her environment, but also of his/her own behavior (e.g., actions). This relates to a thesis currently recognized in neurobiological research (Di Ferdinando & Parisi, 2004) that internal representations of an agent are based not only on the properties of the sensory input, but also on the properties of the actions with which the agent responds to this sensory input. The internal states can represent world states, but may also refer to more complex temporal patterns occurring in the past.

Sometimes, when a structure of a neurological circuit of an organism is known, it is possible to relate postulated internal states to certain real neurological states of an organism. The neurological model of *Aplysia Californica*, suggested by Roberts and Glanzman (2003) allows finding some correspondences between the postulated internal states described in the example of this paper and the real physical states of the organism. The observation states from our model can be related to activation states of

sensory neurons, whereas the memory states (to some extent) can be put into correspondence with an enhancement of the strength of the synaptic connection between the sensory and motor neurons and with an associative increase in the excitability of the siphon sensory neurons of *Aplysia* (as followed from a correspondence with professor Glanzman).

However, the rules for the creation of internal states of an agent proposed in this approach are based on the idealized assumptions describes above, which may lead to internal states that do not correspond in a direct manner to internal states actually occurring in certain biological organisms. if such a direct correspondence is aimed for, to ensure the biological plausibility of the models constructed using the proposed approach for specific forms of organisms (types of agents) the rules for creation of intrinsic states may be adjusted correspondingly.

Also other existing frameworks and approaches that include different types of mental states of an agent (e.g., BDI (Rao & Georgeff, 1991), KARO (van Linder & van der Hoek & Meyer, 1998), Schweiger Gallo & Gollwitzer (2007)) can be considered for internal representation. In particular, these frameworks recognize attitudes of agents such as desires, intentions, and goals. More specifically, in (Gollwitzer, 1999) it is shown that intentions can be implemented by if-then plans that describe when, where and how a set goal of an agent has to be put into action: "if situation x is encountered, then the agent will perform behavior y". However, such if-then plans can be specified using the approach proposed in this paper by temporal relations between externally observable and internal states of an agent. In this case no introduction of supplementary internal concepts is required, and the intentional aspects of the agent behavior are implicitly realized through the temporal rules in the behavioral specification of the agent. However, goal and intention concepts could also be considered explicitly by adding them to our ontology in order to get more transparency. However, this will add no essential expressivity, as they would be a renaming of already available complex expressions over our memory states; see also (Jonker, Treur, and Vries, 2002). In future work it will be investigated, which alternative or additional attitudes of agents could be included into the internal framework in order to more transparently represent complex behavior of an agent.

In general, the transformation into executable format can be achieved in different ways, depending on the format of an external behavioral specification of an agent system. For example, for translating agent behavioral specifications expressed in modal temporal logics into executable format, procedures described in (Fisher, 1996) can be used. This paper exploits a procedure to generate an executable internal behavioral specification from a more expressive external specification than is possible in modal temporal logics. The executable format introduced in this paper has similarities with the production rule representation formats used in existing cognitive architectures. For example, in the ACT-R architecture (Anderson, 1996) rules are stored in the procedural memory, which is essentially specified by a production system and can be easily expressed by formulae from the executable specification introduced in this paper.

The generated executable specification can be used to perform different types of analysis, in particular, simulations and the analysis of consequences of agent behavior. Several examples to illustrate both these types of analysis are demonstrated in this paper. Alternative methods for temporal analysis of reactive systems are

discussed in (Manna & Pnueli, 1995); also these methods can be applied, once an executable behavioral specification has been generated.

The analysis of consequences of agent behavior is performed by means of model checking techniques using the SMV model checker. Notice that an SMV-specification comprises constants, variables and state transition rules with limited expressiveness (e.g., no quantifiers). Furthermore, for expressing one complex temporal relation a large quantity (including auxiliary) of transition rules is needed. Specification of agent system behavior in the more expressive predicate-logic-based language TTL is much easier. TTL proposes an intuitive way of creating a specification of system dynamics, which still can be automatically translated into a state transition system description, as shown in this paper.

References

Allen, C., and Bekoff, M., (1997). Species of Mind: the philosophy and biology of cognitive ethology. MIT Press.

Anderson, J. R. (1996). ACT: A simple theory of complex cognition. American Psychologist, 51, 355-365.

Arnold, A. (1994). Finite transition systems. Semantics of communicating systems. Prentice-Hall.

Balkenius, C., & Moren, J. (1999). Dynamics of a classical conditioning model. Autonomous Robots, 7, 41-56.

Carew, T.J. & Walters, E.T., & Kandel, E.R. (1981). Classical conditioning in a simple withdrawal reflex in Aplysia Californica. The Journal of Neuroscience, 1(12), 1426-1437.

Clarke, E.M., & Grumberg, E.M., & Peled, D.A. (1999). Model Checking, MIT Press, Cambridge Massachusetts, London England.

Di Ferdinando, A. & Parisi, D. (2004) Internal representations of sensory input reflect the motor output with which organisms respond to the input. In Carsetti A. (ed.): Seeing, Thinking and Knowing. Kluwer, Dordrecht, 115-141

Dudai Y. (1990). The Neurobiology of Memory. Concepts, Findings, Trends. Oxford: Oxford University Press.

Fisher, M. (1996). An Introduction to Executable Temporal Logics, Knowledge Engineering Review 11(1), 3-36.

Fitting, M. (1996). First-order Logic and Automated Theorem Proving, 2nd edition, Springer-Verlag.

Hawkins, J. (2004). On Intelligence, Henry Gholt and Co Ltd.

Heil, J. (2000). Philosophy of Mind. Routledge.

Hunter, W.S. (1912). The delayed reaction in animals. Behavioral Monographs, 2, 1-85.

Jonker, C.M., & Treur J., & Wijngaards W.C.A. (2003). A temporal-modelling environment for internally grounded beliefs, desires, and intentions. Cognitive Systems Research Journal, 4(3), 191-210.

Jonker, C.M., & Treur, J. (2002) Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. International Journal of Cooperative Information Systems, 11, 51-92.

Jonker, C.M., Treur, J., and Vries, W. de, (2002). Temporal Analysis of the Dynamics of Beliefs, Desires, and Intentions. Cognitive Science Quarterly (Special Issue on Desires, Goals, Intentions, and Values: Computational Architectures), vol. 2, 2002, pp.471-494.

Kim, J. (1996). Philosophy of Mind. Westview Press

Kowalski, R., & Sergot, M. (1986). A logic-based calculus of events. New Generation Computing, 4, 67-95.
Manna, Z., & Pnueli A. (1995) Temporal verification of reactive systems, Springer-Verlag, Berlin Heidelberg New York.
Manzano, M. (1996). Extensions of First Order Logic, Cambridge University Press.
McMillan, K. (1993). Symbolic Model Checking, Kluwer Academic Publishers.
Priest, S. (1991). Theories of the Mind. Penguin.
Putman, H. (1975). Mind, Language, and Reality: Philosophical papers, vol.2. Cambridge: Cambridge University Press.
Rao, A. S. & Georgeff, M. P. (1991). Modeling agents within a BDI architecture. In Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR '91). Morgan Kaufmann, Cambridge, MA, 473-484.
Reiter, R. (2001). Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. Cambridge MA: MIT Press.
Roberts, A.C., & Glanzman, D.L. (2003). Learning in Aplysia: looking at synaptic plasticity from both sides. Trends in Neurosciences, 26, 662-670.
Schweiger Gallo, I., & Gollwitzer, P. M. (2007). Implementation intentions: A look back at fifteen years of progress. Psicothema, 19, 37-42.
Skinner, B.F. (1935). The generic nature of the concepts of stimulus and response. Journal of General Psychology, 12, 40-65.
Skinner, B.F. (1953). Science and human behavior. New York: Macmillan.
Sharpanskykh, A. & Treur, J. (2005). Modeling of Agent Behavior Using Behavioral Specifications (Tech. Rep. 06-02ASRAI; http://hdl.handle.net/1871/9123). Vrije Universiteit, Amsterdam.
van Ham, F., & van de Wetering, H., & van Wijk, J.J. (2002). Interactive Visualization of State Transition Systems. IEEE Transactions on Visualization and Computer Graphics, 8(4), IEEE CS Press, 319-329.
van Linder, B.W., & van der Hoek, & Meyer, J.-J. Ch. (1998). Formalising Abilities and Opportunities of Agents, Fundamenta Informaticae, 34(1-2), 53-101.
Vardi, M.Y. (1996). An automata-theoretic approach to linear temporal logic. In: Proceedings of the VIII Banff Higher Order Workshop, in: Lecture Notes in Computer Science, vol. 1043, Springer-Verlag, 238–266.
Watson, J. B. (1913). Psychology as the Behaviorist Views It. Psychological review, 20, 158-177.

Chapter 5

Specification and Verification of Dynamics in Cognitive Agent Models [1]

Abstract. Within many domains, among which biological and cognitive areas, multiple interacting processes occur among agents with dynamics that are hard to handle. Current approaches to analyse the dynamics of such processes, often based on differential equations, are not always successful. As an alternative to differential equations, this paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties. This language supports the specification of both qualitative and quantitative aspects, and therefore subsumes specification languages based on differential equations. A software environment has been developed for TTL, that supports editing TTL properties and enables the formal verification of properties against a set of traces. The TTL environment proved its value in a number of projects within different domains.

1 Introduction

In domains such as Biology and Cognitive Science, the dynamics of the multiple interacting processes among different agents involved poses modelling challenges. Currently, differential equations are among the techniques most often used to address this challenge, with partial success. For example, in the area of intracellular processes, hundreds or more reaction parameters (for which reliable values are rarely available) are needed to model the processes in question. Thus, describing these processes in

[1] This chapter appeared as Bosse, T., Jonker, C.M., van der Meij, L., Sharpanskykh, A., Treur, J.: Specification and Verification of Dynamics in Cognitive Agent Models. In: C. Butz, N.T. Nguyen, Y. Takama (eds), Proceedings of the 6th International Conference on Intelligent Agent Technology, IAT'06. IEEE Computer Society Press, 247-255 (2006) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

terms of differential equations can seriously compromise the feasibility of the model. Likewise, in the area of Cognitive Science, the Dynamical Systems Theory that is also based on differential equations (DST, see e.g., [21]), is well practiced and successful. However, the models typically only address lower-level agent cognitive processes such as sensory or motor processing. DST has less to offer for modelling the dynamics of higher-level processes with a mainly qualitative character, such as agent reasoning, complex task performance, and certain capabilities of language processing.

For formal qualitative modelling of processes at a high level of abstraction, logic-based methods have proved useful. For example, variants of modal temporal logic [2, 12, 14, 19, 24] gained popularity in agent technology. However, many of the logic-based methods lack the quantitative expressivity, needed, e.g., for modelling processes for which precise timing relations play an essential role (e.g., biological and chemical processes).

Thus, within several disciplines the need exists for general modelling and analysis techniques capable to deal with complex agent systems that comprise both quantitative and qualitative aspects. This paper introduces the Temporal Trace Language (TTL) as such a technique for the analysis of dynamic properties within complex domains, and especially, for the cognitive domain. In Section 2, a novel perspective is put forward for the development of such a technique, based on the idea of checking dynamic properties on given sets of traces. Section 3 shows how dynamics of an agent system can be modelled using the TTL language. Examples of the application of TTL are presented in Section 4. Section 5 describes the tools that support the TTL modelling environment in detail. In particular, the TTL Property Editor and the TTL Checker Tool are discussed. Section 6 is a conclusion.

2 Perspective of this Paper

As follows from the discussion above, the demands for dynamic modelling and analysis approaches suitable for specifying agent systems in natural domains are nontrivial. In particular, the possibility of both discrete and continuous modelling of a system at different aggregation levels is demanded. Furthermore, numerical expressivity is required for modelling systems with explicitly defined quantitative relations best presented by difference or differential equations. Moreover, for specifying qualitative aspects of a system, modelling languages should be able to express logical relationships between parts of a system.

Desiderata for analysis techniques include both the generation and formalisation of simulated and empirical trajectories or traces, as well as analysis of complex dynamic properties of such traces and relationships between such properties. A *trace* as used here represents a temporally ordered sequence of states of an agent system. Each state is characterised by a number of *state properties* that hold. Simulated traces may be obtained by performing simulations based on both quantitative (or continuous) and qualitative (or discrete) variables.

Taken together, the desiderata for modelling languages and analysis techniques described above are not easy to fulfil. On the one hand, high expressivity is desired, on the other hand feasible analysis techniques are demanded. To provide automated

support for these analyses the expressivity of the modelling language can be limited, thereby compromising the desiderata for modelling languages. For example, the expressivity may be limited to difference and differential equations as in DST (excluding logical relationships), or to propositional modal temporal logics (excluding numerical relationships). In the former case, calculus can be exploited to do simulation and analysis based on continuous variables only [21]. In the latter case, simulation is based on a specific logical executable format, which does not allow expressions involving continuous variables (e.g., executable temporal logic [2]). Another possibility is to use a number of dedicated formal languages with limited expressiveness and related to them analysis techniques for checking different particular static and dynamic aspects of a system (e.g., structural consistency of a model, dynamic aspects of execution), as proposed in the methodology for the development of correct software KORSO [11]. The languages used in this project describe different formats of system specifications, relations between them (e.g., by refinement based on proof obligations) and the temporal development of these specifications for all phases of the software life cycle. However, in order to guarantee the overall correctness of a system some properties are required to be expressed using more than one language with different types of semantics. Thus, the problem of verification across different not related proof systems arises that is not addressed in this project.

The problem of checking relationships between dynamic properties of a system, identified above as one of the desiderata for analysis techniques, is essentially the problem of justifying entailment relations between sets of properties defined at different aggregation levels of a system's representation. In general, entailment relations can be established either by logical proof procedures or by checking properties of a higher aggregation level on the set of all theoretically possible traces generated by executing a system specification that consists of properties of a lower aggregation level (i.e., by performing model checking [12, 19, 24]). To make it feasible to check relationships between dynamic properties, expressivity of the language for these properties has to be sacrificed to a large extent. However, checking properties on a given set of traces of practical size (instead of all theoretically possible ones), obtained empirically or by simulation, is computationally much cheaper. Therefore, in that case the language for these properties can be more expressive, such as the sorted predicate logic temporal trace language TTL described in this paper. TTL fulfils all of the identified above desiderata for modelling languages and can be used both for formalisation of empirical and simulated traces and for analysis of properties on traces. Although TTL cannot be used to generate traces by simulation, an executable sublanguage of TTL, such as LEADSTO, cf. [6], may be defined for this purpose. Moreover, decidable fragments of TTL may be defined for the analysis of relationships between dynamic properties of a system.

Finally, having a language for simulation and languages for analysis within one subsuming language also opens the possibility of having a declarative specification of a simulation model, and thus to involve simulation models in logical analyses.

3 A Language to Model Agent Behaviour

The Temporal Trace Language (TTL) presented here is developed from the assumption that the dynamics of an agent system can be described as evolution of states of agents and an environment over time, as for modal temporal logics, see e.g., [2, 12, 14, 19, 24]. TTL has some similarities with situation calculus, see [22] and event calculus, see [16]. Time in TTL is assumed to be linearly ordered and depending on the application, it may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form with a linear ordering. An agent interacts with a dynamic environment via its *input* and *output* (interface) states. At its input the agent receives observations from the environment whereas at its output it generates actions that can change a state of the environment.

An agent state at a certain point in time as used here is an indication of which of the state properties of the agent and its environment (e.g., observations and actions) are true (hold) at that time point. For specifying state properties for the input, output, internal, and external states of an agent A, ontologies, named IntOnt(A), InOnt(A), OutOnt(A), and ExtOnt(A) respectively, are used which are specified by a number of sorts, sorted constants, variables, functions and predicates (i.e., a signature). State properties are specified using a standard multi-sorted first-order predicate language based on such ontologies. For example, a state property expressed as a predicate pain may belong to IntOnt(A), whereas the atom has_temperature(environment, 7) may belong to ExtOnt(A).

To characterize the dynamics of the agent and the environment, *dynamic properties* relate properties of states at certain points in time.

To enable reasoning about the dynamic properties of arbitrary systems the language TTL includes special sorts, such as: TIME (a set of linearly ordered time points), STATE (a set of all state names of an agent system), TRACE (a set of all trace names; a trace or a trajectory can be thought of as a timeline with for each time point a state), and STATPROP (a set of all state property names). Throughout the paper, variables such as t, t1, t2, t', t" stand for variables of the sort TIME; and variables such as γ, γ1, γ2 stand for variables of the sort TRACE.

A state of an agent is related to a state property via the satisfaction relation $\models$ formally defined as a binary infix predicate (or by holds as a binary prefix predicate in the software environment). For example, "in the output state of agent A in trace γ at time t property p holds" is formalized by state(γ , t, output(A)) $\models$ p. If the indication of an agent aspect is not essential, the third argument is left out: state(γ, t) $\models$ p.

Both state(γ , t, output(A)) and p are terms of the TTL language. TTL terms are constructed by induction in a standard way for sorted predicate logic from variables, constants and functional symbols typed with TTL sorts. Dynamic properties are expressed by TTL-formulae inductively defined by:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds(v_1, u_1) is an atomic TTL formula.
(2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1 = \tau_2$ is an atomic TTL formula.
(3) If t_1, t_2 are terms of sort TIME, then $t_1 < t_2$ is an atomic TTL formula.
(4) The set of well-formed TTL-formulae is defined inductively in a standard way based on atomic TTL-formulae using boolean propositional connectives and quantifiers.

For example, the dynamic property

> 'in any trace γ, if at any point in time t1 agent A observes that it is dark in the room, whereas earlier a light was on in this room, then there exists a point in time t2 after t1 such that at t2 in the trace γ agent A switches on a lamp'

is expressed in formalized form as:

$$\forall t1\ [\ [\ state(\gamma, t1, input(A)) \models observed(dark\_in\_room)\ \&$$
$$\exists t0<t1\ [\ state(\gamma, t0, input(A)) \models observed(light\_on)]$$
$$\Rightarrow\ \exists t2 \geq t1\ state(\gamma, t2, output(A)) \models performing\_action(switch\_on\_light)\ ]$$

As TTL uses order-sorted predicate logic as a point of departure, it inherits the standard semantics of this variant of predicate logic. That is, the semantics of TTL is defined in a standard way, by interpretation of sorts, constants, functions and predicates, and a variable assignment. However, in addition the semantics involves some specialised aspects. As a number of standard sorts are present, the elements of these sorts are limited to instances of specified terms in these sorts, as is usual, for example, in logic programming semantics. For example, for the sort TIME it is assumed that in its semantics its elements consist of the time points of the fixed time frame chosen. Moreover, for the sort TRACE, it is assumed that in its semantics its elements consist of a (limited) number of elements named by constants. Furthermore, for the sort STATPROP for state properties it is assumed that in its semantics its elements consist of the set of terms denoting the propositions built in a chosen state language (this is called reification). A full description of the technical details of TTL's semantics is beyond the scope of the current paper. For this purpose, see [23].

By executing dynamic properties traces can be generated and visualised, for example as in Figure 1. Here, the time frame is depicted on the horizontal axis. The names of predicates are shown on the vertical axis. A dark box on top of the line indicates that the predicate is true during that time period.

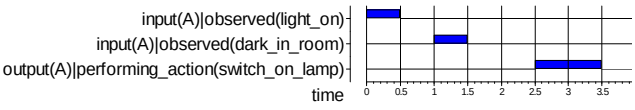


Fig. 1. Example visualisation of a trace

4 Application Areas

The TTL language and its supporting software environment have been applied in research projects addressing different topics in cognitive domains, such as human reasoning, conditioning, consciousness, psychotherapy, and philosophy of mind. The main research goal in these projects was to analyse the behavioural dynamics of the agents involved (e.g., [5, 7, 8, 9, 10]). TTL was used to formalise dynamic properties of these processes at a high level of abstraction. Next, such properties were automatically checked against simulated or empirical traces. Examples of the application of TTL in different areas are presented in this section.

4.1 Modelling and Analysis of Hybrid Systems

Hybrid systems incorporate both continuous and discrete components. The dynamics of the former can be described by differential equations, those of the latter can be represented by finite-state automata. Both continuous and discrete dynamics of components influence each other. In particular, the input to the continuous dynamics is the result of some function of the discrete state of a system; whereas the input of the discrete dynamics is determined by the value of the continuous state.

A modelling method for hybrid systems should be capable of expressing both quantitative and qualitative properties of the system and integrating them into one model. TTL satisfies this requirement. Systems of differential equations can be expressed in TTL using discrete or dense time frames. As an example, Euler's method, see [20], for solving differential equations is modelled in TTL. Euler's method approximates a differential equation $dy/dt = f(y)$ with the initial condition $y(t_0)=y_0$ by a difference equation $y_{i+1}=y_i+h*f(y_i)$ ($i \geq 0$ is the step number and $h>0$ is the integration step size). This equation can be modelled in TTL in the following way:

$\forall\gamma\ \forall t\ \forall v$: $\mathrm{LVALUE}^{\mathrm{GTERMS}}\ \mathrm{state}(\gamma, t) \models \mathrm{has\_value}(y, v) \Rightarrow$
$\mathrm{state}(\gamma, t+1) \models \mathrm{has\_value}(y, v + h \bullet f(v))$

States specify the respective values of y at different time points and the difference equation is modelled by a transition rule from the current to the successive state. The traces γ satisfying the above dynamic property are the solutions of the difference equation. More precise and stable numerical approximation methods (e.g., Runge-Kutta, dynamic step size, see [20]) can be expressed in TTL in a similar manner.

4.2 Analysis of Trace Conditioning in TTL

The example given in this section is taken from [5]. In that paper, TTL is used to analyse the temporal dynamics of trace conditioning. In general, research into conditioning is aimed at revealing the principles that govern associative learning. An important issue in conditioning processes is the adaptive timing of the conditioned response to the appearance of the unconditioned stimulus. This feature is most apparent in an experimental procedure called *trace conditioning*. In this procedure, a trial starts with the presentation of a *warning stimulus* (S1, comparable to a conditioned stimulus). After a blank interval, called the *foreperiod*, an *imperative stimulus* (S2, comparable to an unconditioned stimulus) is presented to which the participant responds as fast as possible. The *reaction time* to S2 is used as an estimate of the conditioned state of preparation at the moment S2 is presented. In this case, the conditioned response obtains its maximal strength, here called *peak level*, at a moment in time, called *peak time*, that closely corresponds to the moment the unconditioned stimulus occurs.

Machado [18] developed a basic model that describes the dynamics of these conditioning processes in terms of differential equations. The structure of this model is shown in Figure 2. The model posits a layer of *timing nodes* and a single *preparation node*. Each timing node is connected both to the next (and previous) timing node and to the preparation node. The connection between each timing node and the preparation node (called *associative link*) has an adjustable weight associated

to it. Upon the presentation of a warning stimulus, a cascade of activation propagates through the timing nodes according to a regular pattern. Owing to this regularity, the timing nodes can be likened to an internal clock or pacemaker. At any moment, each timing node contributes to the activation of the preparation node in accordance with its activation and its corresponding weight. The activation of the preparation node reflects the participant's preparatory state, and is as such related to reaction time. The weights reflect the state of conditioning, and are adjusted by learning rules, of which the main principles are as follows. First, *during* the foreperiod extinction takes place, which involves the decrease of weights in real time in proportion to the activation of their corresponding timing nodes. Second, *after* the presentation of the imperative stimulus a process of reinforcement takes over, which involves an increase of the weights in accordance with the current activation of their timing nodes, to preserve the importance of the imperative moment. Machado describes the more detailed dynamics of the process by a mathematical model (based on linear differential equations), representing the (local) temporal relationships between the variables involved. For example,

$$dX(t,n)/dt = \lambda X(t,n-1) - \lambda X(t,n)$$

expresses how the activation level of the n-th timing node $X(t+dt,n)$ at time point t+dt relates to this level $X(t,n)$ at time point t and the activation level $X(t,n-1)$ of the (n-1)-th timing node at time point t. Similarly, as another example,

$$dW(t,n)/dt = -\alpha X(t,n)W(t,n)$$

relates the n-th weight $W(t+dt,n)$ at time point t+dt to this weight $W(t,n)$ at time point t and the activation level $X(t,n)$ of the n-th timing node at time point t.

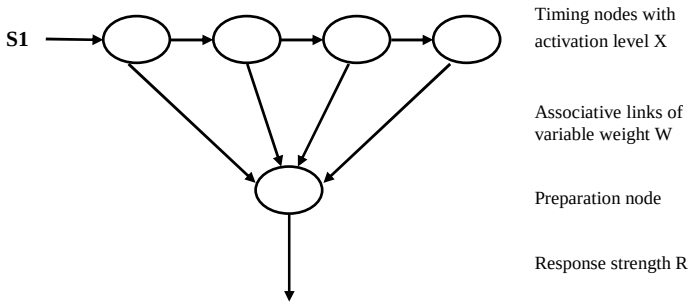


Fig. 2. Structure of Machado's conditioning model

In [5], a number of dynamic properties relevant for trace conditioning have been formalised in TTL. These properties were taken from the existing literature on conditioning, such as [17], in which they were mainly expressed informally. TTL turned out useful to express these properties in a formal manner. An example of such a property (taken from [17], p.372) is given below, both in informal, semi-formal and in formal notation:

Global Hill Preparation
Informal: 'The state of conditioning implicates an increase and decay of response-related activation as a critical moment is bypassed in time'.

Semi-formal: 'In trace γ, if at t1 a stimulus s1 starts, then the preparation level will increase from t1 until t2 and decrease from t2 until t1 + u, under the assumption that no stimulus occurs too soon (within u time) after t1.' Formally:

has_global_hill_prep(γ:TRACE, t1,t2:TIME, u:INTEGER) ≡
∀t',t":TIME ∀p',p":REAL
[state(γ, t1) ⊨ stimulus_occurs & ¬ stimulus_starts_within(γ, t1, t1+u) &
state(γ, t') ⊨ preparation_level(p') & state(γ, t") ⊨ preparation_level(p")
⇒ [t1 ≤ t' < t" ≤ t2 & t" ≤ t1 + u ⇒ p' < p"] &
[t2 ≤ t' < t" ≤ t1 + u ⇒ p' > p"]]

stimulus_starts_within(γ:TRACE, t1,t2:TIME) ≡
∃t:TIME [state(γ, t) ⊨ stimulus_occurs & t1 < t < t2]

These (and various similar) properties were automatically verified using the TTL checker tool against a number of (empirical and simulation) traces. Among these properties were also properties that compare different traces, such as: 'the conditioned response takes more time to build up and decay and its corresponding asymptotic value is lower when its corresponding critical moment is more remote from the warning signal.' (cf. [17])
Such properties cannot be expressed, for example, in modal temporal logics, just like familiar properties such as 'exercise improves skill', expressing that the more intensive a training history, e.g., of an athlete, the better the skill will be.

4.3 Application of TTL in Other Areas

Besides the conditioning area, TTL has been applied in many other domains as well. In order to give a representative overview in limited space, below a number of TTL formulae used in other domains are presented (both in informal and formal notation):

Proper Rejection Grounding
From the domain of human reasoning [10]:
'In any trace γ, if an assumption is rejected, then earlier on there was a prediction for it that did not match the corresponding observation result'.

∀t ∀A:INFO_EL ∀S1:SIGN
state(γ, t) ⊨ rejected(A,S1) ⇒
[∃t':TIME ∃B:INFO_EL ∃S2,S3:SIGN
state(γ, t') ⊨ prediction_for(B, S2, A, S1) & state(γ, t') ⊨ observation_result(B, S3) & S2 ≠ S3
& t' ≤ t1]

Representational Content of c
From a paper about representational content (cf. [15]) for the mental state of an agent that intensively interacts with the environment [9]:
'In any trace γ, internal state c occurs iff in the past once observation o1 occurred, then action a1(1), then o2(1), then a1(2), then o2(2), then a1(3), and finally o2(3)'.

∀t1,t2,t3,t4,t5,t6,t7 [t1≤t2≤t3≤t4≤t5≤t6≤t7
& state(γ, t1, input) ⊨ o1
& state(γ, t2, output) ⊨ a1(1) & state(γ, t3, input) ⊨ o2(1)
& state(γ, t4, output) ⊨ a1(2) & state(γ, t5, input) ⊨ o2(2)
& state(γ, t6, output) ⊨ a1(3) & state(γ, t7, input) ⊨ o2(3)
⇒ ∃t8 ≥ t7 state(γ, t8, internal) ⊨ c]
& ∀t8 [state(γ, t8, internal) ⊨ c ⇒
∃t1,t2,t3,t4,t5,t6,t7 t1≤t2≤t3≤t4≤t5≤t6≤t7≤t8
& state(γ, t1, input) ⊨ o1
& state(γ, t2, output) ⊨ a1(1) & state(γ, t3, input) ⊨ o2(1)
& state(γ, t4, output) ⊨ a1(2) & state(γ, t5, input) ⊨ o2(2)
& state(γ, t6, output) ⊨ a1(3) & state(γ, t7, input) ⊨ o2(3)]

Learning Behaviour of Aplysia
From a study [8] of adaptive processes of the sea hare Aplysia Californica [13]:
'In any trace γ, if a siphon touch occurs, and at three different earlier time points t1, t2, t3, a siphon touch occurred, directly followed by a tail shock, then the animal will contract'.

∀t [state(γ, t) ⊨ siphon_touch &
∃t1, t2, t3, t4, t5, t6
t1<t2 & t2<t3 & t3<t4 & t4<t5 & t5<t6 & t6<t &
state(γ, t1) ⊨ siphon_touch & state(γ, t2) ⊨ tail_shock & state(γ, t3) ⊨ siphon_touch & state(γ, t4) ⊨ tail_shock & state(γ, t5) ⊨ siphon_touch & state(γ, t6) ⊨ tail_shock]
⇒ ∃t7 t7≥t & state(γ, t7) ⊨ contraction

Food Delivery Succesfulness
From an analysis [7] of the domain of ant colony behaviour [4]:
'In any trace γ, there is at least one ant that brings food back to the nest'.

∃t ∃a:ANT ∃l:LOCATION ∃e:edge
state(γ, t) ⊨ is_at_location_from(a, l, e) & state(γ, t) ⊨ nest_location (l) & state(γ, t) ⊨ to_be_performed(a, drop_food)

5 Tools

This section presents the software environment[1] that was built in SWI-Prolog to support the process of specification and automated verification of dynamic properties on a limited set of traces. Basically, this software environment consists of two closely integrated tools: the Property Editor and the Checker Tool.

The Property Editor provides a user-friendly way of building and editing properties in TTL. By means of graphical manipulation and filling in forms a TTL specification can be constructed. TTL specifications may also be provided as plain text. When a TTL specification is created, the Checker Tool can be used to verify automatically whether a TTL property from the specification holds for a given set of traces. User interaction with the tools involves three separate actions:

1. Loading, editing, and saving a TTL specification in the Property Editor (see Figure 3).

[1] The software can be downloaded from the following URL: http://www.cs.vu.nl/~wai/TTL.

2. Loading and inspecting traces to be checked by activating the Trace Manager. Both, traces produced by simulations (see [6]) and empirical traces can be used for verification. Empirical traces provided to the TTL Checker may be obtained by formalizing empirical data from log-files produced by information systems or from results of experiments.
3. Checking a property against a set of loaded traces by the Checker Tool. The property is compiled and checked, and the result is presented to the user.

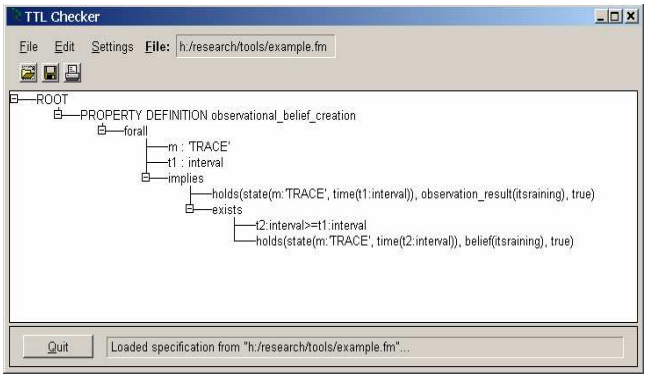


Fig. 3. The TTL Checking Environment

The following sections provide more details on implementation of these tools. In particular, Section 5.1 describes the implementation of the TTL Editor and Section 5.2 discusses the verification procedure underlying the TTL checker.

5.1 Implementation of the TTL Editor

A *TTL specification* constructed in the TTL Property Editor consists of a number of user-defined property definitions and sort definitions. A property definition consists of a header (property name and arguments, i.e., prop_name(v1:s1, v2:s2)) and a body (a TTL formula). Arbitrary sorts may be defined by enumerating their elements.

A *TTL formula* is constructed from atomic TTL formulae by conjunction, (Formula1 and Formula2), disjunction (Formula1 or Formula2), negation (not Formula), implication and quantification (forall ([v1:s1, v2:s2], Formula), exists ([v1:s1, v2:s2 < term2], Formula)).

Atomic TTL formulae correspond to user-defined properties, *holds atoms* (e.g., holds(state(trace1, t, output(ew)), a1 ∧ a2) or state(trace1, t, output(ew)) ⊨ a1 ∧ a2), mathematical expressions (e.g. term1 = term2, term1 > term2) and built-in properties (i.e., complex properties encoded into the implementation language).

All TTL formulae are constructed from *terms* that are implemented as Prolog terms (e.g., fn(t1,t2) , n1, t1 + t3, 1.3). Constants, variables and functions from which terms are constructed should be typed with appropriate sorts. For example, each variable should be declared as variable_name: sort. The software supports a number of built-in sorts, among which sorts for integer, real and range of integers (i.e., sorts

integer, real, between(i1:integer,i2:integer)), the sort for the set of all states (STATE) and the sort for the set of all traces (TRACE). Furthermore, libraries with predefined general purpose and domain-specific sorts and functions are available for creating terms.

5.2 Verification by the TTL Checker

After a TTL property is specified in the Editor and traces being loaded by the Trace Manager, the Checker Tool may be used to determine if the considered property holds on the loaded traces. To perform such verification an algorithm has been developed.

The verification algorithm is a backtracking algorithm that systematically considers all possible instantiations of variables in the TTL formula under verification. However, not for all quantified variables in the formula the same backtracking procedure is used. Backtracking over variables occurring in holds atoms is replaced by backtracking over values occurring in the corresponding holds atoms in traces under consideration. Since there are a finite number of such state atoms in the traces, iterating over them often will be more efficient than iterating over the whole range of the variables occurring in the holds atoms. Formulae that contain variables quantified over infinite sorts not occurring in a holds atom cannot be checked by the TTL Checker.

As time plays an important role in TTL-formulae, special attention is given to continuous and discrete time range variables. Because of the finite variability property of TTL traces (i.e., only a finite number of state changes occur between any two time points), it is possible to partition the time range into a minimum set of intervals within which all atoms occurring in the property are constant in all traces. Quantification over continuous or discrete time variables is replaced by quantification over this finite set of time intervals.

In order to increase the efficiency of verification, the TTL formula that needs to be checked is compiled into a Prolog clause. Compilation is obtained by mapping conjunctions, disjunctions and negations of TTL formulae to their Prolog equivalents, and by transforming universal quantification into existential quantification. Thereafter, if this Prolog clause succeeds, the corresponding TTL formula holds with respect to all traces under consideration.

The complexity of the algorithm has an upper bound in the order of the product of the sizes of the ranges of all quantified variables. However, if a variable occurs in a holds atom, the contribution of that variable is no longer its range size, but the number of times that the holds atom pattern occurs (with different instantiations) in trace(s) under consideration. The contribution of an isolated time variable is the number of time intervals into which the traces under consideration are divided.

The specific optimizations discussed above make it possible to check realistic dynamic properties with reasonable performance. In particular, checking the property ‘Learning Behaviour of Aplysia’ given in Section 4.3 (involving eight different time points) against a single trace with three state atoms occurring in the verified formula and 28 changes of atom values over time takes 0.76 sec. on a regular PC. With the increase of the number of traces with similar complexity as the first one, the verification time grows linearly: for 3 traces - 3.9 sec., for 5 traces - 6.59 sec.

However, the verification time is polynomial in the number of isolated time range variables occurring in the formula under verification.

6 Conclusion

This paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties of cognitive agent models. Although the language has a logical foundation, it supports the specification of both qualitative and quantitative aspects, and subsumes specification languages based on differential equations. TTL allows for explicit reference to time points and time durations, which enables modelling of the dynamics of continuous real-time phenomena. Furthermore, more specialised languages can be defined as a sublanguage of TTL. For the purpose of simulation, the executable language LEADSTO has been developed [6]. For verification of properties, different decidable fragments of predicate logic (e.g., [1]) can be defined as sublanguages of TTL.

TTL has some similarities with the situation calculus [22] and the event calculus [16], which are two well-known formalisms for representing and reasoning about temporal domains. However, a number of important syntactic and semantic distinctions exist between TTL and both calculi. In particular, the central notion of the situation calculus - a situation - has different semantics than the notion of a state in TTL. That is, by a situation is understood a history or a finite sequence of actions, whereas a state in TTL is associated with the assignment of truth values to all state properties (a "snapshot" of the world). Moreover, in contrast to the situation calculus, where transitions between situations are described by actions, in TTL actions are in fact properties of states.

Moreover, although a time line has been recently introduced to the situation calculus [22], still only a single path (a temporal line) in the tree of situations can be explicitly encoded in the formulae. In contrast, TTL provides more expressivity by allowing explicit references to different temporally ordered sequences of states (traces) in dynamic properties. For example, this can be useful for expressing the property of trust monotonicity:

'For any two traces $\gamma 1$ and $\gamma 2$, if at each time point t agent A's experience with public transportation in $\gamma 2$ at t is at least as good as A's experience with public transportation in $\gamma 1$ at t, then in trace $\gamma 2$ at each point in time t, A's trust is at least as high as A's trust at t in trace $\gamma 1$'.

$\forall \gamma 1, \gamma 2$
$[\forall t, \forall v1{:}VALUE\ [\ state(\gamma 1, t) \models has\_value(experience, v1)\ \&$
$[\forall v2{:}VALUE\ state(\gamma 2, t) \models [\ has\_value(experience, v2) \rightarrow v1 \leq v2\ ]]] \Rightarrow$
$[\forall t, \forall w1{:}VALUE\ [\ state(\gamma 1, t) \models has\_value(trust, w1)\ \&$
$[\forall w2{:}VALUE\ state(\gamma 2, t) \models [\ has\_value(trust, w2) \rightarrow w1 \leq w2\ ]]]]]$

Other examples of such properties, where different histories are compared are given in Section 4.2 above on trace conditioning.

In contrast to the event calculus, TTL does not employ the mechanism of events that initiate and terminate fluents. Events in TTL are considered to be functions of the external world that can change states of components, according to specified properties

of a system. Furthermore, similarly to the situation calculus, also in the event calculus only one time line is considered.

TTL can also be related to temporal languages that are often used for verification (e.g., propositional temporal logic (PTL) and linear-time logic (LTL) [3, 12, 14]). The general idea of translation of a LTL formula into a TTL expression is rather straightforward: by replacing the temporal operators of LTL by quantifiers over time. E.g., the following LTL formula

$$\boldsymbol{G}(\text{observation\_result(itsraining)} \rightarrow \boldsymbol{F}(\text{belief(itsraining)}))$$

where the temporal operator ***G*** means 'for all later time points', and ***F*** 'for some later time point' is translated into the following TTL expression:

$$\forall t1\ [\ \text{state}(\gamma, t1) \models \text{observation\_result(itsraining)} \Rightarrow$$
$$\exists t2 > t1\ \text{state}(\gamma, t2) \models \text{belief(itsraining)}\ ]$$

Note that the translation is not bi-directional, i.e., it is not always possible to translate TTL expressions into LTL expressions. An example of a TTL expression that cannot be translated into LTL is again the property of trust monotonicity.

Furthermore, TTL also allows expressivity provided by different extensions of PTL. In particular, the extended temporal logic (ETL) [25] provides a possibility to express any property definable by a regular expression on sequences of states, which cannot be expressed in PTL. Due to the fact that the syntax of TTL provides quantifiers, predicates, and arithmetic functions, such properties can be also expressed in TTL. For example, the property "a given proposition p has to be true in every even state of a sequence" can be expressed in TTL as follows: $\forall t\ \ \text{state}(\gamma, 2 \bullet t) \models p$.

To support the formal specification and analysis of dynamic properties in TTL, special software tools (the Property Editor and the Checker Tool) have been developed. The Property Editor has an intuitive graphical interface for building and editing TTL properties, and the Checker Tool employs an efficient algorithm for the formal verification of properties against a limited set of traces. Although this form of checking is not as exhaustive as model checking (which essentially means checking properties on the set of all traces generated by model execution), in return, it allows more expressivity in specifying properties.

The TTL environment has been tested and proved its value in a number of projects within different domains; e.g., [5, 7, 8, 9, 10]). During this work, the TTL environment has been further developed to provide automated support.

References

1. Andreka, H., Nemeti, I., and van Benthem, J. (1998). Modal Languages and Bounded Fragments of Predicate Logic. Journal of Philosophical Logic, 27(3): 217-274, 1998.
2. Barringer, H., M. Fisher, D. Gabbay, R. Owens, & M. Reynolds (1996). The Imperative Future: Principles of Executable Temporal Logic, Research Studies Press Ltd. and John Wiley & Sons.
3. Benthem, J.F.A.K., van (1983). The Logic of Time: A Model-theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse, Reidel, Dordrecht.
4. Bonabeau, J. Dorigo, M. and Theraulaz, G. (1999). Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York.

5. Bosse, T., Jonker, C.M., Los, S.A., Torre, L. van der, and Treur, J. (2005). Formalisation and Analysis of the Temporal Dynamics of Conditioning. In: Mueller, J.P. and Zambonelli, F. (eds.), *Proceedings of the Sixth International Workshop on Agent-Oriented Software Engineering, AOSE'05*, pp. 157-168.
6. Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J. (2005). LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. In: Eymann, T., et al. (eds.), *Proc. of the Third German Conference on Multi-Agent System Technologies, MATES'05*. LNAI, vol. 3550. Springer Verlag, pp. 165-178
7. Bosse, T., Jonker, C.M., Schut, M.C., and Treur, J. (2004). Simulation and Analysis of Shared Extended Mind. *Simulation Journal* (Transactions of the Society for Modelling and Simulation), vol. 81, 2005, pp. 719 - 732.
8. Bosse, T., Jonker, C.M., and Treur, J. (2006). An Integrative Modelling Approach for Simulation and Analysis of Adaptive Agents. In: *Proc. of the 39th Annual Simulation Symposium.* IEEE Computer Society Press, pp. 312-319.
9. Bosse, T., Jonker, C.M., and Treur, J. (2005). Representational Content and the Reciprocal Interplay of Agent and Environment. In: Leite, J., Omincini, A., Torroni, P., and Yolum, P. (eds.), *Proc. of the Second Int. Workshop on Declarative Agent Languages and Technologies, DALT'04*. LNAI, vol. 3476. Springer Verlag, pp. 270-288.
10. Bosse, T., Jonker, C.M., and Treur, J. (2006). Formalization and Analyisis of Reasoning by Assumption. *Cognitive Science Journal*, vol. 30, issue 1, pp. 147-180.
11. Broy, M., and Jahnichen, S. (1995). KORSO: Methods, Languages, and Tools for the Construction of Correct Software - Final Report. LNCS, vol. 1009. Springer Verlag.
12. Clarke, E.M., Grumberg, O., and Peled, D.A. (2000). Model Checking. MIT Press.
13. Gleitman, H. (1999). *Psychology*. W.W. Norton & Company, New York.
14. Goldblatt, R. (1992). Logics of Time and Computation, 2nd edition, CSLI Lecture Notes 7.
15. Kim, J. (1996). *Philosophy of Mind*. Westview Press.
16. Kowalski, R., and Sergot, M. (1986). A logic-based calculus of events, New Generation Computing, 4: 67-95.
17. Los, S.A. and Heuvel, C.E, van den. (2001). Intentional and Unintentional Contributions to Nonspecific Preparation During Reaction Time Foreperiods. *Journal of Experimental Psychology: Human Perception and Performance*, vol. 27, pp. 370-386.
18. Machado, A. (1997). Learning the Temporal Dynamics of Behaviour. *Psychological Review*, vol. 104, pp. 241-265.
19. Manna, Z., and Pnueli, A. (1995). Temporal Verification of Reactive Systems: Safety. Springer Verlag.
20. Pearson, C.E. (1986). Numerical Methods in Engineering and Science. CRC Press.
21. Port, R.F., Gelder, T. van (eds.) (1995). Mind as Motion: Explorations in the Dynamics of Cognition. MIT Press, Cambridge, Mass.
22. Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, Cambridge MA: MIT Press.
23. Sharpanskykh, A. and Treur, J. (2005). Verifying Interlevel Relations within Multi-Agent Systems: Formal Theoretical Basis, Technical Report TR-1701AI. VU Amsterdam, 2005. http://hdl.handle.net/1871/9777
24. Stirling, C. (2001). Modal and Temporal Properties of Processes. Springer Verlag.
25. Wolper, P. (1983). Temporal logic can be more expressive. *Information and Control*, vol. 56(1-2), pp. 72-99

Part III

Methods for Modeling and Analysis of Organizations

This part presents a detailed description of the proposed organization modeling and analysis methods. The methods are described along the modeling views identified in the introduction.

The performance-oriented view is presented in Chapters 1 and 2. Chapter 1 describes a formal language for modeling organizational performance indicators, some related verification techniques and methodological issues of creating and revising performance-oriented specifications. In Chapter 2 a formal framework for modeling goals based on performance indicators is described. Further, this chapter considers methodological and analysis issues related to goals. Both chapters contain formal definitions of the concepts and the relations of the performance-oriented view, a well as a set of axioms that represent special types of generic constraints and describe the rules of the correct use of the identified concepts and relations.

The process-oriented view described in Chapters 3 and 4 contains information about the organizational functions, how they are related, ordered and synchronized and the resources they use and produce. Furthermore, it addresses the actual execution of processes in organizational scenarios. In Chapter 3 the formal process-oriented modeling language L_{PR} is introduced. Furthermore, this Chapter identifies a set of generic and domain-specific constraints for the process-oriented view and introduces the automated verification techniques for establishing correctness of organizational specifications with respect to these constraints. Chapter 4 presents formal techniques for analysis of executions of organizational scenarios based on process-oriented models of organizations specified using the language L_{PR}. For the formalization of executions, a dedicated sorted predicate language L_{EX} is used, which is based on L_{PR}. A part of the introduced analysis techniques is dedicated to establishing the correspondence between formalized executions (i.e., traces) and process-oriented specifications. Other techniques provide the analyst with wide possibilities to analyze organizational dynamics and to evaluate organizational performance. The type of analysis used in this Chapter is based on the general method of checking dynamic properties on a set of traces introduced in Chapter 5 of Part II.

Note that both constraints defined in the process-oriented view and properties checked on executions of organizational scenarios may be specified using concepts and relations of other views. For this a number of predicates that relate different views are introduced both in the language L_{PR} and in the language L_{EX}. An illustration of how the concepts and relations of the process- and performance-oriented views are used together for the organizational performance evaluation is given in Chapter 4.

Chapters 5 and 6 introduce the organization-oriented view. Chapter 5 describes structural *subrole*-relations between roles at different aggregation levels and dynamic aspects of interaction between roles. The sorted first-order predicate logic with finite sorts is used for the formalization of structural properties of the view, whereas the TTL is used for expressing rules of interaction between roles. Furthermore, the application of two analysis techniques (trace-based analysis and verification of relations between different aggregation levels) introduced in Chapters 4 and 5 of Part II in the context of the organization-oriented view is demonstrated in Chapter 5. The described analysis concerns domain-specific properties mostly. The generic integrity constraints on subrole relations and interaction relations between roles will be described in Part IV. Chapter 6 considers formal authority relations on roles that include responsibility relations for organizational tasks and resources, superior-

subordinate relations and others. Furthermore, Chapter 6 introduces a set of constraints on the concepts and relations both from the organization-oriented view and accross multiple views. Moreover, both Chapters 5 and 6 establish relations with the agent-oriented view by defining relationships between roles and agents and by specifying constraints over agents.

Chapter 7 describes an approach for modeling the characteristics and behavior of agents situated in a formally specified organization. The approach is based on the theoretical findings from social science and enables analysis of how different organizational and environmental factors influence the behavior and performance of agents.

In Chapter 8 the complexity monotonicity thesis is proposed that establishes a relation between the environmental complexity and the behavioral complexity of an agent situated in this environment. More specifically, for more complex environments, more complex behaviour and more complex mental capabilities of agents are needed. This thesis is tested in a number of example scenarios of animal behavior. Organizations can be also considered as structures that often have to react on and enhance internal complexity of the environment, with which organisms need to cope. Therefore, some results presented in Chapter 8 can be also applied in organizational context. However, a more detailed investigation is still required.

Chapter 1

Modeling Organizational Performance Indicators[1]

Abstract. Performance measurement and analysis is crucial for steering the organization to realizing its strategic and operational goals. Relevant performance indicators and their relationships to goals and activities need to be determined and analyzed. Current organization modeling approaches do not reflect this in an adequate way. This paper attempts to fill the gap by presenting a framework for modeling performance indicators within a general organization modeling framework.

1 Introduction

Measuring and analyzing organizational performance plays an important role in turning organizational goals to reality. The performance is usually evaluated by estimating the values of qualitative and quantitative performance indicators (e.g., profit, number of clients, costs). It is essential for a company to determine the relevant indicators, how they relate to the formulated company goals and how they depend on the performed activities. In practice such analysis is usually done in an informal, ad-hoc way. This paper introduces a framework for modeling performance indicators and the relationships between them, which constitutes a part of a general framework for organization modeling and analysis.

In the general framework, organizations are considered from different perspectives (views). *Process-oriented view* describes the workflow as well as static structures of tasks and resources. *Performance-oriented view* is characterized by a goal structure, a

[1] Part of this chapter appeared as Popova, V., Sharpanskykh, A.: Modelling Organizational Performance Indicators. In: Barros, F. et al. (eds.): Proceedings of the International Modeling and Simulation Multiconference IMSM'07, SCS Press, 165-170 (2007) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

performance indicators structure, and relations between them as well as relations between goals and tasks, performance indicators and processes, goals and roles or agents. *Organization-oriented view* defines the organizational roles, each associated with a set of tasks and characterized by authority and responsibility relations on tasks, resources and information. Commitment, obligation and power relations and sets of competences required for agent allocation to roles are also defined. *Agent-oriented view* identifies different types of agents with their characteristics and behavior, and principles for allocating agents to roles based on the matching between agent capabilities and competences required for roles.

The formal language and axiomatic basis for *modeling performance indicators* within the *performance-oriented view* are described in this paper as well as the performance evaluation process and methodological issues of creating and revising performance-oriented models. Some verification techniques specific for performance-oriented organization models are briefly discussed. The presentation is organized as follows. The case study used for illustration is described in Section 2. In Section 3 the main concepts are defined. The relationships between them are defined in Section 4 as well as the semantic aspects of the introduced language with its axiomatic basis. Section 5 discusses the evaluation of organizational performance. Methodological guidelines are given in Section 6. Section 7 discusses related work on performance measurement. Section 8 concludes the paper with a summary and future research directions.

2 Introduction to the Case Study

The proposed approach is applied for modeling and analyzing an organization from the security domain. The main purpose of the organization is to deliver security services (e.g., private property surveillance, safeguard) to different types of customers (individual, firms and enterprises). The organization has well-defined structure with predefined (to a varying degree) job descriptions for employees. The total number of employees in the organization is approximately 230.000 persons working in several regions. The global management of the organization is performed by the board of directors, which includes among others the directors of the different divisions (corresponding to the different regions). Within each region a number of areas exist controlled by area managers. An area is divided into several units, controlled by unit managers. Each unit serves a number of locations, for which the contracts with customers have been signed and security officers are allocated. The allocation of employees is performed based on plans created by planning groups.

The model that corresponds to the part of the organization concerned with the planning process will be used in this paper to illustrate concepts, relations and techniques related to the performance-oriented view. Therefore, the planning process is described here in more detail. The planning process consists of forward (or long-term) planning and short-term planning. Forward planning is the process of creation, analysis and optimization of forward plans that describe the allocation of security officers within the whole organization for a long term (4 weeks). Forward plans are created based on customer contracts by forward planners from the forward planning

group. During the short-term planning, plans that describe the distribution of security officers to locations within a specific area for a short term (a week) are created and updated based on the forward plan and the available up-to-date information about the security employees. Furthermore, based on short term plans, daily plans are created. Within each area short-term planning is performed by the area planning team that consists of planners and is guided by a team leader. During the planning process short-term planners interact actively with forward planners (e.g., for consultations, problem solving). Furthermore, forward planners have a number of supervision functions with respect to short-term planners.

3 Performance Oriented Concepts

Every organization exists for the achievement of one or more goals. This varies depending on the type of organization, e.g. the main goal of a manufacturing company can be the realization of maximal profit while the goal of a non-profit organization can be to effectively protect wild animals. Being aware of these goals is a prerequisite to taking measures for their satisfaction. To ensure continued success, the organization should monitor its performance with respect to its goals. The notions of a goal and a performance indicator are therefore essential. They are the main building blocks of the performance-oriented view of our approach.

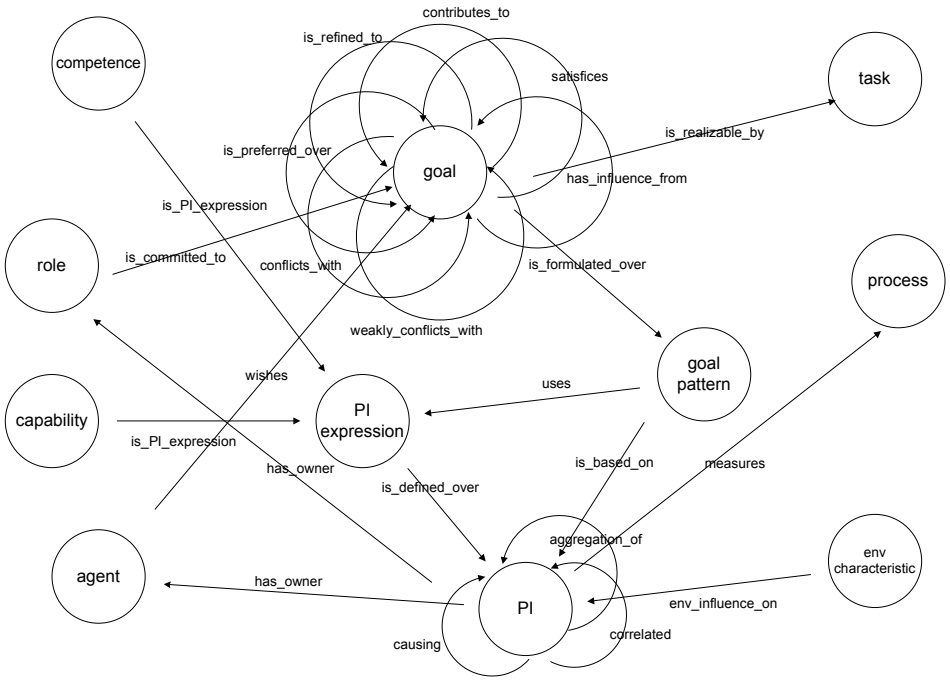


Fig. 1. A meta-model for the *performance-oriented view*.

Fig.1 gives a graphical representation of the concepts and relationships of the *performance-oriented view*. This paper describes a modeling approach and a formal language for PIs as part of the performance-oriented view. Other concepts and views will be considered elsewhere.

In the following paragraphs, the definitions for performance indicators and performance indicator expressions are given with their specific characteristics.

Performance indicator – quantitative or qualitative indicator that reflects the state/progress of the company, unit or individual. The following characteristics can be specified for each performance indicator:

Name;

Definition;

Type – continuous or discrete;

Time frame – (if applicable) for which time frame is the performance indicator defined, the length of the time interval for which it will be evaluated, e.g. the indicator 'yearly profit' has time frame 'year', 'number of customers per day' has time frame 'day';

Scale – if relevant, the measurement scale for the performance indicator, different scales can be predefined and referred to here;

Min value, Max value – when a predefined scale is used and only a part of this scale is relevant for the particular performance indicator;

Source – which was the internal or external source used to extract the performance indicator: company policies, mission statements, business plan, job descriptions, laws, domain knowledge, etc. – these sources contain (informal) statements about the desired state or behavior of the company and regulations it has to obey;

Owner – the performance of which role or agent does it measure/describe;

Threshold – the cut-off value separating changes in the value of the performance indicator considered small and changes considered big; used to define the degree of influence of between performance indicators (see Section 4);

Hardness – a performance indicator can be soft or hard where soft means not directly measurable, qualitative, e.g. customer's satisfaction, company's reputation, employees' motivation, and hard means measurable, quantitative, e.g., number of customers, time to produce a plan.

Example:
PI name: **PI5**;
Definition: average correctness of produced plans
Type: discrete;
Time frame: month;
Scale: very_low-low-med-high-very_high;
Source: mission statement, job descriptions;
Owner: forward and daily planning departments
Threshold: 2 units;
Hardness: soft;

PI name: **PI27**;
Definition: time to create a new short-term plan after all operational data is received
Type: continuous;

Time frame: month
Scale: REAL
Min value: 0;
Max value: max_time_CSP;
Unit: hour;
Source: job descriptions
Owner: daily planning departments
Threshold: 24 hours;
Hardness: hard;

PI name: **PI29**;
Definition: efficiency of allocation of security officers to objects;
Type: discrete;
Time frame: month
Scale: very_low-low-med-high-very_high;
Source: job descriptions
Owner: forward and daily planning departments
Threshold: 24 hours;
Hardness: soft;

Appendix A contains the list of performance indicators identified for the case study for the forward and daily planning departments with the corresponding characteristics.

The set of performance indicators that can be defined for one organization can be very large and it is often not feasible to monitor all of them. Therefore the companies select a subset of indicators, called *key performance indicators*, that can give a representative picture of the performance and the costs of measuring and monitoring are reasonable. It is essential for the company to choose its key performance indicators carefully to form a balanced (with respect to the company activities, involved parties, etc.) and sufficiently complete set [6]. The key performance indicators of the organization should be reflected in its goals.

The process of extracting the performance indicators from source documents involves asking the question: What should be measured / observed to ensure the requirements in the document? Performance indicators are often represented by nouns in the text; modifiers such as adjectives give information about the type, scale of measurement and what is considered a desirable value of the performance indicator (used in performance indicator expressions, goal patterns and goals), e.g., the job description requires a planner ‘to ensure high accuracy of calculation when creating a plan’ then ‘accuracy of calculation in plan creation’ is a performance indicator and ‘high’ – its desired value.

Often the performance indicators that can be extracted from documents such as the mission statements and policies are soft and difficult to assess. In order to evaluate such a performance indicators it is usually beneficial to find a closely related hard indicator that can be measured instead and that can give an impression on the state of the soft one. For example customer satisfaction cannot be measured directly but it is possible to design questionnaires that will be used to collect information on customer’s opinion and classify it in predefined ranges (high, medium, etc.). The results from such a study give an impression on the actual degree of satisfaction but it

is important to note that the actual degree of satisfaction might deviate from the calculated value and is not directly measurable. The domain knowledge used is that for properly designed questionnaires, there is correlation between the degree of satisfaction and the results from the study.

The second concept relevant for modeling performance indicators is the performance indicator expression which is defined below.

Performance indicator expression – a performance indicator or a mathematical statement over a performance indicator containing >, ≥, =, < or ≤. A performance indicator expression can be evaluated to a numerical, qualitative or Boolean value for a time point, for the organization, unit or agent. For example using the above defined performance indicators we can formulate performance indicator expressions as follows: PI27 ≤ 48h; PI5 = high.

Within the performance-oriented view also goal patterns and goals are defined which are addressed in details in [10]. In the following paragraph a brief overview is given.

Performance indicator expressions are used to define goal patterns which are properties that can be checked to be true or false for the organization, unit or individual at a certain time point or period. For example a goal pattern based on PI27 can be GP1:'achieved that PI27 ≤ 48h'. Goals are objectives that describe a desired state or development and are defined by adding to goal patterns information such as desirability and priority. A goal based on GP1 can be G1:'It is required to achieve that PI27 ≤ 48h'. Goals can be hard (satisfaction can be clearly established) or soft (satisfaction cannot be clearly established). For soft goals degrees of *satisficing* are defined. Goals can be organizational (i.e., belong to an organization, unit or role) or individual (i.e., belong to an agent). Individual goals may comply with, be disjoint or conflict with organizational goals. This can be determined by analyzing the relations between the performance indicators on which the goals are based (see Section 6). Goal can be refined into subgoals forming a goals hierarchy. Information about the satisfaction of lower level goals can be propagated to determine satisfaction of high level goals (see Section 5 which discusses the evaluation of organizational performance).

The performance-oriented concepts are related to *other views* in the following way. Goals are realized by performing organizational functions described by *tasks*. *Processes* are specific instances of tasks temporally ordered in a workflow and performed by roles. Performance indicators are associated to specific aspects of the execution of particular processes. A *role* represents a predefined set of functionalities performed within the organization which can be allocated to *agents*. Roles and agents can be committed to organizational or individual goals respectively. Roles are characterized by sets of *competences*, required to perform a certain task. Competences can be credentials (i.e., material or digital objects certifying accomplishments; e.g., diplomas, certificates), and skills (i.e., abilities that can be demonstrated, e.g., typing speed, flexibility). Skills are formulated as performance indicator expressions over individual performance indicators. Agents are autonomous entities, characterized by their individual goals and capabilities. Individual goals of agents are based on individual performance indicators. *Capabilities* can be credentials or skills that are possessed by agents. Skills are formulated as performance indicator expressions over

individual performance indicators. An agent can only play a role if it has the capabilities to match the competences required for the role.

4 Modeling Relationships between Performance Indicators

The formal language used for specifying the meta-model for the performance-oriented view is a variant of the first order sorted predicate language. In this language, for each concept a special sort is introduced, containing all the names of concept instances (e.g., sort PI contains all names of performance indicators). The characteristics (attributes) of the concepts are represented by relations (predicates) with arguments: a concept name, an attribute name and a value the attribute (e.g., ***has_attribute_value***: PI × ATTRIBUTE × VALUE). In the following for readability such predicates are used in the more compact form: concept.attribute=value. Specific values that have been measured for performance indicators during or after the execution of organizational processes can be specified using the predicate ***PI_has_value***: PI × PI_VALUE where the sort PI_VALUE includes the sort VALUE together with all possible evaluations of soft performance indicators (e.g. low, medium, high). For more details on recording and analyzing the execution of organizational processes and on evaluating performance indicators the reader is referred to [9]. In the following paragraphs, the relations between performance indicators are defined. In order to provide formal meaning and to enable formal verification (e.g., consistency or integrity checking), the axiomatic basis is also defined.

causing: PI × PI × {very_pos, pos, neg, very_neg}: The first performance indicator causes change in the same direction (positive) or opposite direction (negative) to the second performance indicator. Very_positive describes the situation when small change in one performance indicator causes big change in the other. Similarly for very_negative. The distinction between small and big change can be subjective and therefore should be defined carefully by the designer using input from domain experts. It is specific for each performance indicator and is specified in the model by the threshold values assigned to performance indicators. When the value of a performance indicator increases or decreases, positive or negative difference can be calculated and compared to the threshold value to determine whether it is considered a small or big change. This informal explanation of the causality relation can be formalized as follows using the Temporal Trace Language [12] (p1 and p2 are variables over sort PI):

causing(p1, p2, pos) iff:
∀γ ∀t ∀a,b:PI_VALUE state(γ,t)|= [PI_has_value(p1, a) ∧ PI_has_value(p2, b)] ⇒
∀t1>t [∀c:PI_VALUE state(γ,t1)|= [PI_has_value(p1, c) ∧ c>a] ⇒
∃t2≥t1 ∃d:PI_VALUE state(γ,t2)|= [PI_has_value(p2, d) ∧ d>b]] & [∀e:PI_VALUE state(γ,t1)|= [PI_has_value(p1, e) ∧ e<a] ⇒
∃ t2≥t1 ∃f:PI_VALUE state(γ,t2)|= [PI_has_value(p2, f) ∧ f<b]]

causing(p1, p2, very_positive) iff:
∀γ ∀t ∀a,b:PI_VALUE state(γ,t)|= [PI_has_value(p1, a) ∧ PI_has_value(p2, b)] ⇒
∀t1>t [∀c:PI_VALUE state(γ,t1)|= [PI_has_value(p1, c) ∧ c>a ∧ c-a<p1.threshold] ⇒
∃t2≥t1 ∃d:PI_VALUE state(γ,t2)|= [PI_has_value(p2, d) ∧ d>b ∧ d-b>p2.threshold]] & [∀e:PI_VALUE state(γ,t1)|= [PI_has_value(p1, e) ∧ e<a ∧ a-e<p1.threshold] ⇒

$\exists t2 \geq t1 \ \exists f{:}PI\_VALUE\ state(\gamma, t2) \models [PI\_has\_value(p2, f) \wedge f < b \wedge b\text{-}f > p2.threshold]]$

The causality relations for the negative and very_negative cases are defined in a similar manner.

correlated: PI × PI × {pos, neg}: The first performance indicator is correlated positively or negatively to the second performance indicator, i.e., changes in the first performance indicator result in changes in the second one in the same (pos) or opposite (neg) direction and the other way round. This is defined by the following axiom:

correlated (p2, p1, pn), where pn:{pos, neg} iff:
causing(p1, p2, pn) & causing(p2, p1, pn)

aggregation_of: PI × PI: The first performance indicator is an aggregation of the second performance indicator. If the aggregation relation exists between performance indicators, then these performance indicators are also positively correlated with each other.

$\forall$ p1,p2:PI: aggregation_of(p1, p2) $\Rightarrow$ correlated (p1, p2, pos)

Both performance indicators in the aggregation relation have the same type and unit. This is expressed by the following axiom:

$\forall$ p1,p2:PI: aggregation_of(p1, p2) $\Rightarrow$ p1.type=p2.type & p1.unit=p2.unit

The aggregation relation exists for example between performance indicators of the same type with time frame attributes related by the aggregation relation, e.g., performance indicator 'revenue for a year' is an aggregation for performance indicator 'revenue for a month'. Aggregation relation between performance indicators can be defined based on the relations of performance indicators to processes and organizational roles. More specifically, the performance indicators of the same type are related by aggregation, when their owners (roles, agents) are related by the structural aggregation relation is_part_of, e.g., is_part_of(group1,deptA). For example performance indicator 'number of planners in deptA' is an aggregation of performance indicator 'number of planners in group1'. Similarly if performance indicators of the same type measure the same aspect of execution of process instances of tasks related by is_subtask_of relation, e.g., is_subtask_of(collect_data, create_plan) then often aggregation relation exists between these performance indicators.

Using the standard procedure from the sorted first-order predicate logic, terms and formulae over sort PI can be built, expressing different types of mathematical relations between performance indicators. For example, organizational_profit=organizational_revenue-organizational_costs; (PI1>3 & PI2=4.5) $\Rightarrow$ PI3 > 5.2.

In the following more detailed examples are given in the frames of the case study using some of the performance indicators defined in Appendix A.

Examples:

Name: **PI1**

Definition: The level of correctness of plans with respect to the contracts of the employees, the laws, the general policy of the company and division

Type: discrete
Time frame: month
Scale: very_low-low-medium-high-very_high
Source: mission statement, job descriptions
Owner: forward and daily planning departments
Threshold: 2 units
Hardness: soft

Name: **PI2**
Definition: the level of knowledge of employees involved in (forward) planning about the current contracts of the employees, the laws, the general policy of the company and division
Type: discrete
Time frame: month
Scale: very_low-low-medium-high-very_high
Source: mission statement, job descriptions
Owner: forward and daily planning departments
Threshold: 2 units
Hardness: soft

For these two performance indicators the following relation was discovered:
causing(PI2, PI1, pos)

Name: **PI30**
Definition: average level of optimality of forward, short-term and daily planning for efficient allocation of security officers
Type: discrete
Time frame: month
Scale: very_low-low-medium-high-very_high
Source: job descriptions
Owner: forward and daily planning departments
Threshold: 2 units
Hardness: soft

Name: **PI31**
Definition: average level of optimality of every forward plan for efficient allocation of security officers
Type: discrete
Time frame: month
Scale: very_low-low-medium-high-very_high
Source: job descriptions
Owner: forward planning department
Threshold: 2 units
Hardness: soft

For these two performance indicators the following relation was discovered:
aggregation_of(PI30, PI31)

Figure 2 shows the structure containing the main relationships between the performance indicators that were identified for the case study.

Performance indicators relate to tasks, processes, roles, agents by the following relations:

has_owner: PI × {ROLE, AGENT}: A performance indicator measures/describes the performance of a role or agent. Roles can be atomic or composite at any level including the level of the organization.

measures: PI × PROCESS: A performance indicator expresses an aspect of the performance of the process execution, e.g. 'time to produce a daily plan' measures the time performance of the execution of the process 'produce a daily plan', 'production costs' measures the cost performance of the general process 'production'.

Environmental conditions influence the execution of processes of an organization, thereby, also influence values of performance indicators related to these processes. This influence can be positive or negative and is specified by the following relation:

env_influence_on: ENV_CHARACTERISTIC × PI × {pos, neg}: An environmental characteristic of the sort ENV_CHARACTERISTIC influences a performance indicator in a positive or negative way (i.e., contributes to the increase/decrease of a performance indicator). For example, a large amount of rain contributes negatively to the amount and quality of harvest.

is_defined_over: PI_EXPRESSION × PI: A performance indicator expression is defined over a performance indicator.

Other types of relations between performance indicator, processes and roles, related to power, supervision, authorization, etc. will be discussed in the organization-oriented view.

5 Performance Evaluation

Every task in an organization contributes to the satisfaction of one or more organizational goals through performing its process instances. Each goal is formed based on a certain performance indicator(s) which can be measured (directly or indirectly) during or after the process execution depending on the goal evaluation type – in the end or during a certain period of time (evaluation period defined as goal horizon). The satisfaction (degree of satisficing) of the goal(s) is determined by comparing the measured value(s) with the corresponding goal expression(s). Further, the obtained goal satisfaction (satisficing) measure is propagated by applying the rules defined in [10], upwards in the goal hierarchy for determining the satisfaction (degree of satisficing) of higher level goals. Thus, the organizational performance is evaluated by determining the satisfaction (degree of satisficing) of key organizational goals. The same principles can be applied for evaluation of agent performance.

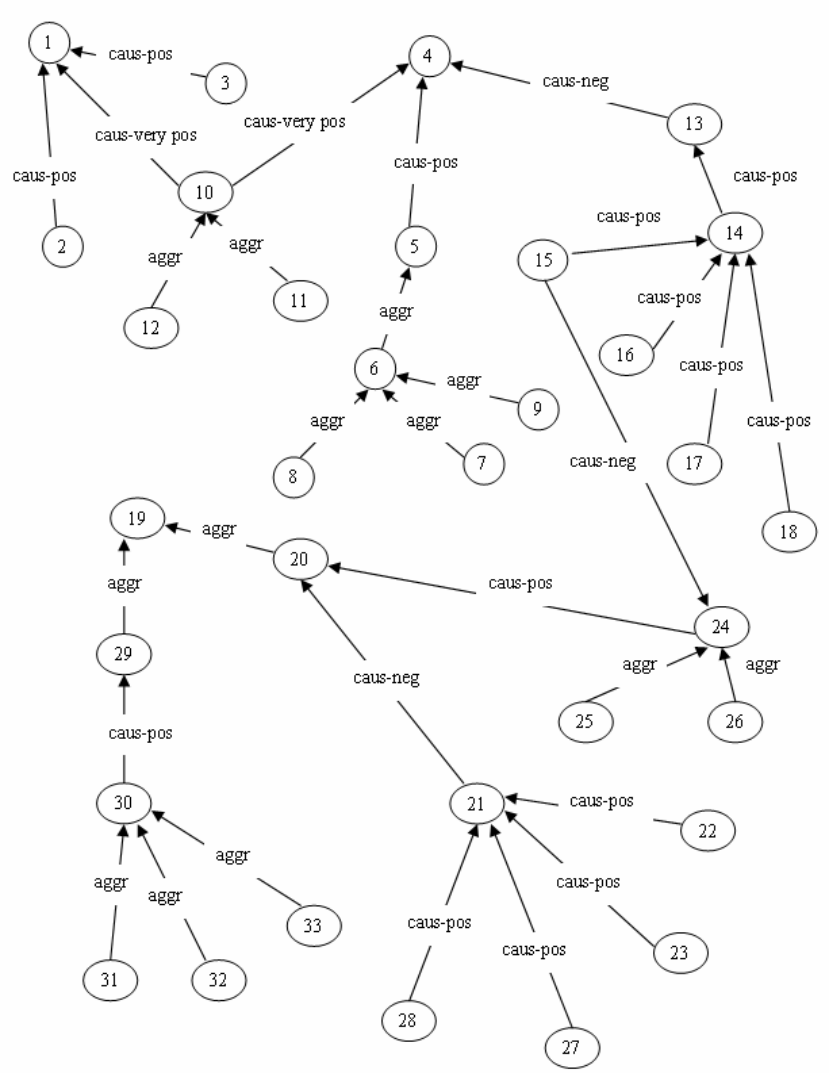


Fig. 2: The relationships between the performance indicators identified for the case study

As illustration of the proposed performance evaluation procedure consider the following example. For estimating the performance of the organization from the case study, the satisfaction of the key goal G3.1: 'It is required to maintain high efficiency of the planning process' has to be determined. One of the goals in its refinement is G3.1.1 'It is required to achieve that the number of times the planning activities (creating and updating of a plan) exceed the allowed durations is equal to 0'. This goal is refined into four goals: G3.1.1.1: 'It is required to achieve that the time to update a short-term plan given operational data is at most 48 hours', G3.1.1.2: 'It is required to achieve that the time to create a daily plan given operational data is at

most 24 hours', G3.1.1.3: 'It is required to achieve that the time to create a short-term plan after all operational data is received is at most a week' and G3.1.1.4: 'It is required to achieve that the time to create a forward plan after all operational data is received is at most a week'. These goals are related to tasks: G3.1.1.1 is realized by the task 'update_shortterm_plan', G3.1.1.2 is realized by the task 'create_daily_plan', G3.1.1.3 by 'create_shortterm_plan' and G3.1.1.4 by 'create_forward_plan'. By measuring the actual execution of the process instances of these tasks, it is determined that the values of the related performance indicators of these goals (PI22: 'time to update short term plan', PI23: 'time to create daily plan', PI27: 'time to create short term plan', and PI28: 'time to create forward plan') do not exceed the prescribed durations. Thus, goals G3.1.1.1, G3.1.1.2, G3.1.1.3 and G3.1.1.4 are satisfied. Due to the refinement relation G3.1.1 is also satisfied and contributes positively to the satisfaction of G3.1 and thus to the overall performance evaluation.

6 Methodological and Analysis Issues

Methodological issues discussed in this Section concern the construction and the revision of the PIs structures. As it was discussed in Section 3, organization's performance indicators can be extracted from different sources. To build a structure of performance indicators, relations between them should be identified, for which: (1) original documents can be analyzed for finding explicit references to such relations; (2) knowledge of domain experts and existing libraries of relations between performance indicators may be used; (3) performance indicators attributes and relations between these attributes (e.g., relations between time-related attributes and attributes that relate performance indicators to the organization and task structures) can be exploited (see Section 4); (4) from the existing relations in the performance indicators structure new relations may be inferred; (5) data mining techniques may be applied to the data collected during the organization operation; (6) intuitions of the modeler may be used after testing by domain experts or simulations; (7) relations to the task structure and the goal structure may be exploited.

As it follows from the definitions in Section 4 all the considered types of relations between performance indicators can be reduced to causality relations. Technique (4) allows inference of some missing causality relations from the existing performance indicators structure. In general the inference rules (i.e., the generic constraints on PIs) are specified in the form

$$\text{causing}(p1, p2, s1) \,\&\, \text{causing}(p2, p3, s2) \Rightarrow \text{causing}(p1, p3, s3),$$

where p1, p2 belong to the sort PI and s1, s2, s3 are of sort SIGN={very_neg, neg, pos, very_pos}. More specific (instantiated) inference rules are generated based on Table 1, in which s3 values are given in the cells on the intersection of columns containing s1 values with rows containing s2 values. These inference rules can also be used for the verification of integrity of the performance indicators structure.

Table 1. Inference Rules for Causal Relationships

s2 \ s1	Very neg	Neg	Pos	Very pos
Very neg	Very pos	Very pos	Very neg	Very neg
Neg	Very pos	Pos	Neg	Very neg
Pos	Very neg	Neg	Pos	Very pos
Very pos	Very neg	Very neg	Very pos	Very pos

Examples:

Name: **PI1** – as defined earlier

Name: **PI10**
Definition: level of correctness of administrative processing of all planning data in the system.
Type: discrete
Time frame: month
Scale: very_low-low-medium-high-very_high
Source: job descriptions
Owner: forward and daily planning departments
Threshold: 2 units
Hardness: soft

Name: **PI12**
Definition: level of correctness of administrative processing of short-term and daily planning data in the system.
Type: discrete
Time frame: month
Scale: very_low-low-medium-high-very_high
Source: mission statement, job descriptions
Owner: daily planning departments
Threshold: 2 units
Hardness: soft

For these performance indicators the following relations were identified:

(1) causing(PI10, PI1, very_pos)

(2) aggregation_of(PI10, PI12)

The second relation implies that:

(3) causing(PI12, PI10, pos)

Therefore based on relations (1) and (3) and the rules in Table 1 we can conclude that:

(4) causing(PI12, PI1, very_pos)

Such inferred relations are not shown on Figure 2 in order to simplify the picture.

Further let us consider technique (7). The task structure of an organization may provide insight to discover relations between performance indicators. Often refinement relations specified in the task structure correspond to causality relations in the performance indicators structure. For example, based on the refinement relation between the task 'create a correct plan' (related by its process instance to performance indicator 'time for creating a correct plan') and its subtask 'check a plan' (related to performance indicator 'time to check a plan'), the performance indicators 'time to check a plan' and 'time for creating a correct plan' are related by positive causing relation. Refinement relations might also be reflected by other types of relations in the performance indicators structure.

Further, as it follows from the goal definition given in [10], goals and performance indicators form two highly interrelated structures – changes in one structure almost always imply changes in the other structure. Thus, the performance indicators structure and the goal structure may be created simultaneously. Usually, high level goals of a company are of a strategic (long-term) type. Such goals are often made operational by refining them into lower level tactical (short-term) goals. The identified in such a way refinement relation, by analogy with the task refinement, can be reflected in the performance indicators structure by the corresponding relation between performance indicators, on which the considered goals are based. More specifically, if goals are related by refinement relation, then the corresponding performance indicators are related by a causality relation. Furthermore, if the performance indicator expressions for goals related by refinement, contain comparison functions (e.g. >,<) or measures of degrees (such as 'high', 'low'), or goal patterns are specified by functions such as 'increased'/'decreased', then the specific type of causality may be determined (at least if it is positive or negative). For example, in the case study both goal expressions for G3.1: "It is required to maintain high efficiency of the planning process" and for G3.1.2: "It is required to maintain high level of promptness of communication of forward, short-term and daily planning data to all concerned employees" contain the equality relation to the value "high". According to the principles explained above, this corresponds to the positive causality relation between the PIs "efficiency of the planning process" and "level of promptness of communication of forward, short-term and daily planning data to all concerned employees", which is indeed the case in the PI structure.

In general, the refinement and aggregation of goals can be performed based on information about relations in an organization structure, task structure, temporal dependencies and relations between performance indicators.

The identification of conflict relations between goals is of particular importance for the design and evaluation of organizations. To identify such conflicts, the goal patterns and the performance indicators structure can be used. More specifically, by knowing the type of the causality relation between performance indicators and the types of the goal patterns, the presence of a conflict between goals can be determined. For example, the goal 'It is required to maximize the time for checking the proposed plan for accuracy' and the goal 'It is required to minimize the time for producing a correct plan' are in conflict, since the performance indicators 'time for examining the plan for accuracy' and 'time for producing an accurate plan' are related by positive

causality relation and the corresponding goal patterns are based on opposite types: maximize and minimize. If a conflict between high level goals is found, then via the refinement the cause of the conflict can be found at the lowest level of the goal structure. For this the relations between performance indicators and the domain knowledge are exploited.

7 Related Literature on Performance Measurement

The area of performance measurement is an active field of research in management science attracting interest from both academic and practitioner circles. Researchers have been busy identifying and classifying important performance indicators for any company (e.g. [6]) and those relevant for different specific domains e.g. logistics, production, supply chains, etc. (e.g. [1, 3, 7, 13]). Results have also been reported on real life case studies aimed at giving more insight on the relative importance and appropriateness of performance indicators in different situations. Originally only numerical, mostly financial, indicators were considered, however, nowadays it is believed that non-financial and non-numerical indicators such as customer satisfaction, employee motivation, innovation, quality, market share can be very informative as well (e.g. [5]).

Since performance measurement is a central issue for every organization, the organization's model should take it into account. In organization modeling, however, this is currently done implicitly at best. We are only aware of one methodology, GRAI [2], which explicitly models performance indicators however only in the context of decision making and without taking into account the relationships among the performance indicators and between the performance indicators and other notions such as goals.

Letier et al., on the other hand, define in [8] quality variables which can be related to the performance indicators defined in this approach in order to model partial degree of satisfaction of a goal. Based on them objective functions are defined which are used in the formulation of goals. A major difference is that in [8] probabilistic reasoning is used to determine the partial satisfaction of goals which is reflected in the definitions of objective functions and goals.

It should be noted that sometimes measures such as customer satisfaction, profit, production costs, delivery time (typical performance indicators) are visible in other models as well – often in the definition of goals but they always remain implicit and the relationships between them are usually not discussed.

8 Conclusion

This paper presents an approach for modeling performance indicators and the relationships between them which constitutes a part of an expressive general framework for organizational modeling and analysis. The proposed approach is part of the performance-oriented view of the framework which provides formal tools for analyzing organizational and individual performance and relating current performance

to the organizational goals and their satisfaction as well as to tasks and processes of the organization. Due to its expressivity and formal basis the framework can be used in enterprise information systems. It also allows building structures that can be used for complex analysis both within the performance-oriented view and between the performance-oriented view and other views of the general framework. Some possibilities for analysis are mentioned here but will be elaborated and applied on larger case studies elsewhere. Other views and how they are related to each other will also be presented separately.

References

1. P.C. Brewer and T.W. Speh. "Using the balanced scorecard to measure supply chain performance" *Journal of Business Logistics* 21(1), 2000, 75-93.
2. G. Doumeingts, B. Vallespir, and D. Chen. "Decisional Modelling using the GRAI Grid". In : Bernus, P., Mertins, K. and Schmidt, G. (Eds): *Handbook on Architectures of Information Systems*, Springer-Verlag (1998) 313-338
3. F.T.S. Chan. "Performance measurement in a supply chain" *International Journal of Advanced Manufacturing Technology* 21(7), 2003, 534-548.
4. M.S. Fox. "The TOVE project: towards a common-sense model of the enterprise". In Petrie, C.J., Jr.(Ed): *Proceedings of ICIEMT'92*, MIT Press (1992) 310-319
5. C.D. Ittner and D.F. Larcker. "Coming Up Short on Nonfinancial Performance Measurement" *Harvard Business Review*, 81(11), 2003, 88-96.
6. R.S. Kaplan, and D.P. Norton. "The balanced scorecard – measures that drive performance", *Harvard Business Review*, January-February 1992, pp. 71-79.
7. E. Krauth, H. Moonen, V. Popova, and M. Schut. "Performance Measurement and Control in Logistics Service Providing". In: C.-S. Chen, J. Filipe, I. Seruca and J. Cordeiro, editors, *Proceedings of Seventh International Conference on Enterprise Information Systems, ICEIS 2005*, pp. 239-247.
8. E. Letier, and A. van Lamsweerde. "Reasoning about partial goal satisfaction for requirements and design engineering", *Proceedings of 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2004, pp. 53 – 62.
9. V. Popova and A. Sharpanskykh. "Formal analysis of executions based on process-oriented models", Technical Report 071601AI, Vrije Universiteit Amsterdam, http://hdl.handle.net/1871/10545
10. V. Popova and A. Sharpanskykh. "Formal modelling of goals in agent organisations". In V. Dignum, F. Dignum, E. Matson, B.Edmonds, editors, *Proceedings of Agent Organisations: Modelling and Simulation Workshop* during the *20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, 2007, pp. 74-86.
11. V. Popova and J. Treur. "A specification language for organisational performance indicators". In: Ali, M., and Esposito, F., editors, *Proceedings of 18th International Conference IEA/AIE 2005, Lecture Notes on Artificial Intelligence*, vol. 3533, Springer, 2005, pp. 667-677.
12. A. Sharpanskykh, and J.Treur. "Verifying interlevel relations within multi-agent systems". In: Brewka, G., Coradeschi, S., Perini, A., and Traverso, P., editors, *Proceedings of the 17th European Conference on Artificial Intelligence, ECAI'06*, IOS Press, 2006, pp. 290-294.
13. G. Vaidyanathan. "A Framework for Evaluating Third-Party Logistics" *Communications of the ACM*, January 2005 48, 1: 89-94.

Appendix A. A specification of the performance indicators from the case study

Name: **PI1**
Definition: the level of correctness of plans with respect to the contracts of the employees, the laws, the general policy of the company and division
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI2**
Definition: the level of knowledge of employees involved in (forward) planning about the current contracts of the employees, the laws, the general policy of the company and division
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI3**
Definition: the level of up-to-dateness of the software system used in (forward and daily) planning with respect to the contracts of the employees, the laws, the general policy of the company and division
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI4**
Definition: effectiveness of allocation of security officers
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: forward and daily planning departments, unit manager, security officers
threshold: 2 units
hardness: soft

Name: **PI5**
Definition: average correctness of produced plans

type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI6**
Definition: level of correctness of every produced plan
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI7**
Definition: level of correctness of every produced forward plan.
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: forward planning department
threshold: 2 units
hardness: soft

Name: **PI8**
Definition: level of correctness of every produced daily plan.
type: discrete
time frame: day
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: daily planning departments
threshold: 2 units
hardness: soft

Name: **PI9**
Definition: level of correctness of every produced short-term plan.
type: discrete
time frame: week
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: daily planning departments
threshold: 2 units
hardness: soft

Name: **PI10**
Definition: level of correctness of administrative processing of all planning data in the system.
type: discrete
time frame: month

scale: very_low-low-medium-high-very_high
source: job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI11**
Definition: level of correctness of administrative processing of forward planning data in the system.
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: forward planning department
threshold: 2 units
hardness: soft

Name: **PI12**
Definition: level of correctness of administrative processing of short-term and daily planning data in the system.
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: daily planning departments
threshold: 2 units
hardness: soft

Name: **PI13**
Definition: average level of deviation from daily plans in their application.
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: daily planning departments, unit manager, security officers
threshold: 2 units
hardness: soft

Name: **PI14**
Definition: level of deviation from the produced daily plan in its application
type: discrete
time frame: day
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: daily planning departments, unit manager, security officers
threshold: 2 units
hardness: soft

Name: **PI15**
Definition: the number of concerned security officers not informed on time about the produced daily plan
type: discrete

time frame: day
scale: INTEGER
min value: 0
max value: max_officers
unit: employees
source: job descriptions
owner: daily planning departments
threshold: 10
hardness: hard

Name: **PI16**
Definition: promptness of data change forms delivery by security officers
type: discrete
time frame: day
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: security officers
threshold: 2 units
hardness: soft

Name: **PI17**
Definition: promptness of data change forms delivery to the planners
type: discrete
time frame: day
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: unit manager
threshold: 2 units
hardness: soft

Name: **PI18**
Definition: level of correctness of data change forms delivered to the planners
type: discrete
time frame: day
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: unit manager
threshold: 2 units
hardness: soft

Name: **PI19**
Definition: efficiency of planning and allocation of security officers
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI20**
Definition: efficiency of the planning process

type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI21**
Definition: the number of times the planning activities (create and update plans) exceed the allowed durations
type: continuous
time frame: month
scale: REAL
min value: 0
max value: max_time
unit: hour
source: job descriptions
owner: daily planning departments
threshold: 24 hours
hardness: hard

Name: **PI22**
Definition: the time to update a short-term plan given operational data
type: continuous
time frame: month
scale: REAL
min value: 0
max value: max_time_UST
unit: hour
source: job descriptions
owner: daily planning departments
threshold: 24 hours
hardness: hard

Name: **PI23**
Definition: the time to create a daily plan given operational data
type: continuous
time frame: day
scale: REAL
min value: 0
max value: max_time_D
unit: hour
source: job descriptions
owner: daily planning departments
threshold: 12 units
hardness: hard

Name: **PI24**
Definition: level of promptness of communication of forward, short-term and daily planning data to all concerned employees.
type: discrete

time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI25**
Definition: level of promptness of communication of every produced forward plan to all concerned employees.
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: forward planning department
threshold: 2 units
hardness: soft

Name: **PI26**
Definition: level of promptness of communication of every produced short-term plan to all concerned employees.
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: daily planning departments
threshold: 2 units
hardness: soft

Name: **PI27**
Definition: the time to create a short-term plan after all operational data is received
type: continuous
time frame: month
scale: REAL
min value: 0
max value: max_time_CST
unit: hour
source: job descriptions
owner: daily planning departments
threshold: 24 hours
hardness: hard

Name: **PI28**
Definition: the time to create a forward plan after all operational data is received
type: continuous
time frame: month
scale: REAL
min value: 0
max value: max_time_CFP
unit:
source: job descriptions
owner: forward planning department

threshold: 48 units
hardness: hard

Name: **PI29**
Definition: efficiency of allocation of security officers
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: mission statement, job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI30**
Definition: average level of optimality of forward, short-term and daily planning for efficient allocation of security officers
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: forward and daily planning departments
threshold: 2 units
hardness: soft

Name: **PI31**
Definition: average level of optimality of every forward plan for efficient allocation of security officers
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: forward planning department
threshold: 2 units
hardness: soft

Name: **PI32**
Definition: average level of optimality of every short-term plan for efficient allocation of security officers
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions
owner: daily planning departments
threshold: 2 units
hardness: soft

Name: **PI33**
Definition: level of optimality of every daily plan for efficient allocation of security officers
type: discrete
time frame: month
scale: very_low-low-medium-high-very_high
source: job descriptions

owner: daily planning departments
threshold: 2 units
hardness: soft

Chapter 2

Formal Modelling of Goals in Organizations [1]

Abstract. Each organization exists or is created for the achievement of one or more goals. To ensure continued success, the organization should monitor its performance with respect to the formulated goals. In practice the performance of an organization is often evaluated by estimating its performance indicators. In most existing approaches on organization modelling the relation between performance indicators and goals remains implicit. This paper proposes a formal framework for modelling goals based on performance indicators and defines mechanisms for establishing goal satisfaction, which enable evaluation of organizational performance. Methodological and analysis issues related to goals are discussed in the paper. The described framework is a part of a general framework for organization modelling and analysis.

1 Introduction

Organizations exist for achieving certain goals by coordinating the execution of appropriate activities among actors and by handling the complexity of interactions with the environment. Therefore, the viability and success of an organization depend on how effectively the organization manages its internal activities and how well its behaviour fits with the environmental conditions. The behaviour of an organization is usually guided by its strategic and tactical goals that depend on the professional

[1] Part of this chapter appeared as Popova, V., Sharpanskykh, A.: Formal Modelling of Goals in Agent Organizations. In: V. Dignum, F. Dignum, E. Matson, B. Edmonds (eds.), Proceedings of the Workshop on Agent Organizations: Models, and Simulation at IJCAI'07, 74-86 (2007) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

orientation (i.e., domain, types of activities) and specific characteristics of the organization, interests of concerned stakeholders and on the type of the environment (e.g., a market) in which the organization is situated.

The performance of an organization is often evaluated by estimating the values of its qualitative and quantitative performance indicators (e.g., profits, number of clients). Therefore, to ensure the effectiveness of an organization, all the principal performance indicators (PIs) should be reflected in its goals. While in most existing approaches on organization modelling the relation between PIs and goals remains implicit, this paper defines a clear and general mechanism for specifying goals based on PIs. Then, the performance of an organization can be evaluated by estimating the (level of) satisfaction of its goals.

Different types of goals can be identified in organizations. The satisfaction of some of them can be determined in a clear-cut way by evaluating conditions in goal expressions (e.g., "ensure that an order is processed within 24 hours"). Such goals are sometimes called *hard goals*. The satisfaction of other goals is difficult to assess (e.g., "maximize the customer satisfaction"), since they refer to not directly measurable quantities. Such goals are often called *soft goals*. In this paper both hard and soft goals are described and the corresponding mechanisms for establishing the goal satisfaction are specified.

The individuals (agents) assigned to certain positions (roles) in an organization have personal goals based on individual PIs that may comply with, be disjoint or conflict with organizational goals. The performance of individuals can be determined in the same way as the performance of an organization.

Furthermore, the satisfaction of goals often can only be established in a framework, in which goals are related to other concepts (such as tasks, roles and agents). Such a framework for the performance-oriented modelling is considered in this paper with the main focus on goal modelling. An elaborated description of PIs and the related techniques relevant to the framework are given in [16].

The framework for the performance-oriented modelling constitutes a part of a general framework for organization modelling and analysis. In the general framework, organizations are considered from other perspectives (or views) as well. In particular, *the process-oriented view* describes static hierarchies of tasks and resources, flows of control (or workflows), relations between processes and resources and considers the actual execution of organization scenarios. Within *the organization-oriented view* organizational roles, their authority, responsibility and power relations are defined. In *the agent-oriented view* different types of agents with their characteristics and behavior are identified and principles for allocating agents to roles are formulated.

Note that the identified views are related to each other by means of sets of common concepts. For example, the relations between goals and roles are introduced in the performance-oriented view. Further these relations are used in the organization-oriented view to describe mechanisms of goal assignment and delegation, which also use power and authority relations from the organization-oriented view.

In all these views environmental conditions, in which the organization is functioning, are taken into account: they influence the specification of organization concepts and relations between them (e.g., the formulation of goals and the specification of tasks), thus, affecting the structure and behaviour of a particular

organization model. Furthermore, the type of the environment determines a part of the domain knowledge, which is represented by unconditionally valid facts and rules about the environment that directly influence all the activities within the organization. Another part of the domain knowledge is defined by intrinsic properties of the organization itself.

Concepts and relations within every view are formally described using dedicated languages expressive enough to convey structures and processes of organizations of most types. To provide the formal meaning for the concepts and to enable different specific (for a view) and general (across different views) formal types of analysis of organization models (e.g., by simulations and verification), an axiomatic basis is defined that establishes formal relations between concepts within one view and across different views. Furthermore, the formal definition of organizational models and the axiomatic basis enable semantic integration of different ontologies for enterprise modelling, implemented in information systems of organizations aiming at cooperation or integration.

The formal language and the set of axioms specific for *modelling goals* within the *performance-oriented view* are described in this paper. Furthermore, some of the verification techniques specific for performance-oriented organization models are described as well as some methodological issues related to creating and revising goal structures, and the process of organizational performance evaluation based on a goal hierarchy. Other views of the general framework will be considered elsewhere.

The presentation is organised as follows. Section 2 introduces the case study used to illustrate modelling and analysis techniques. In Section 3 the main concepts for the goal modelling framework are specified. The relationships between them are described and formalized using the dedicated logic-based language in Section 4. Section 5 discusses how the performance of the organization is evaluated in the introduced framework. Some design principles are given in Section 6. In Section 7 the related work on goal-oriented modelling is discussed. Finally, Section 8 concludes the paper.

2 Introduction to the Case Study

The proposed approach was applied for modelling and analyzing an organization from the security domain within the project CIM (Cybernetic Incident Management, see http://www.almende.com/cim/). The main purpose of the organization is to deliver security services to different types of customers. The organization has well-defined multi-level structure that comprises several areas divided into locations with predefined (to a varying degree) job descriptions for employees (approx. 230.000 persons). The global management of the organization (e.g., for making strategic decisions) is performed by the board of directors, which includes among others the directors of the different divisions (regions). Within each region a number of areas exist controlled by area managers. An area is divided into several units, supervised by unit managers. Within each unit a number of locations are served, for which the contracts with customers are signed and security officers are allocated. The allocation of employees is performed based on plans created by planning groups.

The examples given in this paper will be related to the part of the organization concerned with the planning of the allocation of security officers to different locations of customers. The planning process consists of the forward (or long-term) planning and the short-term planning. The forward planning is the process of creation, analysis and optimization of forward plans for the allocation of security officers within the organization for a long term (4 weeks) based on customer contracts. It is performed by forward planners from the forward planning group, managed by the manager of planning. During the short-term planning, plans that describe the allocation of security officers to locations within a certain area for a short term (a week) are created and updated based on the forward plan and up-to-date information about the security employees. Based on short term plans, daily plans are created. For each area the short-term planning is performed by the area planning team that consists of planners and is guided by a team leader. During the planning process short-term planners interact actively with forward planners (e.g., for consultations, problem solving). Furthermore, forward planners have a number of supervision functions with respect to short-term planners.

3 Concepts for Goal Modelling

Each organization exists for the achievement of one or more goals. This varies depending on the type of organization and the environmental conditions, in which the organization is situated, e.g. the main goal of a manufacturing company can be the realization of maximal amount of profit, whereas the goal of a non-profit organization for animal protection can be to rescue maximal number of wild animals. Being aware of these goals is a prerequisite to taking measures for their satisfaction. To ensure continued success, the organization should monitor its performance with respect to the formulated goals. To enable the goal-based performance evaluation, organizational goals should be formulated over performance measures (indicators).

Definition 1. (Performance indicator (PI))
A performance indicator is defined as a measure, quantitative or qualitative, that can be used to give a view on the state or progress of the company, a unit within the company or an individual (e.g., time to produce a short-term plan, efficiency of allocation of security officers).

The set of relevant PIs is company-specific. Furthermore, causal and other relationships may exist between different PIs.

Expressions can be formulated over PIs containing >, = or <, for example for defining target values: an expression over the PI P1:“efficiency of allocation of security officers” is defined as `P1 = high`. PI expressions are used to define goal patterns.

Definition 2. (Goal pattern)
A goal pattern is a property over one or more PI expressions that can be checked for a given state/time point or interval for the company or an individual agent.

Goal patterns are characterized by: (1) *name*; (2) *definition*; and (3) *type*.
Type determines the way the property will be checked:

(a) *achieved (ceased)* – it should be checked whether the property is true (false) for a specific time point;
(b) *maintained* (*avoided*) – it should be checked whether the property is true (false) for the duration of a specific time interval;
(c) *optimized* (maximized, minimized, approximated) – it should be checked if the value of the PI expression has increased, decreased or approached a given target value for the duration of a given time interval.

Achieved, *ceased*, *maintained* and *avoided* are used on PI expressions that are evaluated to a Boolean value; *optimised* is defined over PI expressions that are evaluated to value of any type that is ordered (for maximized, minimized) or for which a distance measure is defined (approximated).

Consider the following examples of goals patterns: "maintained efficiency of allocation of security officers to objects = high" based on the *maintained* pattern type and "achieved that time to produce a short-term plan given operational data ≤ 48" based on the *achieved* pattern type.

Goals are formulated by adding to goal patterns information such as desirability and priority.

Definition 3. (Goal)
Goal is an objective to be satisfied describing a desired state or development of the company or an individual.

For example "it is required to maintain high efficiency of allocation of security officers". A goal is characterized by: (1) *name*; (2) *definition*, (3) *priority*; (4) *evaluation type*; (5) *horizon*; (6) *ownership*; (7) *perspective*; (8) *hardness*; and (9) *negotiability*.

Priority is defined by a numerical estimation between 0 and 1; alternatively {very high, high, medium, low, very low}. When less information about goal priorities is available, a (partial) ordering on goals may be defined.

Evaluation type determines if a goal is based on goal pattern with type *achieved* or *ceased* (*achievement goal*), i.e., it is evaluated for a given state/time point, or if a goal is based on goal pattern of type *maintained*, *avoided* or *optimized* (*development goal*), i.e., it is evaluated for a given time interval.

Horizon specifies within which time interval (for development goals) or at which time point (for achievement goals) is the goal supposed to be satisfied: (a) *long-term goal*; (b) *mid long-term goal*; (c) *short-term goal*.

Ownership can be organizational, i.e., belongs to an organization/unit/role, follows from the highest level goals of the company, and individual, i.e., belongs to an agent. Normally, organizational goals have a high level of priority. Goals of agents may comply with organizational goals to a varying degree. The priority of individual goals might depend on the company policy: some companies might assign lower priority to individual goals than to organizational ones; others might decide to involve and motivate the agents by taking into account their goals and avoiding some conflicts that might exist between individual and organizational goals.

Perspective (for organizational goals) defines, which point of view is described by the goal: of *management*; of a *supplier*; of a *customer*; or of the *society*. Even though all organizational goals belong to the organization itself, they can reflect the point of view of an external party which desires the organization to perform in a certain way.

For example the society wants the organization to obey society's norms and values. It is sometimes beneficial for the company to adopt goals desired by other parties e.g. to conform to the relevant laws.

It is also important to note that the different points of view will often be conflicting, for example while customers might want low prices, the management wants high profits, however if the prices are lowered that will decrease the profits. Such conflicts should be recognised during the design phase and made explicit in order to deal with them. For example priorities can be defined in order to specify which goal is more important to satisfy.

Hardness distinguishes *soft* and *hard* goals. The satisfaction of a *soft goal* cannot be clearly established. We use the term *satisficing* to indicate an acceptable degree of satisfaction of a soft goal. Soft goals are given labels that correspond to their degrees of satisficing/denial with a natural order between the labels: *satisficed* > *weakly_satisficed* > *undetermined* > *weakly_denied* > *denied*. Satisfaction of hard goal can be established quantitatively. Hard goals also have labels ordered as follows: *satisfied* > *undetermined* > *failed*. In the example below goal G3.2 (this and the following goals are named by labels from the goal tree constructed for the considered case study) is soft, PI "efficiency of allocation of security officers" cannot be objectively established to be maintained high or not, instead we use a subjective estimation of degree of satisficing. Goal G3.1.1.1 is hard – it can be seen if PI "time to update a short-term plan given operational data" is at most 48 hours.

By *negotiability* goals are divided into *non-negotiable* (i.e., need to be satisfied, no compromise is possible) and *negotiable* (negotiation is possible in case of conflicts with other goals). This can be used for conflict resolution at the design phase.

Examples:

Goal name: G3.2
Informal definition: It is required to maintain high efficiency of allocation of security officers to objects
Priority: high
Horizon: long-term
Evaluation type: development goal (maintain goal pattern)
Ownership: organizational
Perspective: management, customer
Hardness: soft
Negotiability: negotiable

Goal name: G3.1.1.1
Informal definition: It is required to achieve that the time to update a short-term plan given operational data is at most 48 hours.
Priority: high
Horizon: short-term
Evaluation type: achievement goal (achieve goal pattern)
Ownership: organizational
Perspective: management
Hardness: hard
Negotiability: negotiable

Goals are realizable by tasks in an organization. *A task* represents a function performed in an organization by its role(s). A role is characterized by a set of functionalities performed by it. Roles are characterized by sets of competences, which are required to perform a certain task. Competences can be credentials (i.e., material or digital objects certifying accomplishments; e.g., diplomas, patents, certificates), and skills (i.e., abilities that can be demonstrated and/or tested; e.g., typing speed, flexibility, programming skills). Roles are allocated to agents to perform tasks in an organization. Roles and agents are committed to certain goals. An agent can only play a particular role if it has the capabilities that match the competences required in the role description. In addition to organizational goals, an agent may pursue its own individual goals that comply or conflict with organizational goals. These and other concepts will only briefly be discussed in this paper and will be extensively considered in the descriptions of other views.

4 Formal Goal Modelling

In this Section first the concepts and relations introduced previously in Section 3 will be formalized using the first-order sorted predicate language (Section 4.1). After that, goal structures that comprise both soft and hard goals will be introduced in Section 4.2. In particular, this Section includes the formal description of relations between goals and goal satisfaction principles in goal structures.

4.1 Formalizing concepts for goal modelling

The formal language used for specifying the meta-model for the performance-oriented view is the first order sorted predicate language [14]. In this language, for each type of a concept a special sort is introduced, which contains all the names of concept instances (e.g., sort GOAL contains all the names of goals). The semantics for this language is defined in a standard way, by interpretation of sorts, constants, functions and predicates, and a variable assignment. The characteristics (or attributes) of the concepts are represented by corresponding relations (predicates) with arguments: a concept name, an attribute name and a value for the attribute (e.g., has_attribute_value: GOAL x ATTRIBUTE x VALUE). In the following for better readability such predicates will be used in the more compact form: concept.attribute=value. Using this dedicated language a number of relations between goals and other concepts are defined that are included into the meta-model for the performance-oriented view (the graphical representation of the meta-model is given in Figure 1). To provide the formal meaning for the introduced relations and to enable formal verification of organization models (e.g., consistency and integrity checking), the set of axioms is defined along the definitions or relations.

Goals are constructed based on PIs using the relations introduced below.

is_based_on: GOAL_PATTERN × PI: The goal pattern in the first argument is defined over the PI in the second argument.

uses: GOAL_PATTERN × PI_EXPRESSION: Goal pattern defined over PI expression.

In goal patterns the symbols <, >, and = from PI expressions are interpreted as functions: PI x {NUM_VALUE, QUALIT_VALUE} → PI_EXPRESSION, where NUM_VALUE is a sort containing all numerical values, and QUALIT_VALUE contains all qualitative values.

For example, the goal pattern GP1 "maintained efficiency of allocation of security officers to objects = high" is based on the PI P2 "efficiency of allocation of security officers to objects" and uses the PI expression PE1 formulated over P2 (P2=high): is_based_on(GP1,P2); uses(GP1,PE1).

is_formulated_over: GOAL × GOAL_PATTERN: The goal in the first argument is defined over the goal pattern in the second argument.

For example, the goal G3.2 defined earlier is formulated over the goal pattern GP1.

Goals are related to tasks, roles and agents by the following relations:

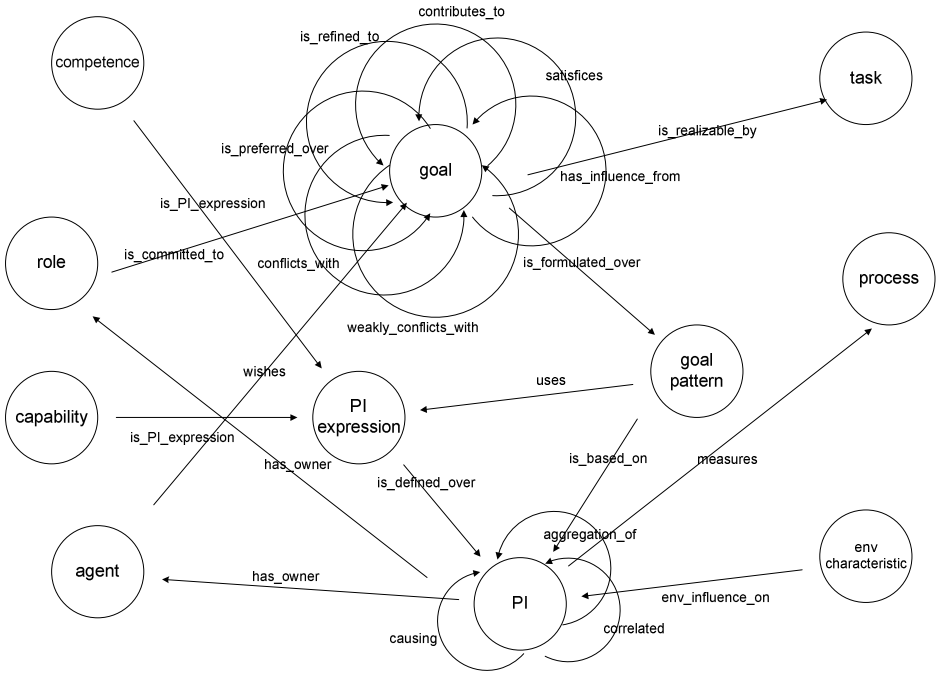


Fig. 1. A Meta-model for the performance-oriented view

is_realizable_by: GOAL x TASK_LIST: The goal in the first argument is realizable by the list of tasks in the second argument.

is_committed_to: ROLE × GOAL: The goal is an organizational goal and the role is committed to the satisfaction of this goal.

wishes: AGENT × GOAL: The goal is an individual goal of the agent.

For example, role Planner is committed to goal G3.1.1.1, which is realizable by task T4.4.1 "update short-term plan", i.e., is_committed_to(Planner, G3.1.1.1) &

is_realizable_by(G3.1.1.1, L41) & is_in_task_list(L41, T4.4.1), where is_in_task_list: TASK_LIST x TASK.

4.2 Modelling of Goal Structures

A goal structure can be built by refining high level goals (top-down approach) and aggregating lower lever goals into higher level goals (bottom-up approach). Since goals in the modelling framework can be of two types: hard and soft, different types of refinement relations should be considered.

First consider refinement of hard goals. Hard goals are refined into *and-lists* of hard goals (sort AND_GOAL_LIST), in which the goals are connected by AND relation.

is_refined_to: GOAL × AND_GOAL_LIST: Defines a refinement of a hard goal into a list of hard goals, which contribute to its satisfaction. The refinement means that when all the goals in the list are satisfied then the goal in the first argument will be satisfied as well. If one or more goals in the list fail and no other refinement exists where all goals are satisfied, then the goal in the first argument will fail too. More formally, we introduce the predicates satisfied: GOAL and failed: GOAL to express the satisfaction state of a goal and these predicates can then be used to formulate the following axioms:

∀ l: AND_GOAL_LIST is_refined_to(g, l) & (∀gi:GOAL is_in_ goal_list(gi,l) ⇒ satisfied(gi))
⇒ satisfied(g)

∀ l: AND_GOAL_LIST (is_refined_to(g, l) ⇒ ∃gi: GOAL is_in_ goal_list(gi, l) & failed(gi))
⇒ failed(g)

where ***is_in_goal_list***: GOAL × GOAL_LIST expresses that a goal is in a goal list. Sort AND_GOAL_LIST is a subsort of GOAL_LIST, which contains names of all goal lists.

is_subgoal_of: GOAL × GOAL: The first argument is a goal which is a subgoal of the goal in the second argument, i.e., it takes part in a refinement list of the second goal.

The relation between is_in_goal_list and is_subgoal_of is established by the following axiom expressing that if the goal G2 is refined into the list L, and G1 is one of the goals in the list L, then G1 is a subgoal of G2:

∀ G1, G2: GOAL, ∀ L: GOAL_LIST: is_in_goal_list(G1, L) & is_refined_to(G2, L)
⇒ is_subgoal_of(G1, G2)

When more than one refinements are defined, they are considered as alternatives connected by OR, i.e., they allow a choice, which measures to take to satisfy the goal.

The refinement of hard and soft goals will be illustrated in the context of the goal structure given in Figure 2. This structure is constructed from the most important and relevant goals related to the planning process of the company considered in the case study. The detailed description for the goals of this structure is given in Appendix A.

Example:

In Figure 2 the hard goal G3.1.1 “It is required to achieve that the number of times the planning activities (creating and updating of a plan) exceed the allowed durations is equal to 0” is refined into the and-list that consists of goals G3.1.1.1 “It is required to achieve that the time to update a short-term plan given operational data is at most 48 hours”, G3.1.1.2 “It is required to achieve that the time to create a daily plan given

operational data is at most 24 hours", G3.1.1.3 "It is required to achieve that the time to create a short-term plan after all operational data is received is at most a week", and G3.1.1.4 "It is required to achieve that the time to create a forward plan after all operational data is received is at most a week". This refinement is formally defined by the following relations:

is_in_and_goal_list(G3.1.1.1, L)
is_in_and_goal_list(G3.1.1.2, L)
is_in_and_goal_list(G3.1.1.3, L)
is_in_and_goal_list(G3.1.1.4, L)
is_refined_to(G3.1.1, L)
is_subgoal_of(G3.1.1.1, G3.1.1)
is_subgoal_of(G3.1.1.2, G3.1.1)

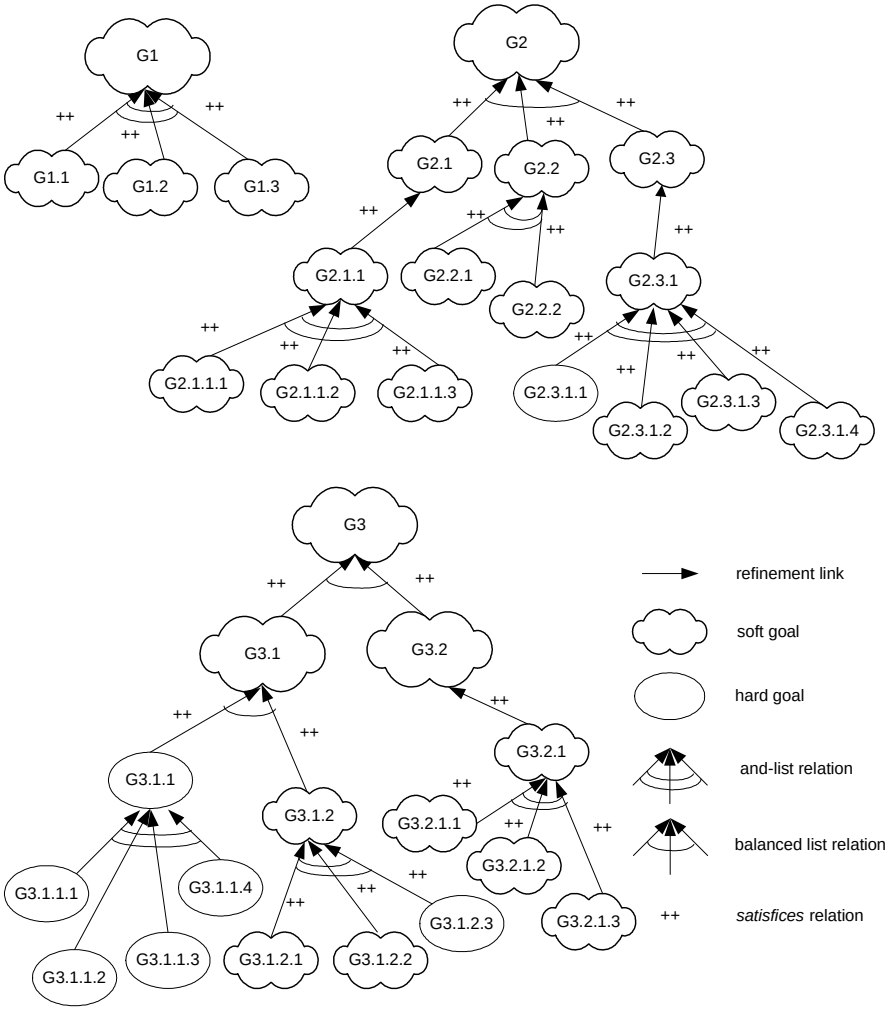


Fig. 2. A partial goal structure for the considered case study (for a detailed description of goals see Appendix A)

Goal and PIs structures are closely related to each other. In particular, if goals are related by the refinement relation, then the corresponding PIs are related by a causality relation. This is expressed by the following axiom, where EFFECT = {very_negative, negative, positive, very_positive}:

∀ G1, G2: GOAL, ∀ L: GOAL_LIST ∀ GP1, GP2: GOAL_PATTERN ∀PI1, PI2: PI:
is_in_goal_list(G1, L) & is_refined_to(G2, L) & is_based_on(GP1, PI1) & is_formulated_over(G1, GP1) &
is_based_on(GP2, PI2) & is_formulated_over(G2, GP2) ⇒ ∃pn: EFFECT causing(PI1, PI2, pn).

Now let us consider the refinement of soft goals. Since the satisfaction of soft goals cannot be established in a clear-cut way, the process of refinement of soft goals also differs from the refinement of hard goals. It is more difficult to clearly define decomposition for soft goals. Instead we talk about positive contribution from other goals in the satisfaction of the goal to be refined. Such contribution can vary in its degree (i.e. strength) which is expressed by the following relations, in which the goal in the second argument is soft and the goal in the first argument can be soft or hard:

satisfices: GOAL × GOAL: The first goal strongly contributes in a positive way to the satisficing of the second goal. If the first goal is satisfi(c)ed and any other influences are ignored then the second goal is considered satisficed.

contributes_to: GOAL × GOAL: The first goal contributes positively to the satisficing of the second goal, however might not be enough to satisfice it.

The precise meaning of these relations is defined through the propagation rules defined for goals related by refinement. These rules are used to determine the degree of satisfaction/satisficing of a higher level goal (specified by a label) based on the available information about the degrees of satisfaction/satisficing of lower level goals in its refinement. To determine the label of a higher level goal, first the propagated labels from lower level goals of the refinement list are determined using Table 1. Then, the propagated labels are combined depending on the type of the refinement list to determine the label of the higher level goal.

Table 1. The table for determining the propagated labels for a higher level goal based on the satisfaction/satisficing labels of lower level contributing goals and types of contributing links.

Label of contributing goal \ Type of link	**satisfices**	**contributes_to**
satisficed / satisfied	satisficed	weakly_satisficed
weakly_satisficed	weakly_satisficed	undetermined
undetermined	undetermined	undetermined
weakly_denied	weakly_denied	undetermined
denied / failed	denied	weakly_denied

Lower level goals can be combined using *and-* and *balanced contribution* relations in lists which contribute positively to the satisficing of the higher level soft goal.

has_influence_from: GOAL × GOAL_LIST: The goals in the list contribute positively to the satisficing of the soft goal in the first argument. For each goal in the list it is defined separately what the level is of its contribution (the type of the link) using the above defined relations satisfices and contributes_to.

The combination of goals in an and-list implies that if all goals in the list are satisfi(c)ed then the higher level goal will also be satisficed. In order to ensure this the following constraint is enforced: at least one of the goals in an and-list is connected with a link of the type satisfices to the higher level goal. When lower level goals are combined in an and-list, the label of a higher level goal is defined by the minimal label propagated from the goals in this list using the defined order between the labels.

Example
Consider the refinement of the goal G1 "It is required to maintain that the level of correctness of plans with respect to the contracts of the employees , CAO, de Arbeidstijdenwet (Dutch labor legislation), the general company policy /the policy of the business unit Security is very high" in the goal structure (Figure 2). G1 is refined into the and-list L that consists of three soft goals G1.1, G1.2 and G1.3. All goals in the list L are connected to G1 through a satisfices-link. It means that all subgoals of G1 are equally important for the satisfaction of G1. Note that all contribution relations of soft goals in the goal structure in Figure 2 are of type satisfices. This is because for this case study only the most essential goals, satisfaction of which considerably influences the productivity of the organization, have been chosen. The refinement of G1 is formalized by the following relations:

is_in_and_goal_list(G1.1, L)
is_in_and_goal_list(G1.2, L)
is_in_and_goal_list(G1.3, L)
has_influence_from(G1, L)
satisfices(G1.1, G1)
satisfices(G1.2, G1)
satisfices(G1.3, G1)

Furthermore, let us have the levels of satisfycing of G1.1, G1.2 and G1.3 based on measurement and observation. They are assessed to be as follows: G1.1 is satisficed, G1.2 is weakly satisficed, G1.3 is weakly denied. We can now propagate this knowledge taking into account the type of the links using the Table 1 in order to find out the level of satisficing of G1. The propagation for G1.1 results in the label satisfied, for G1.2 – the label weakly_satisfied, and for G1.3 – the label weakly_denied. Taking the minimal label we conclude that goal list L propagates weakly_denied.

Another kind of relation between goals represents balanced contribution which gives us the possibility to describe more fine-tuned ways of contributing which favour the majority influence. The rule that is used to calculate the exact effect first quantifies the propagated labels of lower level goals and then takes the (weighted) average which is subsequently discretized again to the closest label, resulting in the sought label for the higher level soft goal. The quantification scale for the propagated labels may look as follows: *satisficed* = 2, *weakly_satisficed* = 1, *undetermined* = 0, *weakly_denied* = -1, *denied* = -2. Then, to fine-tune influences that the lower level goals from the balanced list (and thus, the propagated labels) have on the determination of the label for the higher level goal, weights can be assigned for the lower level goals in the list. Let the quantified propagated labels from the goals in the balanced list be g_i and the weights defined for each goal in the list are w_i. Then the influence of the balanced list on the higher level goal is calculated using a formula of the type: $\Sigma_i w_i g_i / \Sigma_i w_i$.

To specify the weight of a goal in a balanced list the relation ***has_weight_in_list***: GOAL × INTEGER × BAL_GOAL_LIST is defined.

When a goal is refined in one list only then the influence calculated using the described above rules defines the satisficing label of the goal. Sometimes a goal is refined into alternative influence lists related by OR. This reflects the knowledge that these lists are in conflict or competition and if one is satisficed then the probability that the rest will also be satisficed is lower. In such situations we use the following strategy: first the influences of the and- and balanced lists are calculated separately and then the highest among them label is assigned to the higher level goal.

Example

Consider the refinement of the goal G2 "It is required to maintain high effectiveness of allocation of security officers" (see Figure 2). G2 is refined into the balanced list L of three soft goals G2.1 "It is required to maintain high average correctness of produced plans", G2.2 "It is required to maintain high level of correctness of administrative processing of all planning data in the system", and G2.3 "It is required to maintain low average level of deviation from daily plans in their application". All goals in the list L are connected to G2 through a satisfices link. Furthermore, weights for goals in L are defined: 2 for both G2.1 and G2.3, and 1 for G2.2. Formally:

is_in_goal_list(G2.1, L)
is_in_goal_list(G2.2, L)
is_in_goal_list(G2.3, L)
satisfices(G2.1, G2)
satisfices(G2.2, G2)
satisfices(G2.3, G2)
has_influence_from(G2, L)
has_weight_in_list(G2.1, 2, L)
has_weight_in_list(G2.2, 1, L)
has_weight_in_list(G2.3, 2, L)

Let us assume that the degrees of satisficing of the lower-level goals G2.1, G2.2 and G2.3 are known. Let G2.1 be satisficed and G2.2 and G2.3 be weakly_satisficed. Then, using the Table 1 the propagated labels are obtained: satisficed for G2.1 and weakly_satisficed for both G2.2 and G2.3. The labels are quantified so that the degree of satisficing of G2.1 is considered 2 and the degree of satisficing of G2.2 and G2.3 is considered 1.

Then, the degree of satisficing of G2 is calculated as (2*2+1+1*2)/5 = 1.4 which we round up to 1 (which corresponds to weakly_satisficed in our scale).

Apart from the refinement links discussed so far, we can also define conflicts, which represent negative relations between (hard or soft) goals or lists of goals.

conflicts_with: AND_GOAL_LIST × AND_GOAL_LIST: Represents joint negative effect between lists of goals, i.e., the goals in both lists cannot be satisfi(c)ed or weakly satisficed at the same time. More precisely, if all goals in one list are satisfi(c)ed then at least one goal in the other is failed or denied; if all goals in one list are at least weakly satisficed, at least one goal in the other is at most weakly denied.

weakly_conflicts_with: AND_GOAL_LIST × AND_GOAL_LIST: Represents weak joint negative effect between lists of goals, i.e., the goals in both lists cannot be satisfi(c)ed at the same time. More precisely, if all goals in one list are satisfi(c)ed then at least

one goal in the other is at most weakly denied; if all goals in one list are at least weakly satisficed then at least one goal in the other is at most weakly satisficed.

Conflicts can be defined at each two levels of the goal structure, however if the hierarchy is sufficiently complete then these conflicts should be propagated through the goal refinement to the lowest level at which the sources of these conflict can be found. Conflicts can also be used at the analysis and evaluation phases by propagating satisfaction labels bottom-up when only partial information is available. For example let goals g1 and g2 be in conflict at the lowest level of the goals structure and let g1 be known to be satisfied. Then if the satisfaction label of g2 is not known it can be assumed to be at most weakly denied if g2 is soft and failed if g2 is a hard goal. If however it is known that g2 is satisfi(c)ed then that points at an inconsistency in the model.

In the goal structure given in Figure 2 no conflicts between goals are present. However, in the past the considered in the case study organization contained conflicts between its general company goals and low level goals of its subdivisions, which caused the inefficient operation of the company. For example, the high-level goal of the company "It is required to maintain high effectiveness of allocation of security officers within the organization" was in conflict with the goal "It is required to maintain high level of independency (autonomy) of all area planning teams from each other". The conflict was identified in the refinement of these goals: it is often the case that in order to create an effective plan in one area, data about the available and scheduled employees from other areas were needed. Without cooperation the autonomous area planning teams often created inefficient plans, which suffered from the shortage of information. For example, plans for a certain area were created on the basis of the shortage of security officers in this area, whereas in other areas available (not scheduled) employees were in plenty.

5 Goal-based Performance evaluation

The explicit identification of PIs in the structure of goal expressions and the satisfaction propagation mechanisms in goal hierarchies described in the previous Section 4 provide means for the evaluation of organizational performance, which are described in this Section.

Consider the process of goal-based performance evaluation in detail. Every task performed in an organization contributes to the satisfaction of a certain organizational goal(s). Each goal is formed based on a PI(s). This PI(s) can be measured (directly or indirectly) during or after the task execution depending on the goal evaluation type, in the end or during a certain period of time (an evaluation period defined as a goal horizon). Then, by comparing the measured value(s) with the goal expression(s), the satisfaction (degree of satisficing) of the goal(s) is determined. Further, the obtained goal satisfaction (satisficing) measure is propagated by applying the rules defined in Section 4, upwards in the goal hierarchy for determining the satisfaction (degree of satisficing) of higher level goals. Thus, the organizational performance is evaluated by determining the satisfaction (degree of satisficing) of key organizational goals. The same principles can be applied for evaluation of agent performance.

As illustration of the proposed performance evaluation procedure consider the evaluation of satisfaction of one of the most important organizational goals from the case study – goal G3.1 “It is required to maintain high efficiency of the planning process”. Figure 2 shows the refinement of G3.1 into a balanced list of more specific goals: the hard goal G3.1.1 and the soft goal G3.1.2. Goal G3.1.1 has the weight 3 in the list and G3.1.2 has the weight 2. Furthermore, the goal G3.1.1 is refined into an and-list that consists of four hard goals and G3.1.2 is refined into an and-list of two soft goals and one hard goal. The lowest level goals (goals corresponding to the leaves) of this structure are related to tasks from the task structure created for the organization from the case study. Since task modelling is out of scope of this paper, only names of some tasks from the task structure will be used here. The lowest level goals, their PIs and the corresponding tasks for the considered example are given in Table 2.

Table 2. The relation between goals and tasks, and the level of satisfaction of goals from the case study

Goal name	Goal expression	PI	Related task	Level of goal satisfaction/satisficing
G3.1.1.1 (hard)	It is required to achieve that the time to update a short-term plan given operational data is at most 48 hours	The time to update a short-term plan given operational data	update_shortterm_plan	satisfied
G 3.1.1.2 (hard)	It is required to achieve that the time to create a daily plan given operational data is at most 24 hours	The time to create a daily plan given operational data	create_daily_plan	satisfied
G 3.1.1.3 (hard)	It is required to achieve that the time to create a short-term plan after all operational data is received is at most a week	The time to create a short-term plan after all operational data is received	create_shortterm_plan	satisfied
G 3.1.1.4 (hard)	It is required to achieve that the time to create a forward plan after all operational data is received is at most a week	The time to create a forward plan after all operational data is received	create_forward_plan	satisfied
G 3.1.2.1 (soft)	It is required to achieve high level of promptness of communication of every produced forward plan to all concerned employees	The level of promptness of communication of every produced forward plan to all concerned employees	inform_all_concerned_about_forward_plan	weakly_satisficed
G 3.1.2.2 (soft)	It is required to achieve high level of	The level of promptness of	inform_all_concerned_a	weakly_satisficed

	promptness of communication of every produced short-term plan to all concerned employees	communication of every produced short-term plan to all concerned employees	bout_ shortterm_p lan	
G 3.1.2.3 (hard)	It is required to achieve that the number of concerned security officers not informed on time about the produced daily plan is zero	The number of concerned security officers not informed on time about the produced daily plan	inform_all_ concerned_a bout_ daily_plan	satisfied

Furthermore, for each goal in this table the level of satisfaction is identified. This is done by measuring the values of the PIs, on which these goals are based, during or after the task execution. For most of the hard goals these data can be extracted from the log-files and databases of the enterprise management system, and thus the satisfaction of the goals can be directly determined. On the contrary, the level of satisficing of soft goals is often difficult to identify, since they are based on soft PIs. In order to evaluate a soft PI it is usually beneficial to find a closely related hard indicator that can be measured instead and that can give an impression on the state of the soft one. For example, to estimate the soft PI "The level of promptness of communication of every produced forward plan to all concerned employees", on which the goal G 3.1.3.1 is based, for every forward plan the time interval between the moment when the plan is created and the moment when it is communicated to all concerned employees is measured. In 10-15% of all cases, this interval was longer than it is allowed by company's regulations. By the choice of the designer, this corresponds to the weakly_satisficed label for the goal G 3.1.3.1.

By applying the propagation rules, the label for the goal G3.1.1 is determined as satisfied, and for the goal G3.1.2 – as weakly_satisficed. For the calculation of the propagated label for the goal G3.1 resulted from the balanced list, the following quantification scale is used: *satisficed/satisfied* = 3, *weakly_satisficed* = 1, *undetermined* = 0, *weakly_denied* = -1, *denied/failed* = -3. Then, the degree of satisficing of G3.1 is calculated as (3*3 + 1*2)/5=2.2, which corresponds more closely to the label *satisficed*, which gives a strong positive evaluation of the overall organizational performance.

6 Methodological issues in the Design of Goal Structures

In this Section some techniques for building a consistent goal hierarchy are described. Since in the proposed framework goals are based on PIs and are related to other concepts (e.g., roles and tasks), many of the methodological issues with respect to goals will be considered in relation to these concepts.

Usually, high level goals of a company are of a strategic (long-term) type. Such goals are often made operational by refining them into lower level tactical (short-term) goals. In such a way a goal-structure is created by a top-down design process. The refinement of goals may proceed until subgoals are found, which could be

realized by (possibly single) lowest-level tasks from the task hierarchy. In practice, the top-down design approach is often combined with the bottom-up approach, which is performed by aggregation of goals. For example, in the goal elicitation approach described in [4] subgoals are identified by asking "how" questions about the goals already determined, and parent goals are identified by asking "why" questions.

To fine-tune goal and task structures, and relations between them at the design phase, backwards reasoning approaches on a goal structure can be used. These approaches are particularly useful for the analysis of cases of a soft goal refinement. More specifically, given that a higher level soft goal is required to be satisfied to a certain degree, and provided the type of a list into which this goal is refined (i.e., and or balanced) and types of refinement links between goals, it is possible to determine the least degree of satisfaction of the lower level goals from the refinement list. This information constitutes constrains on lower-level goals that can be used for the revision or (re)formulation of goals and corresponding tasks, and relations between them.

Furthermore, relations between goals can be identified by using relations in the corresponding PIs structures [16]. In PIs structures different types of relationships between PIs are identified: e.g., causality, correlation and aggregation. A refinement relation between goals often corresponds to a causality relation between the corresponding PIs on which these goals are based. For example, in the goal structure in Figure 2 goals G2.3.1 and G2.3.1.4 are related by refinement, whereas the corresponding PIs "the level of deviation from the produced daily plan in its application" (for G2.3.1) and "the level of correctness of data change forms delivered to the planners" (for G2.3.1.4) are related by a negative causality relation in the PI structure developed for this case study. This means that the higher is the level of correctness of data change forms delivered to the planners, the less is the level of deviation from the produced daily plan in its application. Furthermore, a goal refinement relation may also correspond to an aggregation relation in a PI structure. For example, goals G2.1 and G2.1.1 are related by refinement and their corresponding PIs are related by aggregation. Note that an aggregation relation also assumes a positive correlation.

Since goal and PI structures are closely related, it is important to guarantee consistency and correspondence of these structures to each other. For this a dedicated consistency check can be performed, based on the following principle. If goals are related by the refinement relation, then the PIs corresponding to these goals are related by a certain (positive or negative) causality relation. To determine the exact type of causality, goal expressions should be analyzed. If the PI expressions for goals related by refinement, contain an equality relation ("=") over comparable (or opposite) measures of degrees (i.e., high/ low, maximal/ minimal) of some variables, then the corresponding PIs are most probably related by positive (or negative) causality relation. Comparison functions (i.e., '>','<') or change functions (i.e., 'increased', 'decreased') in PI expressions can be treated in a similar way. Furthermore, if a precise (mathematical) functional relation between PIs, on which goals in a refinement are based, is known, then the type of the causality relation can be easily determined and used for identifying inconsistencies in the goal structure. Note that since the designer has much of freedom in specifying goal expressions, there is no guarantee that inconsistencies identified in a PI structure are valid. Therefore, all

automatically identified inconsistencies in goal and PI structures still need to be confirmed by the designer.

For example, in the case study both goal expressions for G3.1 and for G3.1.2 contain the equality relation to “high”. According to the principles explained above, this corresponds to the positive causality relation between the PIs “efficiency of the planning process” and “level of promptness of communication of forward and short-term planning data to all concerned employees”, which indeed the case in the PI structure.

The identification of conflict relations between goals is of particular importance for the design and the evaluation of organizations. In order to create an effective organization, it is often advised at the early design phase to take into consideration interests and concerns (expressed as goals) of different stakeholders, who will eventually play a role within the organization and will interact with the organization. The stakeholders may have conflicting goals that should be reflected in an organization model being constructed. Furthermore, conflicts may exist in a goal set of a stakeholder. To identify conflicts between goals, the goal patterns and the PIs structure can be used: by knowing the type of a causality relation between PIs and the types of goal patterns, the presence of a conflict between goals can be determined. The goal structure created for the case study does not contain conflicts; therefore, we shall illustrate the principle of conflict identification by assuming hypothetical goals for the company from the case study: The goal “It is required to maximize the time spent on examining the plan proposal for correctness” and the goal “It is required to minimize the time spent on producing a correct plan” are in conflict, since the corresponding PIs “the time spent on examining the plan proposal for correctness” and “the time spent on producing a correct plan” are related by the positive causality relation, and the corresponding goal patterns are based on the opposite types of functions: maximize and minimize.

If during the design phase a conflict between high level goals is determined, then through the refinement a more precise cause of the conflict can be found at the lowest level of a goal structure. For this the relations between performance indicators and the available domain knowledge are exploited.

For those organization models that do not allow conflicts, the consistency of a model can be achieved by applying different conflict resolution techniques [11]. The common strategy for conflict resolution is based on weakening of goal expressions (e.g., by weakening boundary conditions in the PI expressions; by introducing so-called ‘organizational slacks’). For example, a Planner may have an individual goal to minimize his/her overwork, which is in conflict with the organizational goal G3.1.1.2 “it is required to achieve that the time to create a daily plan given operational data is at most 24 hours”. To be able to create a daily plan within 24 working hours, planners often need to work overtime (due to some other daily occupations), which contradicts his/her own goal. If the company recognizes the importance (i.e., gives a high level of priority) of the individual goal of the Planner, the organizational goal G3.1.1.2 could be weakened by allowing 28 hours for the accomplishment of a daily plan.

This example shows the importance of the goal priority attribute for the process of conflict resolution. For example, it can be used to determine which goal can be modified to a greater degree or even deleted from a model. In general organization goals have the higher priority than individual goals of agents. Therefore, in order to fit

into the organization, an agent sometimes needs to adjust her/his own goals to the organizational ones. On the other hand, sometimes priorities of goals of an agent (e.g., important customer, government) can be so high that the organization decides to revise its goal structure to ensure the satisfiability of agent goals.

For negotiable goals conflicts can be solved by negotiations among the stakeholders, to whom the goals are related [18].

7 Related Work on Goal-Oriented Modelling

Goal-oriented modelling is given a special place in the area of enterprise engineering. Often both organizational and individual goals of the involved actors are considered and distinctions are made between the goals originating from different stakeholders.

Some aspects of our definition of a goal are inspired and come close to existing state-of-the-art approaches in enterprise modelling and requirement engineering [9, 10]. There are however significant differences as well which will be pointed out here. Our analysis pinpointed the following approaches as most relevant – CIMOSA [2], TOVE [5, 6], *i** [19], Tropos [1, 7], goal-oriented agent-based models [3, 8, 12], KAOS [4], the NFR framework [15] ordered roughly in an increasing degree of relevance to this discussion.

In CIMOSA the notion of objectives is used to represent business goals for a particular domain (i.e. a part of the enterprise). No relationships between the objectives are defined therefore no hierarchy of objectives is built. No distinction is made between hard and soft goals. Also in the TOVE model no distinction is made between hard and soft goals. Goals can be decomposed in AND/OR subgoal trees.

The *i** approach focuses on the dependency relationships between the actors. A Strategic Rationale model is built on the level of each actor where its internal reasoning on the relationships between goals, tasks and resources can be modelled. The approach recognises both hard and soft goals and defines a (soft)goal dependency relationships between actors w.r.t. (soft)goals expressing that one actor depends on another to make a condition in the world come true. The goals are only informally specified; no format and unified representation is enforced. The goals hierarchy is coupled to the tasks hierarchy as tasks can be decomposed to goals and tasks. Positive and negative contribution to a different degree of tasks/goals to soft goals are modelled using contribution links. Tropos is a methodology for agent-oriented software development based on *i** thus goals are treated in the similar way as in *i**. The extension Formal Tropos [7] uses a temporal specification language inspired by KAOS.

The agent-oriented enterprise meta-model presented in [8] defines a goal as a desired or undesired state of the environment which is described by states of objects (beliefs, authorisations, resources, etc.). Goals can be refined into alternative sets of other goals using AND/OR relationships. Distinction is made between operational and soft goals – plans can fulfil operational goals but can only contribute positively or negatively to soft goals. Goals can also be organizational or personal. A dependency relationship between organizational roles for the fulfilment of organizational goals is defined.

Moreover, the motivational concept of a goal has been often used for specifying attitudes of agents in multi-agent systems [3, 12]. Usually agent goals are specified as declarative concepts in (modal) logical specifications that describe states of the agent system, which are desirable and could be realized (achieved) by the agent. Often goals are related to other motivational attitudes of agents, such as beliefs, desires and intentions [17]. Moreover, declarative goals are often operationalized in agent programming languages by sequences of actions or plans [3, 12]. Then, the distinction between goals and tasks, essential for the framework proposed in this paper, is not tangible any more.

The KAOS methodology focuses on requirements elaboration and provides support in connecting high-level goals to operations, objects and constraints to be implemented by the software. A goal is defined as an objective to be achieved by the system while an operational objective is called a constraint. Goals and constraints are defined formally using the patterns *achieve*, *cease*, *maintain*, *avoid* and *optimize* which are reused in our approach in the notion of a goal pattern. A difference is that the goal pattern in our approach is based on a PI expression. Soft goals are not considered in KAOS. Goals are structured and operationalized to constraints in AND/OR graphs. Temporal logic is used to define the goals and their relationships. In our approach a wider set of goals is considered, some of which cannot be expressed as temporal logic formulae. This reflects the way organizations define their goals in practice.

The NFR framework focuses on the representation of non-functional requirements on the designed software system through interrelated goals. Three types of goals are defined: NFR, satisficing and argumentation goals. The last two model design decisions and arguments resp. and are hence irrelevant for this discussion. The NFR goals are soft goals which can be refined using different types of relationships describing how the satisficing of the offspring relates to the satisficing of the parent goal. A labelling procedure is defined for determining the degree of satisficing of each node in the goal structure. The label propagation procedure used in our approach is inspired by but different from the one used in the NFR framework. We consider only positive refinement links. The negative links are modelled using conflict links. Furthermore, we enrich the refinement structure with one more relation in addition to AND and OR representing balanced contribution and providing tools for finer definition of how a set of goals together contributes to the satisficing of the higher-level goal.

8 Conclusions

This paper presents a formal goal-oriented modelling approach in the context of the performance-oriented view on organizations. The proposed approach is based on the essential idea that goals expressions should be explicitly defined over performance indicators of an organization. In such a way, a clear-cut relation between organizational objectives and performance measurements is established. The proposed approach includes a diverse vocabulary to express goal-related concepts and relations. Goals are classified along different dimensions: in particular, hardness (hard and soft)

and ownership (organizational and individual). Furthermore, the mechanisms for identifying (the level of) satisfaction/satisficing of goals are defined in the paper, where a special attention is given to the propagation of satisficing labels for soft goals. Often the estimation (evaluation) of soft goals is not straightforward. The paper proposes various means for fine-tuned modelling and evaluation of soft goals, among which the possibility to define different degrees of satisficing of goals and contributions in goal structures (including a balanced list), a variable quantification scale used for the performance evaluation. The potential downside of such expressivity and design freedom is that some organizational models may suffer from the subjectivity of the designer with respect to the choice of scales and types of influences. Because of the high domain dependency, it is difficult to formulate general design principles to specify soft parts of a model and to verify the correctness of certain design choices afterwards. However, some work has been done [13] to apply probabilistic reasoning and statistics based on accumulated data about organizational processes from the past to justify the correctness of the designed model. However, such information is not always available or cannot be used, e.g., when new (or modified) organizational structure and processes are considered. One reasonable suggestion to decrease the level of subjectivity of a designed model is to involve different organizational stakeholders (modellers, domain experts) in the process of determining quantitative measures and degrees of influence for soft parts of the model. Furthermore, performing simulations based on an organizational model with different parameters related to soft goals is helpful for identifying an acceptable design.

Further, the paper presents some guidelines and techniques for building consistent goal structures. For this relations to other concepts from the related views (such as PIs, tasks) are used. To create a consistent and correct model for an organization for any view, interdependencies with other related views should be identified and employed already at the early design stage. For example, organizational goals often have a great impact on the form of an organizational structure (i.e., choice of roles and relations between them), thus this should be reflected in an organization-oriented model for an organization. A detailed description of other views as well as design and analysis issues that concern relations between views are not considered in this paper and will be described elsewhere.

References

1. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems, 8 (2004) 203-236
2. CIMOSA – Open System Architecture for CIM. ESPRIT Consortium AMICE, Springer-Verlag, Berlin (1993)
3. Cohen, P.R. and Levesque, H.J.: Communicative Actions for Artificial Agents. In: Proceedings of the International Conference on Multi-Agent Systems. AAAI Press, 1995.
4. Dardenne, A., van Lamsweerde, A., Fiskas, S.: Goal Directed Requirements Acquisition. Science of Computer Programming, 20 (1993) 3-50
5. Fox, M.S.: The TOVE Project: Towards a Common-Sense Model of the Enterprise. In Petrie, C.J., Jr.(Ed): Proceedings of ICIEMT'92. MIT Press (1992) 310-319

6. Fox, M.S., Barbuceanu, M., Gruninger M., and Lin, J.: An Organization ontology for enterprise modelling. In: M. Prietula, K. Carley and L. Gasser (Eds): Simulating Organizational: Computational Models of Institutions and Groups, AAAI/MIT Press (1997) 131-152
7. Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P.: Model Checking Early Requirements Specification in Tropos. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE'01) (2001) 174-181
8. Jureta I., and Faulkner, S.: An Agent-oriented meta-model for enterprise modelling. In: J. Akoka et al (Eds): ER Workshops 2005, LNCS 3770 (2005) 151-161
9. Kavakli, E., and Loucopoulos, P.: Goal Driven Requirements Engineering: Evaluation of Current Methods. Proc. 8th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD '03) (2003)
10. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: a Guided Tour. In: Proceedings of 5th IEEE International Symposium on Requirements Engineering (2001) 249-263
11. van Lamsweerde, A., Darimont, R., Letier, E.: Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Transaction on Software Engineering, vol. 24(11) (1998) 908-926
12. van Linder, B., van der Hoek, W., and Meyer, J.-J. Ch.: Formalising motivational attitudes of agents: On preferences, goals and commitments. In: Wooldridge, M., Mueller, J.P., and Tambe, M. (eds.), *Intelligent Agents Volume II* (ATAL'95), vol. LNCS 1037, Springer (1996) 17-32
13. Letier E., and Lamsweerde, A. van: Reasoning about partial goal satisfaction for requirements and design engineering. Proceedings of the 12th ACM SIGSOFT International symposium on Foundations of software engineering (2004) 53 – 62
14. Manzano, M.: Extensions of First Order Logic, Cambridge University Press (1996)
15. Mylopoulos, J., Chung, L., Nixon, B.: Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering*, vol. 18, no. 6 (1992) 483-497
16. Popova, V. and Sharpanskykh, A., Modelling Organizational Performance Indicators. In: Barros, F. et al. (eds), Proceedings of the International Modeling and Simulation Multiconference, IMSM'07, SCS Press (2007) 165-170
17. Rao, A.S.: Decision Procedures for propositional linear-time belief-desire-intention logics. In: Wooldridge, M.J., Mueller, J.P. and Tambe, M. (eds.), Intelligent Agents II, vol. LNAI 1037, Springer (1996) 33-48
18. Sycara, K.: Resolving Goal Conflicts via Negotiation. In Proceedings of the Seventh National Conference on Artificial Intelligence (1988) 245-250
19. Yu, E.:. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. 3rd IEEE Int. Symposium on Requirements Engineering (1997) 226-235

Appendix A. A specification of goals for the case study

name: **G1**
informal definition: It is required to maintain that the level of correctness of plans with respect to the contracts of the employees, CAO, de Arbeidstijdenwet, the general company policy, the policy of the business unit Security is very high.
priority: very high
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: non-negotiable
perspective: society

name: **G1.1**
informal definition: It is required to maintain that the level of knowledge of employees involved in (forward) planning about the current contracts of the employees, CAO, de Arbeidstijdenwet, the general company policy, the policy of business unit Security is very high.
priority: very high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: non-negotiable
perspective: society

name: **G1.2**
informal definition: It is required to maintain that the level of up-to-dateness of the software system used in (forward and daily) planning with respect to the contracts of the employees, CAO, de Arbeidstijdenwet, the general policy of Falck/the policy of BU Security is very high.
priority: very high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: non-negotiable
perspective: society

G1.3 = G2.2

name: **G2**
informal definition: It is required to maintain high effectiveness of allocation of security officers
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management, customer

name: **G 2.1**
informal definition: It is required to maintain high average correctness of produced plans.
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.1.1**
informal definition: It is required to achieve high level of correctness of every produced plan.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.1.1.1**
informal definition: It is required to achieve high level of correctness of every produced forward plan.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.1.1.2**
informal definition: It is required to achieve high level of correctness of every produced daily plan.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.1.1.3**
informal definition: It is required to achieve high level of correctness of every produced short-term plan.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.2**
informal definition: It is required to maintain high level of correctness of administrative processing of all planning data in the system.
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.2.1**
informal definition: It is required to maintain high level of correctness of administrative processing of forward planning data in the system.
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.2.2**
informal definition: It is required to maintain high level of correctness of administrative processing of short-term and daily planning data in the system.
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.3**
informal definition: It is required to maintain low average level of deviation from daily plans in their application.
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G2.3.1**
informal definition: It is required to achieve low level of deviation from the produced daily plan in its application.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational

hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.3.1.1**
informal definition: It is required to achieve that the number of concerned security officers not informed on time about the produced daily plan is zero.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: hard
negotiability: negotiable
perspective: management

name: **G 2.3.1.2**
informal definition: It is required to achieve that the promptness of data change forms delivery by security officers is very high
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.3.1.3**
informal definition: It is required to achieve that the promptness of data change forms delivery to the planners is very high
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 2.3.1.4**
informal definition: It is required to achieve that the level of correctness of data change forms delivered to the planners is very high
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3**
informal definition: It is required to maintain high efficiency of planning and allocation of security officers
priority: high

horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3.1**
informal definition: It is required to maintain high efficiency of the planning process.
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3.1.1**
informal definition: It is required to achieve that the number of times the planning activities (creating and updating of a plan) exceed the allowed durations is equal to 0.
priority: high
horizon: long-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: hard
negotiability: negotiable
perspective: management

name: **G 3.1.1.1**
informal definition: It is required to achieve that the time to update a short-term plan given operational data is at most 48 hours.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: hard
negotiability: negotiable
perspective: management

name: **G 3.1.1.2**
informal definition: It is required to achieve that the time to create a daily plan given operational data is at most 24 hours
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: hard
negotiability: negotiable
perspective: management

name: **G 3.1.1.3**

informal definition: It is required to achieve that the time to create a short-term plan after all operational data is received is at most a week.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: hard
negotiability: negotiable
perspective: management

name: **G 3.1.1.4**
informal definition: It is required to achieve that the time to create a forward plan after all operational data is received is at most a week.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: hard
negotiability: negotiable
perspective: management

name: **G 3.1.2**
informal definition: It is required to maintain high level of promptness of communication of forward, short-term and daily planning data to all concerned employees.
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3.1.2.1**
informal definition: It is required to achieve high level of promptness of communication of every produced forward plan to all concerned employees.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3.1.2.2**
informal definition: It is required to achieve high level of promptness of communication of every produced short-term plan to all concerned employees.
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable

perspective: management

G 3.1.2.3 = G 2.3.1.1

name: **G 3.2**
informal definition: It is required to maintain high efficiency of allocation of security officers
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3.2.1**
informal definition: It is required to maintain high average level of optimality of forward, short-term and daily planning for efficient allocation of security officers
priority: high
horizon: long-term
evaluation type: development goal (maintain)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3.2.1.1**
informal definition: It is required to achieve high level of optimality of every daily plan for efficient allocation of security officers
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3.2.1.2**
informal definition: It is required to achieve high level of optimality of every forward plan for efficient allocation of security officers
priority: high
horizon: short-term
evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

name: **G 3.2.1.3**
informal definition: It is required to achieve high level of optimality of every short-term plan for efficient allocation of security officers
priority: high
horizon: short-term

evaluation type: achievement goal (achieve)
ownership: organisational
hardness: soft
negotiability: negotiable
perspective: management

Chapter 3

Process-oriented organization modeling and analysis [1]

Abstract. This paper presents a formal framework for process-oriented modelling and analysis of organisations. The high expressivity of the sorted predicate logic language used for specification allows representing a wide range of process-related concepts (e.g. tasks, processes, resources), characteristics and relations, which are described in the paper. Furthermore, for every organisation, structural and behavioural constraints on process-related concepts can be identified. Some of them should always be fulfilled by the organisation (e.g. physical world constraints), whereas others allow some degree of organisational flexibility (e.g. some domain specific constraints). An organisational specification is correct if it satisfies a set of relevant organisational constraints. This paper describes automated formal techniques for establishing correctness of organisational specifications with respect to a set of diverse constraint types. The introduced framework is a part of a general framework for organisation modelling and analysis.

1 Introduction

Every organisation achieves its goals by performing a set of tasks. Tasks are defined as organisational functions and their specification depends on the organisational type. For example, in mechanistic organisations (Scott 2001) each task is characterized by a detailed procedure that describes every aspect of the task execution, whereas in

[1] Part of this chapter appeared as Popova, V., Sharpanskykh, A.: Process-Oriented Organization Modeling and Analysis. In: J.C.Augusto, J. Barjis, U. Ultes-Nitsche (eds.), Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS 2007), INSTICC Press, 114-126. (2007) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

organic organisations (Scott 2001) a task specification may consist only of interface (i.e. input and output) characteristics and a general task description. Many modern organisations combine different features of both mechanistic and organic organisations. For example, management tasks of an organisation are usually specified at a high level of abstraction, whereas routine production tasks are often defined by detailed procedures.

The execution of tasks is often specified by dynamic structures called flows of control (or workflows), in which tasks are represented by processes. Usually control flows are based on a set of temporal ordering rules over processes.

Mechanistic organisations are often characterized by complex (hierarchical) structures of tasks and processes, whereas in organic organisations these structures are relatively simple, however, constantly varying. Furthermore, both types of organisations include a variety of relations between tasks and processes and other organisational concepts (e.g. resources, roles, agents). Therefore, to handle the high complexity of modern organisations that often possess features of both mechanistic and organic types, automated modelling and analysis techniques are required. Furthermore, to enable automation and guarantee the correctness of analysis, both modelling and analysis technique should be formal.

To this end, this paper introduces a formal framework for process-oriented modelling and analysis. In this framework, tasks, processes, resources and other related concepts are specified in the formal language L_{PR}, based on the sorted first-order predicate logic (Manzano 1996). The high expressivity of predicate logic allows including into L_{PR} a wide range of process-oriented concepts specified by sorts, sorted constants, variables, functions and predicates that represent relations on these concepts. The domains for sorts are considered to be finite, which allows performing effective reasoning and computational analysis on process-oriented specifications of organisations in L_{PR}. Such specifications correspond to structures over L_{PR} that are defined by the particular interpretations of sorts, constants, functions and predicates, and variable assignments.

For every organisation a set of structural and behavioural *constraints* expressed over its tasks and processes can de identified, which should be satisfied by the process-oriented specification. In this paper the set of constraints is represented by the *logical theory* T_{PR} in L_{PR}, i.e. a set of sentences expressed in L_{PR}. This means that all concepts and relations defined in L_{PR} may be used for the specification of constraints. A process-oriented specification in L_{PR} is *correct* if T_{PR} is satisfied by this specification, i.e. all sentences in theory T_{PR} are true in the logical structure corresponding to the specification.

The constraints in T_{PR} may be of different types: some are dictated by the restrictions of the physical world and should be satisfied by any process-oriented specification; others depend on the application domain and may be changed by the designer. The classification of constraints is described in this paper. Furthermore, this paper also introduces automated techniques for establishing the correctness of a process-oriented specification by verifying constraints. Interdependences that may exist in constraint sets are also handled by the proposed verification techniques. To our knowledge there exist no other frameworks that allow the simultaneous verification of different (interdependent) types of constraints based on the extensive set of concepts and relations as can be found in L_{PR}.

The framework introduced in this paper constitutes a part of a general formal framework for organisation modelling and analysis (Popova and Sharpanskykh 2007a) in which organisations are considered from other perspectives (or views) as well. In particular, *the performance-oriented view* (Popova and Sharpanskykh 2007c, Popova and Sharpanskykh 2007d) describes organisational goal structures, performance indicators structures, and relations between them. Within *the organisation-oriented view* (Broek et *al*. 2006, Sharpanskykh 2007) organisational roles, their authority and interaction relations are defined. In *the agent-oriented view* (Popova and Sharpanskykh 2007a) different types of agents with their characteristics and behaviour are identified and principles for allocating agents to roles are formulated. Concepts and relations within every view are formally described using dedicated languages based on the expressive order-sorted predicate logic. Furthermore, the views are connected to each other by means of sets of relations. This enables different types of analysis across different views. An example of such analysis involving the process- and performance-oriented views is the organisational performance evaluation considered in (Popova and Sharpanskykh 2007c). The relations between processes on the one hand and goals, performance indicators, roles and agents on the other hand are introduced in this paper.

The proposed views and concepts of the framework are similar to the ones defined in the Generalized Enterprise Reference Architecture and Methodology (GERAM) (Bernus *et al*. 1998) developed by the IFIP/IFAC Task Force, which forms a basis for comparison of the existing enterprise architectures and serves as a template for the development of new enterprise modelling frameworks. Although many enterprise architectures include a rich ontological basis for creating specifications of different views, most of them provide only a limited support for automated analysis of these specifications, addressed in the category *Enterprise Engineering Tools* of GERAM, primarily due to the lack of formal foundations in these frameworks. In contrast, the proposed framework enables different types of automated analysis both within particular views and across different views (Popova and Sharpanskykh 2007a).

The paper is organized as follows. First, in Section 2, the running example for this paper is described. Section 3 introduces the language L_{PR}. Section 4 describes the classification of constraints. In Section 5, the methods for verification of constraints are given. In Section 6 the related work on process-oriented modelling is discussed. Section 7 concludes the paper.

2 The case study

To illustrate different aspects of our approach a running example is used that describes the operation of a 3PL (third-party logistics) provider. In general, 3PL companies provide logistics services to other companies. The considered operation cycle begins with the customer order intake process, after which the order is processed and depending on the customer (company) is scheduled for some delivery type (for different companies different delivery regulations may be applied). During the delivery the assigned driver is supervised by the assigned fleet manager. After the delivery is finished, the delivery summary report is provided to the customer.

In the context of this example consider a particular delivery scenario (see Fig. 1): The logistics company performs shipments of resources between three bases of some manufacturing enterprise (A, B and C), which are located in different regions. The base A has the storage facilities for both raw materials delivered by suppliers and for finished products of the enterprise prepared for further shipments to customers. The raw materials stored at A are required for the production processes of the enterprise's departments located at B and C (the resources of type rt1 - for B and the resources of type rt2 - for C) and are transported from A to these departments by trucks. Each delivery is preceded by the process of resource loading and is followed by the resource unloading process. The 3PL company owns trucks of two types: large trucks of the type tr1 with three capacity units each (a capacity unit is a relative spatial capacity measure of a truck) and small trucks of the type tr2 with one capacity unit each. All deliveries between the bases of the enterprise are performed by two trucks of the type tr1 and three trucks of the type tr2. The base C is geographically located between A and B, however also a more direct connection between A and B exists. At C two types of products are produced: the finished product of type rt4, which should be shipped to A, and the intermediate product of type rt3 that is used as an assembling part at B. Using raw materials of type rt1 and products of type rt3 the department at B produces finished products of type rt5, which should be subsequently shipped to the storage facilities at A.

In the considered scenario initially all trucks are located at the base A. The company assigns one truck of type tr1 to the direct delivery d1 of the materials of type rt1 from A to B. The delivery d2 of resources of type rt1 from A to B through C starts simultaneously with d1. The delivery d2 shares two trucks of the type tr2 with d3, the delivery of resources of type rt2 from A to C. The mode of sharing of each tr2 type truck is determined by particular weight and spatial restrictions. In particular, the weight limitations prescribe that at most 70% of the capacity unit of a tr2 truck can be filled with resources of type rt2, whereas the remaining 30% should be left empty. When a tr2 truck is shared between resources of types rt1 and rt2 (or of types rt1 and rt3), then the resources of type rt1 may occupy at most 50% of the truck's space. Moreover, one more tr2 type truck is assigned to d3. When d3 is finished and the resources of type rt2 are unloaded, this truck is scheduled to be loaded with resources of type rt4 and to return back to A (the delivery d6). After the resources of the type rt2 are unloaded at C from two trucks of the type tr2, the emptied space in these trucks will be promptly filled with products of type rt3, which are scheduled to be delivered to the base B (delivery d4). The deliveries d2 and d4 again share the tr2 trucks equally. After both d4 and d2 are finished, both tr2 trucks will be unloaded and loaded with finished products of the type rt5, which are scheduled to be delivered to A (the delivery d7). Similarly, the truck used for d1, after being unloaded at B, will be loaded with products of type rt5 that are required to be delivered to A (delivery d5).

3 Process-oriented modelling

Process-oriented specifications in the proposed framework are specified using the sorted predicate language L_{PR}. In this Section, a general overview of L_{PR} is given.

A *task* represents a function performed in the organisation and is characterized by a name and by a maximal and a minimal durations. The sort TASK contains the names of all tasks. The characteristics of the tasks are specified by the following predicate: task: TASK × TASK_PROPERTY × VALUE ∪ STRING. We sometimes use the following short notation for specifying these characteristics: t.p = v, where t∈TASK, p∈TASK_PROPERTY and v∈ VALUE ∪ STRING. Characteristics of other concepts defined below are formalized in a similar way. For example, the task Order_intake has minimal duration 1 hour and maximal duration 2 hours depending on the experience and the efficiency of the agent performing the task: Order_intake.min_duration=1h, Order_intake.max_duration=2h.

Tasks can range from very general to very specific. General tasks can be decomposed into more specific ones using AND- and OR-relations thus forming hierarchies. It is specified using the following predicates:

is_in_task_list: TASK × TASK_LIST specifies a task is a member of a task list

is_decomposed_to: TASK × TASK_LIST specifies that a task is decomposed into an AND-list of tasks meaning that all tasks in the task list together are necessary and sufficient in order to perform the decomposed task. Sometimes alternative decompositions of a task are possible which are connected by an OR-relation. They are specified as separate decompositions of the same task. For example the task DeliveryComp1 which encompasses all deliveries for Company 1, as described in Section 2, can be decomposed in several types of deliveries, modelled as more specific tasks in the task hierarchy of DeliveryComp1, which are all necessary for fulfilling the agreements with the client such as: delivery from A to B (DeliveryAB), from B to A (DeliveryBA), from A to C (DeliveryAC), from C to B (DeliveryCB) and from C to A (DeliveryCA). Therefore they should be modelled as an AND-decomposition list of task DeliveryComp1:

is_decomposed_to(DeliveryComp1, L)
is_in_task_list(DeliveryAB, L)
is_in_task_list(DeliveryBA, L)
is_in_task_list(DeliveryAC, L)
is_in_task_list(DeliveryCB, L)
is_in_task_list(DeliveryCA, L).

These deliveries can be executed in different ways. For example DeliveryAB, which should transport 4 capacity units of resource type rt1, can be performed by two big trucks (DeliveryAB20) or by one big and two small trucks (DeliveryAB12) or by four small trucks (DeliveryAB04). Such information can be represented as alternative decompositions of the task DeliveryAB:

is_decomposed_to(DeliveryAB, L1)
is_in_task_list(DeliveryAB20, L1)
is_decomposed_to(DeliveryAB, L2)
is_in_task_list(DeliveryAB12, L2)
is_decomposed_to(DeliveryAB, L3)
is_in_task_list(DeliveryAB04, L3).

These alternative subtasks can be decomposed further using the information that there are two possible routes between A and B – direct and via C. For example, for DeliveryAB12 it was decided to split the trucks so that one big truck travels directly from A to B and two small trucks travel via C.

A *workflow* is defined by a set of (partially) temporally ordered *processes*. Each process, except for the special ones with zero duration introduced below, is defined

using a task as a template and all characteristics of the task are inherited by the process. This is specified using the predicate: is_instance_of: PROCESS × TASK, e.g. is_instance_of(d1, DeliveryAB). Decisions are also treated as processes that are associated with decision variables taking as possible values the possible decision outcomes.

Definition 1 (A workflow): A workflow with the name w is defined by a tuple <w, P, C> with a set of processes P and a set of ordering relations C on processes from P.

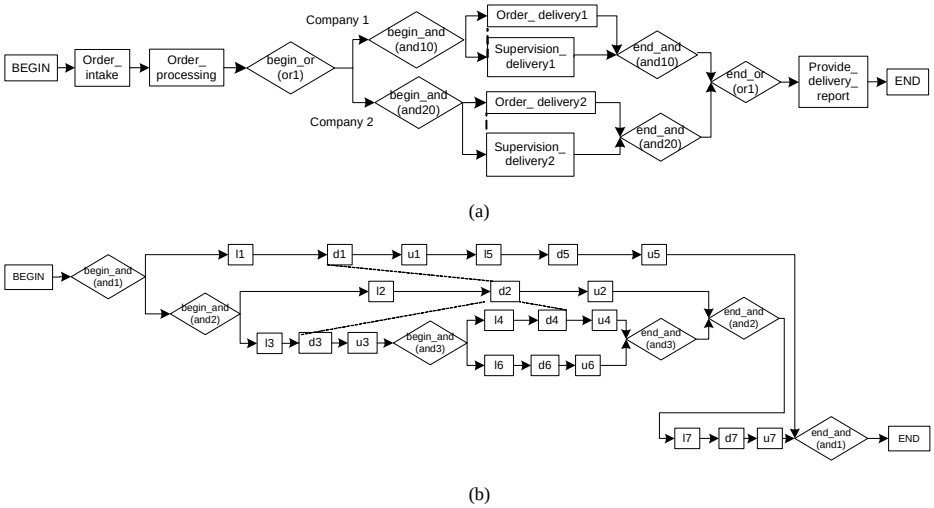


Fig. 1. The generalized workflow that illustrates the operation of a 3PL delivery company (a) and the detailed workflow for the deliveries for a specific company (b).

Fig.1 is a graphical representation of the workflow built for the running example. It shows the processes at two different aggregation levels. Fig.1a shows the overall workflow for processing and executing an incoming order. Fig.1b is a more detailed description of the workflow corresponding to process Order_delivery1 for the orders of Company 1. Analogous detailed workflow can be built for the deliveries to Company 2 using relevant information (left out of the case study for simplicity).

A workflow starts with the process BEGIN and ends with the process END; both have zero duration. The (partial) order of execution of processes in the workflow is defined by sequencing, branching, loop and synchronization relations (referred to as ordering relations) specified by the designer.

A *sequencing relation* is specified by the predicate starts_after: PROCESS × PROCESS × VALUE expressing that the process specified by the first argument starts after the process specified by the second argument with the delay expressed by the third argument, e.g. starts_after(Order_processing, Order_intake, 0) represented graphically by solid arrows between the processes. For each process p, different from BEGIN and END at least two sequencing constraints are defined, which specify the process that precedes p and the process which follows after p.

By specifying sequencing constraints different paths of a workflow are formed. A *path* is a sequence of processes $(p_1, p_2, \ldots, p_{n-1}, p_n)$, where n>1 and starts_after$(p_2, p_1) \wedge$

starts_after(p_3, p_2) $\wedge$ ... $\wedge$ starts_after(p_n, p_{n-1}). A path in a workflow, in which $p_1 = p_n$ is called a *cycle*. No cycles are allowed in the workflow structures.

Synchronization relations define temporal relations between processes that are executed in parallel (e.g. starts_with, finishes_with, starts_during: PROCESS $\times$ PROCESS). An example of such a relation is shown by a dashed line between the beginnings or endings of the processes in Fig. 1, meaning that the connected processes should start or finish simultaneously. For example: starts_with(d2,d3), finishes_with(d2,d4) which comes from the fact that these deliveries share trucks and therefore it is impossible to have them start / finish at different times. Taken together, synchronization and sequencing relations allow specifying all cases of interval relations defined in (Allen 1983).

Branching relations are defined over and- and or-structures. An and(or)-structure with name id, starts with the zero-duration process begin_and(id) (begin_or(id)) and finishes by the zero-duration process end_and(id) (end_or(id)). These special processes are represented graphically by rhombuses. Our treatment of AND-structures is similar to the parallel split pattern combined with all types of the merge pattern from (van der Aalst *et al*. 2003), represented in our case by an and-condition. The first processes in every branch of an and-structure start at the same time. For each and-structure a condition is defined (and_cond: AND_STRUCTURE $\times$ CONDITION_EXPRESSION), which determines when the process p following after the and-structure may start. The following types of conditions may be used: (1) constant any: meaning that as soon as all processes of one of the branches finish, the process p starts; (2) constant all: meaning that as soon as all processes of all branches finish, the process p starts; (3) a condition expressed by a logical formula constructed from the functions finished, not_finished: PROCESS $\rightarrow$ {true, false} using Boolean connectives $\vee$ and $\wedge$. In Fig.1a the and-structures contain the condition value all, meaning that only when all processes in the and-structure are finished, the process specified after the end of the and-structure is allowed to start. This is represented formally as: and_cond(and0, all). In Fig.1b AND-structure and3 contains condition finished(u4) therefore the execution of the workflow will continue only when process u4 finishes irrespective of whether the last process of the other branch (u6) is finished or not. The reason for this is that the branch of d6 (corresponding to the delivery from C to A) does not influence the processes after the AND-structure, thus, they can start theirs execution without waiting.

For every or-structure a condition is defined (or_cond: OR_STRUCTURE $\times$ CONDITION_EXPRESSION), based on which it is determined which branches of the or-structure will start. The condition may consist only of a condition variable or it may be a disjunction of conjunctions of expressions in the form condition_variable [OP value], where OP$\in${=, $\neq$, <, >}, and value belongs to the domain of the condition variable. The following types of condition variables can be used: (1) a decision variable; (2) a variable over the sort that includes all states of a certain object in the environment, e.g. market conditions, customer demand, taxes, weather conditions, etc.; (3) a variable over the sort that includes all values of certain characteristic of an object in the environment. For an or-structure branches are specified that correspond to all possible values of the condition expression, using the predicate or_branch: OR_COND_VALUE $\times$ PROCESS, which expresses that the branch of the or-structure that begins with the process specified in the second argument corresponds to the value of the condition expression specified in the first argument. An or-branch may correspond

to the constant other, which should be interpreted as all other values from the domain of the condition variable. Our treatment of or-structures allows realizing both exclusive and multiple-choice patterns from (van der Aalst *et al*. 2003). The or-structure in the running example, Fig.1a specifies the exclusive choice between two types of delivery depending on the company name. Here the company name is the characteristic company of the environmental object incoming_order and takes two possible values, Company1 and Company2. Therefore the condition of the OR-structure can be represented in the following way:

or_cond(or1, incoming_order.company)
or_branch(Company1, begin_and(and1))
or_branch(Company2, begin_and(and2))

Loop relations are defined over *loop*-structures with conditions that realize cycle patterns from (van der Aalst *et al*. 2003). A loop-structure with name id, starts with the zero-duration process begin_loop(id) and finishes by the zero-duration process end_loop(id). For every loop-structure a Boolean condition (loop_cond: LOOP × CONDITION_EXPRESSION) and the maximal number of times of the loop execution (loop_max: LOOP × VALUE) are specified. No cycles were identified for the running example.

Tasks use, consume and produce resources of different types. Resource types include tools, supplies, components and other material or digital artefacts. Also data are considered as a special resource type. The predicates task_uses, task_consumes, task_produces: TASK × RESOURCE_TYPE × VALUE indicate resource types that are input or output of tasks with their prescribed amounts. Resource types are characterized by: *name*; *category* – discrete or continuous; *measurement_unit*; *expiration_duration* – the length d of the time interval for which a resource type can be used. Specific resources represent instances of particular resource types and inherit their characteristics. The resources have, in addition to the inherited characteristics, also *name* and *amount*. Every resource in the workflow has to be produced by a process of this workflow or be available in the organisation before the beginning of the workflow execution.

For the running example, 5 resource types were identified rt1 to rt5 corresponding to the raw materials, components and products described in Section 2 as well as tr1 and tr2 corresponding to the two types of trucks available for the deliveries. For each task which uses a resource type, the corresponding necessary amount can be specified e.g. task_uses(deliveryAB12, tr1, 1), task_uses(deliveryAB12, tr2, 2), task_uses(deliveryAB12, rt1, 3). Furthermore no resource type is consumed or produced by a task in the example since the operations of the client company are considered out of the scope and are not part of this workflow.

Resource types can sometimes be functionally divisible, i.e. they can be divided in parts in such a way that a part can have a different purpose, therefore is a different resource type. For example a car can be decomposed to its components which do not have the same purpose as the car. This is specified by the following predicate:

is_func_part_of: RESOURCE_TYPE × RESOURCE_TYPE where the second resource type is functionally divisible and the first resource type is its functional part. For example computer might be defined as a functionally divisible resource since its parts have different purpose than the whole computer.

Some resources can be shared (used simultaneously) by a set of processes (e.g. storage facilities, transportation vehicles, some computers). The predicate resource_sharable: RESOURCE_TYPE × PROCESS_LIST defines that the resource type can be used (but not consumed!) by the processes in the list (or a sub-list of this list) at the same time. The shared amount of the resource type should be sufficient for the execution of every process in the process list. Alternative sharing of the same resource type can be specified as well. Our representation of shared resources is different from (Barkaoui and Petrucci 1998) in several aspects: (1) the shared resource amount is used by processes simultaneously; (2) alternative sets of processes that are allowed to share a resource can be defined; (3) different amounts of a resource can be shared simultaneously; (4) specific conditions (requirements) for resource sharing can be defined. In the running example, trucks may be considered as shared resources. Delivery processes d2, d3 and d2, d4 are required to share trucks of type tr2:

```
resource_sharable(tr2, L1)
resource_sharable(tr2, L2)
is_in_process_list(d2, L1)
is_in_process_list(d3, L1)
is_in_process_list(d2, L2)
is_in_process_list(d4, L2)
```

Every resource in the workflow has to be produced by a process of this workflow. This is specified using the predicate: process_output: PROCESS × RESOURCE. In this way the end time point of the process producing the resource is taken as creation time for this resource and the time when it will expire is calculated with respect to this creation time. A process can produce only one resource instance of the same resource type. In some situations, resources could be available in the organisation before the beginning of the workflow execution, for example machines and other durable tools, materials already purchased, etc. In these cases such resources that will be used in the workflow are specified as output of its process BEGIN. For the running example, the used resources are considered available at the beginning of the workflow i.e. produced by the BEGIN process:

```
process_output(BEGIN, r1)
r1.amount = 4
is_instance_of(r1, rt1)
etc.
```

For some application areas it is important to keep track of where the resources are at certain time points. For modelling such information the concept of a *location* is used which for example can represent the available storage facilities. Processes can add or remove resource types from locations which can be specified using the predicates: process_adds_resource_type_to: PROCESS × RESOURCE_TYPE × LOCATION × VALUE and process_rem_resource_type_from: PROCESS × RESOURCE_TYPE × LOCATION × VALUE where the last argument specifies the amount of the added or removed resource type. Resources are considered removed at the starting time point of the corresponding process and are added at the ending time point of the process.

For the running example we define three locations corresponding to the storage facilities of A, B and C. Only delivery processes are expected to add or remove resource types from locations (during loading and unloading the resources are considered to be still at the same location). Therefore we can specify that:

process_rem_resource_type_from(d2, rt1, A, 2)
process_adds_resource_type_to(d2, rt1, B, 2)
etc.

The process-oriented view is related to the *organisation-oriented* and the *agent-oriented views* through the sorts ROLE and AGENT. Each object of the sort ROLE describes a set of functionalities realized by organisational processes in a certain specification, which are assigned together to individuals who will be performing them. These individuals are objects of the sort AGENT. An agent can be allocated to one or more roles if it satisfies the requirements for performing these roles. For example, role_performs_process(Driver,d1) and agent_plays_role(Allan, Driver). For more details the reader is referred to (Popova and Sharpanskykh 2007e).

The process-oriented view is also related to the *performance-oriented view* through the sorts GOAL and PI (performance indicator). Objects of sort GOAL are organisational objectives and are defined as expressions based on performance indicators (objects of sort PI). The performance-oriented view is discussed in detail in (Popova and Sharpanskykh 2007c, Popova and Sharpanskykh 2007d). It relates to the process-oriented view through the following predicates:

is_realizable_by: GOAL × TASK_LIST – the goal can be realized by the tasks in the list

measures: PI × PROCESS – the performance indicator expresses an aspect of the performance of the process.

4 Constraints

Constraints are expressed as formulae in theory T_{PR} that are constructed from terms of L_{PR} in a standard way (Manzano 1996) using Boolean connectives and quantifiers over variables. The constraints are divided in two groups: (1) *generic constraints* need to be satisfied by any specification built using this framework; (2) *domain-specific constraints* are dictated by the application domain of the specification. Two types of generic constraints are considered: (1) structural constraints used to ensure correctness of the workflow, task and resource hierarchies; (2) constraints imposed by the physical world. Both types of generic constraints are described in Sections 4.1. Section 4.2 discusses the domain-specific constraints.

4.1 Generic constraints

The language allows building three types of structures: the workflow, the task hierarchy and the resource hierarchy. For each of them structural constraints are defined.

Workflow structural constraints

With respect to the workflow we define a set of structural constraints: structural correctness, temporal correctness and condition correctness constraints.

Structural correctness of the workflow

First let us introduce *reachability* and *complete reachablitity* relations.

Definition 2 (Reachability relation): The process p2 is reachable from the process p1 in the workflow w (reachable_from_in(p2, p1, w)) if there exists a sequence of processes constructed using the sequencing relations that starts at p1 and includes p2.

On Fig.1b, for example, process l7 is reachable from process u3 through two sequences of processes, one of which is: u3, begin_and(and3), l4, d4, u4, end_and(and3), end_and(and2), l7. However process u3 is not reachable from process l7. Also, for example, process u3 is not reachable from process u2 and process u2 is not reachable from process u3.

The truth value of the relation reachable_from_in(p2, p1, w) is determined as follows:

1. Initial settings: Let L be an empty queue and A be an empty set. Put p1 into L.
2. Until L becomes empty perform steps 3-6.
3. Dequeue L and assign the obtained process name to the variable curr_process.
4. Identify the set A = {a | starts_after(a, curr_process)}
5. If p2∈A, then return true.
6. Enqueue all elements of A in L.
7. Return false.

To investigate the correctness and the computational properties of this algorithm, a workflow <w, P, C> is represented as a unidirected graph G = (V, E), in which each vertex v ∈ V represents some organizational process p ∈ P, and each edge e ∈ E with the initial vertex representing the process x and the terminal vertex representing the process y corresponds to the relation starts_after(y, x). Note that loops represented in the process-oriented language by sequences of processes do not introduce cycles in the corresponding graph representation. The algorithm considers gradually all vertices in the graph that belong to the paths beginning from the vertex that represents the process p1, until it finds the process p2. Since each path in a workflow finishes with the vertex corresponding to the process END, any execution of the algorithm eventually terminates. Such type of algorithms is called breadth-first search and its correctness and computational properties are well established (cf., Cormen et al, 2001).

The time complexity of the proposed algorithm is estimated for the worst case under the assumption that each primitive operation (e.g., assignment of a value, extracting/placing from/into a queue) takes one time unit: The step 1 of the algorithm is performed only once and takes 1 time unit. The steps 2-6 in the worst case can be repeated |P|-1. The step 2 takes 1 time unit each time it is executed. The step 3 takes 2 time units each time when it is executed. All executions of the step 4 taken together in the worst case may take |P-1|·(|E|+1) time units. The step 5 does not take more than |P|-2 time units. The step 6 for all its executions taken together in the worst case may take |P|-2 time units. Thus, the overall time complexity of the algorithm for the worst case is O(|P|*(|E|+|P|). However, in most practical cases the time complexity is less than O(|P|*(|E|+|P|), since often subsets of |P| are considered.

Definition 3 (Complete reachability relation): The process p2 is completely reachable from the process p1 in the workflow w (completely_reachable_from_in(p2, p1, w)) if all process sequences built using the sequencing relations that start at p1 include p2.

For example, the process Provide_delivery_report from the running example, Fig.1a, is completely reachable from the process Order_intake but process Order_delivery1 is not completely reachable from the process Order_intake.

The truth value of the complete reachability relation completely_reachable_from_in(p2, p1, w) is determined by the following algorithm:

1. Initial settings: Let L be an empty queue and A be an empty set. Put p1 into L.
2. Until L is empty perform steps 3-5.
3. Dequeue L and assign the obtained process name to the variable curr_process.
4. Identify the set A={a | starts_after(a, curr_process)}
5. If A=∅, then return false.
   else if A≠{p2}, then enqueue all elements of A in L.
6. Return true.

Using the same type of representation of a workflow as in the previous algorithm (i.e., as a graph), it is easy to see that this algorithm is also a variation of breadth-first search. In contrast to the previous algorithm, this algorithm checks by the gradual consideration of the vertices, if each path that begins with the vertex corresponding to p1 also includes the vertex corresponding to p2. If one (or more) of the paths does not include p2, the condition in the line 5 of the algorithm will eventually become true (i.e., the vertex corresponding to the process END will eventually be reached, for which $\neg\exists$a starts_after(a, END), and therefore A = ∅).

The time complexity of this algorithm is calculated in a similar way as for the previous algorithm and is estimated for the worst case as O(|P|*(|E|+|P|).

Definition 4 (A well-formed and-structure): An and-structure with the name and_id defined in the workflow <w, P, C> is well-formed if the following constraints hold:

(1) $\exists p \in P$ such that p = begin_and(and_id); $\exists p \in P$ such that p = end_and(and_id);

(2) completely_reachable_from_in(end_and(and_id), begin_and(and_id), w) is true.

(3) each path of the and-structure is free of cycles.

Well-formed or- and loop-structures are defined similarly. Both and- and or-structures defined in the running example in Section 2 are well-formed.

Set of processes P(id) of an and-structure with the name id is constructed by the following procedure:

1. Initial settings: Let L be an empty queue and P(id) be an empty set. Put begin_and(id) into L.
2. Until L is empty perform steps 3-5.
3. Dequeue L and assign the obtained process name to the variable curr_process.
4. Identify the set A = {a | starts_after(a, curr_process)}
5. If A ≠ { end_and(p2)}, then put all the elements of A into P(id) and enqueue them in L.
6. Return P(id).

If the graph representation of a workflow is used as in the algorithms above, then the set of processes P(and1) of the and-structure and1 corresponds to the set of all vertices of the corresponding graph that belong to the paths beginning with the vertex representing the process begin_and(and1) and finishing with the vertex representing

the process end_and(and1). The vertices that belong to the considered paths are processed in the order as in the breadth-first search algorithm, therefore the time complexity is estimated for the worst case as O(|P|*(|E|+|P|).

Sets of processes for or- and loop-structures are defined in a similar way.
Now, the structural correctness property for a workflow can be introduced.

Definition 5 (A structurally correct workflow): A workflow <w, P, C> is structurally correct if the following constraints are satisfied:

(1) A workflow contains only one BEGIN (the first process), followed by one process and only one END (the last process) preceded by one process. Formally:

(a) $\exists s \in P\ \exists s1 \in P\ s=BEGIN \wedge starts\_after(s1, s) \wedge (\forall s2 \in P\ starts\_after(s2, s) \Rightarrow s2=s1)$
(b) $\exists s \in P\ \exists s1 \in P\ s=END \wedge starts\_after(s, s1) \wedge (\forall s2 \in P\ starts\_after(s, s2) \Rightarrow s2=s1)$
(c) $\forall s \in P\ (\neg starts\_after(BEGIN, s)) \wedge\ (\neg starts\_after(s, END))$

(2) For every process p, different from BEGIN, END, and the starting and ending processes for and- and or-structures, exactly two sequencing relations should be defined that identify the process that precedes p and the process that follows after p:

(a) $\exists s \in P\ starts\_after(p, s) \wedge (\forall s1 \in P\ starts\_after(p, s1) \Rightarrow s1=s)$
(b) $\exists s \in P\ starts\_after(s, p) \wedge (\forall s1 \in P\ starts\_after(s1, p) \Rightarrow s1=s)$

(3) Loops should be introduced only by loop-structures, no other cycles are allowed:

$\forall p1, p2 \in P\ reachable\_from\_in(p1, p2, w) \Rightarrow \neg reachable\_from\_in(p2, p1, w)$

(4) Processes, over which a synchronization constraint is specified, should not belong to the same or-structure. Formally: for each constraint from c in the form starts_with(p1, p2), finishes_with(p1, p2), starts_during(p1, p2):

$\neg\exists id{:}OR\_STRUCTURE\ p1 \in P(id) \wedge p2 \in P(id)$

(5) All and-, or- and loop-structures in w are well-formed and each process $p \in P$ can be reached from the BEGIN, and the END can be reached from each process:

$\forall p \in P\ reachable\_from\_in(p, BEGIN, w)\ \wedge reachable\_from\_in(END, p, w)$

(6) All and- and or-structures of a loop-structure should begin and finish within the loop. Formally:

For each loop-structure l from the workflow

$\forall a \in AND\_STRUCTURE\ [\ reachable\_from\_in(begin\_and(a), begin\_loop(l), w) \wedge reachable\_from\_in(end\_loop(l), begin\_and(a), w)\ ]$
$\Rightarrow [\ reachable\_from\_in(end\_and(a), begin\_loop(l), w) \wedge reachable\_from\_in(end\_loop(l), end\_and(a), w)\ ]$

Similarly for OR-structures.

(7) Each path of the workflow is free of cycles.

The workflow defined by the running example in Section 2 is structurally correct.

Temporal correctness of the workflow

The duration of each process in a workflow may vary in actual executions and, because of the temporal ordering of processes in the workflow, each process may have different starting points in different executions. Among all starting points the

earliest (est_p) and the latest starting time (lst_p) for each process p can be identified. Before describing a procedure for calculation these time parameters, let us introduce sets of relevant processes and relevant ordering relations, and an algorithm for their construction.

Let $PR(p)$ be a set of relevant processes with respect to $p \in P$ in the workflow $<w, P, C>$, i.e. $PR(p) \subset P$, such that each process in $PR(p)$ influences the starting time of p.

Let $CR(p)$ be a set of relevant ordering relations with respect to $p \in P$ in the workflow $<w, P, C>$, i.e. $CR(p) \subset C$, such that each relation in $CR(p)$ influences the starting time of p.

Sets of relevant processes and relevant ordering relations with respect to $p \in P$ in the workflow $<w, P, C>$ are constructed by the following algorithm:

1. *Initial settings*: Let L be an empty stack, and $PR(p)$ and $CR(p)$ be empty sets. Put p into L.
2. Repeat the steps 3-8 until L is empty, then exit.
3. Remove the element from the top of L and assign its name to the variable current_name.
4. Identify the set $C' = \{ c \in C \mid \exists s \in P [$ c=starts_after(current_name, s) $\vee$ c=starts_with(current_name, s) $\vee$ c=starts_with(s, current_name) $\vee$ c=finishes_with(current_name, s) $\vee$ c=finishes_with(s, current_name) $\vee$ c=starts_during(current_name, s)]$\}$
5. $CR(p) = CR(p) \cup C'$
6. Identify the set $P' = \{p \in P \mid \exists c \in C'$ c= starts_after(current_name, p) $\vee$ c=starts_with(current_name, p) $\vee$ c=starts_with(p, current_name) $\vee$ c=finishes_with(current_name, p) $\vee$ c=finishes_with(p, current_name) $\vee$ c=starts_during(current_name, p)]$\}$
7. $P' = P' \setminus \{p \mid p \in PR(p) \wedge p \in P'\}$
8. If $P' \neq \{START\}$, then $PR(p) = PR(p) \cup P'$ and add all elements of P' to L in any order.

In the graph representation of a workflow, this procedure traces back all paths that include the vertex corresponding to the process p, starting from p. The processes represented by the vertices that belong to the considered paths are included into the set PR(p). Furthermore, the set PR(p) also includes the processes that are not represented by vertices on the considered paths, however are related to these processes by synchronisation relations. The set CR(p) consists of the ordering constraints, formulated over the processes from PR(p). Each execution of this procedure will eventually terminate, since each considered path begins with the vertex representing the process BEGIN. When BEGIN is encountered on some path, no further vertices along this path will be added to the stack L.

The time complexity of this procedure is estimated for the worst case as follows: Given the workflow $<w, P, C>$, the internal cycle (steps 3-8) may be repeated $|P|$ times at most. The steps 4 and 6 take $|C|^2 \cdot |P|$ and $|C| \cdot |P|^2$ time units respectively for all executions of the cycle. The step 5 can be performed $|C|$ time units for all executions of the cycle taken together in the worst case. The execution of the step 7 takes $|P|^2 + 2 \cdot |P|$ time units for all executions of the cycle taken together. The execution of the step 8 takes $3 \cdot |P|$ time units for all executions of the cycle taken together. The overall time complexity of the procedure is estimated for the worst case as: $|C|^2 \cdot |P| + |C| \cdot |P|^2 + |C| + |P|^2 + 8 \cdot |P| + 1$.

The procedure of calculation of est_p and lst_p

The earliest (latest) starting time of a process p in the workflow <w, P, C> is calculated under the assumption that all relevant processes in PR(p) have minimal (maximal) durations specified in their characteristics. Furthermore, $est_{BEGIN} = lst_{BEGIN} = 0$.

To calculate est_p and lst_p in the workflow <w, P, C>:

1. Identify the relevant sets PR(p) and CR(p).
2. Assume that the duration of every $p \in P$ is defined by its property min_duration (max_duration).
3. The duration of every or-structure in PR(p) is equal to the duration of its shortest (longest) branch, calculated as the sum of minimum durations of processes that belong to the branch as follows. For the or-structure with the name or_struct:
   a) Initialization: Let ST be an empty stack, and min_duration=0, curr_duration=0.
   b) Put all elements of the set {a | starts_after(a, begin_or(or_struct))} into ST in any order.
   c) Until ST is not empty perform steps d)-h).
   d) Remove the element from the top of ST and assign its value to the variable curr_process.
   e) Until curr_process ≠ end_or(or_struct), perform steps f) and g).
   f) curr_duration= curr_duration + curr_process.min_duration
   g) curr_process= {a | starts_after(a, curr_process) }
   h) If min_duration=0 ∨ curr_duration < min_duration, then min_duration= curr_duration
   i) Return min_duration
4. The duration of every and-structure in PR(p) depends on its end-condition:
   a) in case the condition is 'any' the duration is determined by the shortest branch of the and-structure;
   b) in case the condition is 'all' the duration is determined by the longest branch of the and-structure;
   c) in case of a more complex condition defined by an and/or-list of processes, the duration is determined by processing the list recursively starting from the most nested parts of a condition:
      – the or-list is replaced by the process with the shortest duration in this list;
      – the and-list is replaced by the process with the longest duration in this list.

   The duration of the process obtained in the end is the duration of the and-structure.
5. To calculate the earliest starting point of the process p the duration of every loop-structure in PR(p) is counted as the sum of minimum durations of processes that belong to the loop. To calculate the latest starting point of the process p the duration of every loop-structure in PR(p) is counted as the product of the sum of maximum durations of processes that belong to the loop and the maximum amount of loop execution times.

The time complexity of this procedure can be estimated as follows: As it has been shown above, the execution of the step 1 takes $|C|^2 \cdot |P| + |C| \cdot |P|^2 + |C| + |P|^2 + 8 \cdot |P| + 1$ in the worst case. The complexity of processing of every or-structure is calculated under the assumption that each or-structure contains $m << |P(or1)|$ branches, where P(or1) is the set of processes of the or-structure calculated as shown above. Then, the execution of the step 3a for an or-structure with the id or1 takes 2 time units. The execution of the step 3b does not take more than $m \cdot |C|$ time units. The internal cycle 3d)-h) can be

repeated m times at most. The step 3d) takes 2 time units. The execution of the steps 3f)-g) takes in the worst case $O(|P(or1)|^2 \cdot |C|)$ time units. The overall time complexity for processing the or-structure with the id or1 is estimated as $O(|P(or1)|^2 \cdot |C|)$. The processing of an and-structure is performed using a similar sequence of steps, as for an or-structure, therefore the complexity of processing of an and-structure with the id and1 under the assumption that each and-structure contains $m<<|P(and1)|$ branches, is also $O(|P(and1)|^2 \cdot |C|)$.

At step 5 each loop of a workflow can be considered as a sequence of processes, in which each and- and or-structure is replaced by a single composite process. When the earliest and latest starting and ending time points of the and- and or-structures included into a loop have been calculated, the corresponding temporal parameters of the whole loop structure can be identified by gradual processing of the sequence of the processes of the loop. Thus, the time complexity of processing a loop with the id l1 is $O(|P(l1)| \cdot |C|)$.

As one can see, the time required for the execution of each step of the procedure is polynomial in number of the processes of corresponding structures of a workflow and in the number of constraints. The time complexity of the procedure for the whole workflow <w, P, C> for the worst case is not greater than $O(|P|^2 C)$.

The earliest (latest) ending time point of the process p (eet_p (let_p)) is calculated as est_p + p.min_duration (lst_p + p.max_duration). Then, the earliest (latest) creation time of the resource r (ect_r (lct_r)) produced by p are defined as: $ect_r = eet_p$ and $lct_r = let_p$, and the earliest (latest) expiration time of r (eet_r (let_r)) is calculated as: $eet_r = ect_r$ + r.expiration_duration ($let_r = lct_r$ + r.expiration_duration).

For the example on Fig.1a process Order_intake starts immediately after the zero-time process BEGIN therefore $est_{Order\_intake} = lst_{Order\_intake} = 0$ (the first time point of the execution of the workflow). As it was mentioned earlier Order_intake.min_duration = 1, Order_intake.max_duration = 2. Therefore $eet_{Order\_intake} = 1$ and $let_{Order\_intake} = 2$. If we model the new order as a resource (data) produced by this process, then $ect_{order} = eet_{Order\_intake} = 1$ and $lct_{order} = let_{Order\_intake} = 2$. If, for simplicity, we assume that the order will only be valid for 8 hours then order.expiration_duration = 8h, therefore $eet_{order} = 9$ and $let_{order} = 10$.

Synchronization relations defined in a specification may influence the starting time of processes in this specification. Moreover, some sequencing/branching/cycle relations of the specification may be in conflict with synchronization relations introduced by the designer. Let us define $[t1, t2] = [est_p, lst_p] \cap [est_s, lst_s]$ and $[t3, t4] = [eet_p, let_p] \cap [eet_s, let_s]$ for processes p and s. A *conflict* occurs in following cases: (a) if starts_with(p, s) is introduced and $[t1, t2] = \varnothing$; (b) if starts_during(p, s) is introduced and $[t1, t2] = \varnothing$; (c) if finishes_with(p, s) is introduced and $[t3, t4] = \varnothing$.

In the workflow in Fig.1b process d1 should start at the same time as process d2, starts_with(d1, d2). Let us assume that l1.min_duration = 1h, l1.max_duration = 2h, l2.min_duration = 3h, l2.max_duration = 4h and no delays are modelled in the relevant part of this workflow. Then, for d1 and d2, $[t1, t2] = [est_{d1}, lst_{d1}] \cap [est_{d2}, lst_{d2}] = [1, 2] \cap [3, 4] = \varnothing$ which indicates a conflict. If however it was modelled that l2.min_duration = 2h then $[t1, t2] = [1, 2] \cap [2, 4] \neq \varnothing$ which is a temporally correct assignment.

Definition 6 (A temporally correct workflow): A workflow <w, P, C> is temporally correct in the specification M if the set of ordering relations in M is not conflicting.

If a workflow is temporally correct, starting points of processes influenced by the introduced synchronization relation are updated, using the values t1, t2, t3 and t4 defined above: (a) in case of starts_with(p, s) assign est_p=t1; est_s =t1; lst_p =t2; lst_s=t2; (b) in case of starts_during(p, s) assign est_p =t1 and lst_p=t2; (c) in case of finishes_with(p, s) assign eet_p=t3; eet_s=t3; let_s=t4; let_s=t4. Then, update the values of the earliest (latest) starting points for the processes reachable from p and for the processes reachable from s.

Condition correctness

Definition 7 (A correct condition): A condition of the or-/loop-structure is *correct* iff the following two constraints are satisfied:

A condition of the or-/loop-structure is *correct* iff:

a) all values of condition variable(s) considered in the structure belong to the domain of this (these) variable(s);

b) all elements from the domain of condition variable(s) are taken into consideration in the structure.

Values of condition variables considered in or- and loop-structures are identified by the following procedure:

a) for or-structures if a condition is expressed by one condition variable, then the condition values are obtained from the corresponding or_branch predicates; if one of the identified values is equal to OTHER, then the domain of the condition variable should contain at least one more element different from other extracted values.

b) if the condition expression is complex, i.e. built using the Boolean connectives, then it is divided into basic expressions in form *condition variable OP value*, where OP∈ {=, ≠, <, >} and each basic expression is analysed as follows:

1. If OP is '=' or '≠', then the domain of the condition variable should contain both the *value* and at least one more value different from *value*.
2. If OP is '>' or '<', then the domain of the condition variable should contain at least one element greater than *value* and at least one element smaller than *value*.

If these conditions are satisfied then the whole complex expression is checked whether it is not always true and not always false. This is performed as follows:

1. For each conjunction: if more than one expression on the same variable is present then the intersection of the sets of domain values defined by these expressions contains at least one value;
2. For each disjunction of conjunctions containing expressions over the same variables: at least one of the intersections of the sets of values defined by the expressions on the same variables should be non-empty.

Tasks and resource inter-level consistency constraints

Tasks form hierarchies based on decomposition relations between them. When building such hierarchies, consistency should be maintained by making sure the set of inter-level constraints is satisfied. Here only some examples of inter-level constraints are given (for the complete list of these and other types of constraints (the reader is referred to Popova and Sharpanskykh 2007e):

'For every and-decomposition of a task, the minimal duration of the task is at least the maximal of all minimal durations of its subtasks'.

Formally, ∀ t:TASK, t1:TASK, L:TASK_LIST is_decomposed_to(t, L) ∧ is_in_task_list(t1, L) ⇒ t.min_duration ≥ t1.min_duration

'If a task uses certain resource type as input then there exists at least one subtask in at least one and-decomposition of this task that uses this resource type.'

'For every and-decomposition of a task, if one subtask uses a resource type as input which in not an input for the composite task then there exists another subtask that produces such resource type.'

'If a task produces certain resource type as output then there exists at least one subtask in at least one and-decomposition of this task that produces this resource type.'

'For every and-decomposition of a task, if one subtask produces a resource type as output which in not an output for the composite task then there exists another subtask that uses such resource type as input.'

Functionally divisible resource or data types also form hierarchies. Due to the wide variety of possible situations, only one consistency constraint can be formulated, which should be satisfied for data types:

'If data type dt2 is a functional part of data type dt1, then the expiration duration of dt1 is at most the expiration duration of dt2.'

Physical world generic constraints

Generic constraints come from the physical world irrespective of the application domain. Here several examples of such constraints are given.

GC1: 'No role executes more than one process at the same time'

Formally:

$$\forall\ r{:}ROLE,\ p1,\ p2{:}PROCESS,\ tp1,\ tp2,\ tp3,\ tp4{:}TIME\_POINT\ role\_performs\_process(r, p1) \wedge role\_performs\_process(r, p2) \wedge est_{p1} = tp1 \wedge let_{p1} = tp2 \wedge est_{p2} = tp3 \wedge let_{p2} = tp4 \Rightarrow ((tp2 < tp3) \vee (tp4 < tp1))$$

GC2: 'Not consumed resources become available after all processes are finished'

GC3: 'For every process that uses certain amount of a resource of some type as input, without consuming it, either at least that amount of resource of this type is available or can be shared with another process at every time point during the possible execution of the process'

GC4: 'Non-sharable resources cannot be used by more than one task at the same time'

GC5: 'For all resource types, if a resource of this type is used by a process then the process starts before all resource instances of this type expire'

4.2 Domain-specific constraints

Domain-specific constraints are imposed by the application domain in which the specific specification will be used and can be classified according to their sources.

Constraints imposed by the organisation have been chosen (e.g. by the management of the company) as necessary and need to be satisfied by any specification for the particular organisation. Such constraints can often be found in company policy documents, internal procedures descriptions, etc. For example:

'At every time point, the amount of available resources is at least a pre-specified minimum amount and/or is at most a pre-specified maximum amount.' (company policy on minimal and maximal amount of resource necessary to store)

'The duration of the execution of the workflow should not exceed a specified maximum duration.'

'Certain information types cannot be used by certain tasks.' (security/privacy)

'If at some time point the amount of a resource at a certain location is below a pre-specified minimal amount then within a pre-specified time interval this amount will become more than the minimal amount'. (company policy on replenishing a resource on time)

Constraints coming from external parties are enforced by an external party such as the society or the government and can contain rules about working hours, safety procedures, emissions, and so on. Sources for such constraints can be laws, regulations, agreements, etc. Examples of specific constraints of this group that might be relevant in some situations can be:

'Specific type of information should be used within a pre-specified time interval after it is created.' (e.g. news items should be communicated only when they are recent)

'A driver should not drive more than 6 hours per day.'

Constraints of the physical world come from the physical world with respect to the specific application domain and should be satisfied by any specification in this domain. This is in contrast to the generic physical constraints which should be satisfied by any specification irrespective of the application domain. For example, 'there is always a break of at least 15 minutes between two consecutive lectures' (follows from the limitation of most humans to stay concentrated on a lecture for a very long time) or 'a location cannot store more than a certain prespecified amount of resource' (storage capacity).

For all these types of constraints there are predefined templates, which can be selected and customized by the designer by assigning specific values to the parameters of the template without the need to express logical formulae. Examples of such templates with their parameters in brackets are:

DC1(p1:PROCESS, p2:PROCESS, d:VALUE): 'If the same agent executes both processes, then there is a delay of duration at least d between the end of the first process and the beginning of the second one' (can also be formulated for a specific agent as additional parameter)

Formally:

$$\forall\ r1, r2{:}ROLE, tp1, tp2, tp3, tp4{:}TIME\_POINT: (\exists a{:}AGENT: (agent\_plays\_role(a,r1) \wedge agent\_plays\_role(a,r2) \wedge role\_performs\_process(r1,p1) \wedge role\_performs\_process(r2,p2) \wedge est_{p1} = tp1 \wedge let_{p1} = tp2 \wedge est_{p2} = tp3 \wedge let_{p2} = tp4) \Rightarrow ((tp2 < tp3) \wedge (tp3 - tp2 \geq d)) \vee ((tp4 < tp1) \wedge (tp1 - tp4 \geq d)))$$

DC2(rt:RESOURCE_TYPE, min_am:VALUE): 'At every time point the amount of resource of type rt available is at least min_am amount'

DC3(t:TASK, dr:VALUE): 'For every agent performing processes of this task the sum of the durations of these processes should not exceed dr'

DC4(rt:RESOURCE_TYPE, min_am:VALUE): 'At the end of the workflow there is at least min_am amount of resource of type rt'

DC5(d:VALUE): 'The duration between the first and the last tasks is at most d'

DC6(rt:RESOURCE_TYPE): 'Data of type rt is created by a task during the workflow'

DC7(rt:RESOURCE_TYPE, t:TASK): 'Resource of type rt cannot be used by task t'

DC8(p1, p2:PROCESS): ‘No agent can play roles that perform p1 and p2’

DC9(a:AGENT, rt:RESOURCE_TYPE): ‘Agent a has no access to resource of type rt’

DC10(r:ROLE, rt:RESOURCE_TYPE): ‘Role r has no access to resource of type rt’

DC11(rt1, rt2:RESOURCE_TYPE): ‘Resources of types rt1 and rt2 can not be used together for the same task’

DC12(t:TASK, dr:VALUE): ‘For every agent performing processes of this task the sum of the durations of these processes should not exceed dr’

DC13(l:LOCATION, rt:RESOURCE_TYPE, m:VALUE): ‘At every time point the amount of resource of type rt at location l is at most m’

DC14(l:LOCATION, rt:RESOURCE_TYPE, m:VALUE, dr:VALUE): At every time point t, if the amount of rt at location l is less than m then there exists a future time point t1≤ t + dr at which the amount of rt at l is at least m’

DC15(ag_list: AGENT_LIST, dr:VALUE): ‘The amount of working hours of each agent from the ag_list should not exceed dr’

DC16(a: AGENT, t: TASK): ‘Agent a should not be allocated to task t’

For the case study, a number of relevant domain-specific constraints can be formulated. The first example is a special case of the constraint DC15, which is imposed by the labour legislation (a constraint coming from an external party):

DC15(DRIVERS, 6): The amount of driving hours for each agent driver should not exceed 6 hours per day.

Another agent-related constraint is a special case of the constraint DC16, comes from the organisation and prevents assigning some drivers for certain types of deliveries (e.g. because of the employee’s preferences or some geographical factors):

DC16(ag5, DeliveryAB): The agent ag5 should not be allocated to any delivery of type deliveryAB.

One more constraint imposed by the organisation describes the resource integrity before and after each delivery:

DC17: For each delivery the amount of resource being loaded on a truck should be equal to the resource amount being unloaded from the truck after the delivery has finished.

Another constraint comes from the physical world and describes the conditions of loading of resources of type rt2 on a truck of type tr2 determined by the weight limitations:

DC18(tr2, rt2, 0.7): Maximum 0.7 capacity units of a truck of type tr2 can be used for a delivery of the resource type rt2, whereas 0.3 remaining units should be left empty.

For shared deliveries of resources of type rt2 with some other resource types, special regulations are formulated as constraints.

The following constraint comes from the customer - Company 1 and is determined by its technological process:

DC19: The difference between the finishing time points of the deliveries d1 and d4 should be less than 2 hours.

The last example identifies the constraint that comes from the external parties (Company 1 and its customers):

DC20: For all time points the amount of resources of type rt5 at the base A should be greater than 25.

5 Correctness verification of a process-oriented specification

The verification of the correctness of a process-oriented specification is performed during or at the end of the design of the specification, depending on the verified types of constraints. In particular, some domain-specific constraints might not (yet) be satisfied for incomplete specifications. The designer can choose the moment when they should be checked. The syntactical check of the specification for a specification and the verification of generic constraints are performed at every design step. Note that often only the set of relevant generic constraints is verified. This set is identified based on the type of the change made by the designer in the specification. For example, if the minimal or maximal duration of a task or a decomposition relation between tasks is changed, then the corresponding task inter-level consistency constraint(s) expressed over these tasks should be checked. Changes in resource structures are dealt with similarly. If the designer changes the set of ordering relations or the (minimum or maximum) duration of a task of which an existing process is an instance, then first the structural constraints of the workflow are checked, and after that physical world and domain specific constraints.

For all other types of changes, the set of constraints that should be checked is formed from physical world and domain specific constrains from T_{PR} expressed over objects involved into relations affected by the change. For the checking, it is assumed that each process p in the workflow $<w, P, C>$ can be active (executed) at any time point during the interval $[est_p, let_p]$. Therefore, it will be checked with respect to the whole interval $[est_p, let_p]$, even though the actual execution of p may take less time. Thus all possible intervals of the execution of p are taken into account. If the constraint is not satisfied in some possible execution, it can be discovered without checking all executions separately. This dedicated verification is computationally much cheaper than the general-purpose state-based analysis of all possible executions of a specification (e.g. by model checking, i.e., by trying one by one all possible combinations of durations of processes (Clarke, Grumberg and Peled. 2000)) however still allows establishing correctness of the model.

Here three example algorithms are given for checking the satisfaction of constraints GC1, DC1 and GC3 defined in Section 4. The first and second algorithms are typical for the verification of constraints over processes, roles and agents, and the third one illustrates the verification of constraints over resource amounts related to processes.

Algorithm for verification of GC1

1. Let L be an empty queue and N be an empty set. Enqueue in L all roles defined in the specification.
2. Until L is empty perform steps 3-5.
3. Dequeue L and assign the obtained value to the variable curr_role.
4. Put into N all processes assigned to curr_role in the specification.
5. For each processes $p1,p2 \in N$ determine if they can be executed at the same time:
   if $\neg((let_{p1} \leq est_{p2}) \vee (let_{p2} \leq est_{p1}))$, then GC1 is not satisfied, exit; else empty N
6. GC1 is satisfied.

The proof of the correctness of this algorithm is straightforward. The algorithm processes one by one all pairs of the processes allocated to a role and identifies if the processes in each pair can be executed at the same time. Two arbitrary chosen processes p1 and p2 cannot be executed simultaneously when either $let_{p1} < est_{p2}$ or $let_{p2} < est_{p1}$. Both these conditions are checked at step 5. Thus, if there is a possibility for some of the processes allocated to a role can be executed at the same time, it will be discovered by the algorithm.

The time complexity of this algorithm is estimated as follows: The internal cycle that includes the steps 3-5 is performed |ROLE| times, where ROLE is the set of all roles defined in the organizational specification. In the worst case, all executions of the step 4 together take $|CS| \cdot |ROLE| + |P|$ time units, assuming that each process of the workflow <W, P, C> is allocated to one role and CS is the complete set of constraints defined for the organization. The execution time of the step 5 is calculated as $|P|!/2 \cdot (|P|-2)!$ or $(|P|^2-|P|)/2$. Thus, the overall time complexity of the algorithm is estimated as $O(|ROLE| \cdot (|P|^2 + |CS|))$.

Algorithm for verification of DC1

1. Let L and N be empty sets. Put into N all roles allocated to A by instantiated constraints in CS'.
2. Put into L all processes assigned to roles in N in the specification.
3. If $p1 \notin N$ or $p2 \notin N$, then DC1 is satisfied, exit.
4. Determine if there is a delay of duration at least d between the processes p1 and p2:
   if $(p1.est>p2.est \wedge p1.let+d \leq p2.est) \vee (p2.est>p1.est \wedge p2.let+d \leq p1.est)$, then DC1 is satisfied;
. otherwise DC1 is not satisfied.

The proof of the correctness of the algorithm is straightforward and follows directly from the structure of the formula representing the constraint.

The time complexity of the algorithms is estimated as O(|CS|), where CS is the complete set of constraints defined for the organization.

Before describing an algorithm for checking GC3 let us introduce a definition of a workflow segment and a labelling procedure for workflow segments.

Definition 7 (A workflow segment): A segment SG of the workflow <W, P, C> is a set of processes from P ordered by C that are executed under the same set of values of

or-conditions from w. This set of values is dynamically formed from the values of conditions of or-structures, from which the processes of SG can be reached. The set SEGMENTS contains all segments of the workflow.

Each segment has a label, assigned according to the following rules:

- the segment that contains processes that are executed independent of any condition values has the label '1'.
- the label for a segment that corresponds to a branch of a certain or-structure is formed from three parts that follow each other:
  (1) the prefix defined by the label L of the segment, to which the beginning process of the or-structure belongs;
  (2) the index of the branch in the or-structure obtained incrementally starting from 1;
  (3) the sequential index of the or-structure in the segment with the label L, put in square brackets.

For example, the process Order_intake from the example introduced in Section 2 belongs to the segment labelled by '1', whereas the process Order_delivery2 belongs to the segment labelled by 1.2[1].

Further the algorithm is given for checking the satisfaction of GC3 with respect to the process p and the resource r. In this algorithm the following notations are used:

res_produced_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_produces(t, r, am)
res_used_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_uses(t, r, am)
res_consumed_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_consumes(t, r, am)

Algorithm for verification of GC3

1. Identify the set of time points TP within the duration of p ($est_p \leq t < let_p$), at which the amount of some resource(s) of type r changes (i.e. time points at which other processes that use/consume/produce a resource of type r may start or finish). For every time point $t \in TP$ perform steps 2-7.
2. Determine the set RS of segments that contain finished before or executed simultaneously with p processes, which execution may influence the amount of resources of type r:

   $RS = \{s \in SEGMENTS \mid \exists a\ a \in s \wedge [let_a < t \wedge [\ am1 > 0 \vee am3 > 0]\ ] \vee [\ let_a > t \wedge est_a \leq t \wedge [\ am1 > 0 \vee am2 > 0 \vee am3 > 0\ ]$, where am1, am2, and am3 are specified in res_consumed_by(r, a, am1), res_used_by(r, a, am2) and res_produced_by(r, a, am3).

3. The labels of segments in RS that correspond to the branches belonging to the same or-structure are grouped.
4. The n-ary Cartesian product of all obtained groups is generated (n is the number of groups): $g_1 \times ... \times g_n$. Each tuple in the obtained product set corresponds to a possible combination of segments in the workflow. In such a way all possible execution of processes in the workflow, which use/produce/consume r and have latest ending time $\leq let_p$ are considered.
5. For every tuple in the product set identify the set of processes PS that corresponds to the tuple. If two or more processes from the same segment related by a sequencing relation may be executed at the same time in different instances of the workflow, replace PS by a number of sets, each of which will

contain only one from these processes.

6. For every set of processes PS corresponding to the tuple, identify the set of resources RPS of type r produced by processes in PS.
7. For every process a ∈ PS that consumes some amount of the resource of type r identify if this amount of not expired resource(s) from RPS is available. Update RPS after every iteration:
   7.1 Initial settings: Let temp_amount = am, where am is defined by res_consumed_by(r, a, am)
   7.2 Until temp_amount > 0 and RPS is not empty perform 7.3 and 7.4
   7.3 Identify resource res ∈ RPS with the smallest earliest expiration time, which did not expire yet. It is assumed that such resource will be used first by a.
   7.4 If res.amount ≥ temp_amount,
   then update the amount of the resource as res.amount = res.amount - temp_amount and set temp_amount = 0.
   else update temp_amount = temp_amount - res.amount and delete res from the RPS.
   7.5 If temp_amount > 0,
   then **GC3 is not satisfied** with respect to the process p and resource type r, exit.
8. For every process a ∈ PS that uses a certain amount of the resource of type r at time point t identify, if this amount of not expired resource(s) from RPS is available. Update RPS after every iteration:
   8.1 Initial settings: Let temp_amount= 0.
   8.2 From the specification identify process lists that may share a resource of type r and that contain as least one process from PS. For each process a ∈ PS at most one list will be chosen from the identified lists. Furthermore, any list that contains a may be selected, since the choice of the list does not influence the amount of available resources in RPS.
   8.3 For every chosen process list L update temp_amount = temp_amount + am, where am is defined in res_used_by(r, s, am) and s is some process from L. Delete all processes that belong to L from PS.
   8.4 For every process a ∈ PS update temp_amount = temp_amount + am, where am is defined in res_used(r, a, am). Then perform the same calculations as on steps 7.2-7.5.
   9. **GP3 is satisfied** with respect to the process p and resource type r.

The informal explanation and the proof of correctness of the algorithm

For the following explanation the notation avail_res_amount (t, r, am) will be used, meaning that at the time point t the total amount of available resources (i.e., not used and not consumed) of the type r equals am.

To check the satisfaction of the constraint GP3 for process p that requires the amount am_req of the resource type r, it is needed to verify that:

(1) $avail\_res\_amount(est_p, r, am) \wedge am \geq am\_req$ (i.e., the available resource amount of type r is sufficient for the execution of the process p) or $\exists p1{:}PROCESS\ \exists l{:}PROCESS\_LIST\ est_p > est_{p1} \wedge let_{p1} \geq est_p \wedge resource\_sharable(r, l) \wedge is\_in\_list(p1, l) \wedge is\_in\_list(p, l)$ (meaning that there exists another process p1 being executed at est_p, with which p may share the resource type r)

(2) for all $t \in (est_p, let_p]$, at which the amount of available resources of the type r changes: avail_res_amount (t, r, am) $\wedge$ am $\geq$ am_req or $\exists$p1:PROCESS $\exists$l:PROCESS_LIST t > $est_{p1} \wedge let_{p1} \geq t \wedge$ resource_sharable(r, l) $\wedge$ is_in_list(p1, l) $\wedge$ is_in_list(p, l)

In the following also this notation is used:
res_produced_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_produces(t, r, am)
res_used_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_uses(t, r, am)
res_consumed_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_consumes(t, r, am)

The amount of the resource type r in (2) may change due to the following events: (a) beginning of some process that uses/consumes the resource amount amt of type r, in this case if no sharing possibilities exist for the process, then avail_res_amount (t_b, r, am - amt), where t_b is the beginning time point of the process and am is the available resource amount of type r at the time point before t_b; (b) finishing of some process that uses the resource amount amt1 or produces the resource amount amt2 of the type r, in this case avail_res_amount (t_e, r, am + amt1) if no other processes share the resource being used or avail_res_amount (t_b, r, am + amt2); here t_e is the finishing time point of the process and am is the available resource amount of type r at the time point before t_e. In the proposed modelling framework it is assumed that during the execution of a process the amount of the resources that it uses does not change. Therefore, the set of time points TP identified at the first step of the algorithm is defined as: $\{t \in$ TIME | $\exists$p1: PROCESS $\exists$am1, am2, am3:VALUE (res_produced_by(r, p1, am1) $\vee$ res_produced_by(r, p1, am2) $\vee$ res_produced_by(r, p1, am3)) $\wedge$ $est_p \leq est_{p1} \leq let_p \wedge t = est_{p1}\} \cup \{t \in$ TIME | $\exists$p1: PROCESS $\exists$am1, am2, am3:VALUE (res_produced_by(r, p1, am1) $\vee$ res_produced_by(r, p1, am2) $\vee$ res_produced_by(r, p1, am3)) $\wedge$ $est_p \leq let_{p1} \leq let_p \wedge t = let_{p1}\}$.

For all $t \in$ TP the value am in avail_res_amount (t, r, am) is calculated based on the amounts of resources of the type r produced and consumed before t, and based on the resource amounts of type r consumed, produced and used at t. The sets of processes that produce, consume and use these resources may be different for different executions of the workflow. Alternative execution paths of the workflow are formed from different executions of the or-structures of the workflow. Furthermore, processes that form these paths may have different duration in different executions. To guarantee the satisfaction of the constraints (1) and (2) for every possible execution of the process p, all execution paths of the workflow that differ in the resource amount am in avail_res_amount (t, r, am) at least at one time point $t \in$ TP, should be checked.

First such paths should be identified. This is performed by the steps 2-6 of the algorithm. An important, though obvious, observation here is that different branches of the same or-structure cannot be executed at the same time, whereas different branches of different or-structures often can be executed simultaneously in various combinations, thus forming different execution paths of the workflow. To identify these paths all branches of the or-structures of the workflow that form segments containing finished before or executed simultaneously with p processes, which execution may influence the amount of resources of type r, are labelled using the procedure described above. Further, the introduced labels of the segments are put into the set RS (step 2). Note that a segment may contain (nested) and-structures. The processes of segments, with labels that have the same prefix and the same sequential index cannot be executed simultaneously. Such segments are combined into groups at step 3. Thus, each group contains alternative partial execution paths. Then, the

Cartesian product of all obtained groups is determined at step 4, thus, defining all possible execution paths of the workflow for the time period $[0, let_p]$ that contain processes that use and/or consume and/or produce resources of type r.

At step 5 of the algorithm for each tuple in the product the set of processes PS is identified that belong to the segments in the tuple and that use and/or consume and/or produce resources of type r. If two or more processes from the same segment related by a sequencing relation may be executed at the same time in different instances of the workflow, i.e., for processes p1 and p2: $[est_{p1}, let_{p1}] \cap [est_{p2}, let_{p2}] \neq \varnothing$ PS is replaced by a number of sets, each of which will contain only one from these processes. This is needed because at most one from the processes related by a sequencing relation can be executed at any time point in any instance of the workflow.

Then, each execution path is processed separately. For each path the set of resources of the type r produced by the processes of the path during the interval $[0, let_p]$ is determined (step 6). Then, for each process of the path that consumes some amount amt of the resource type r, the value of the available resources(s) of the type r with the earliest expiration time decreases by amt at the earliest starting time point of the process (step 7). If no such resource(s) is (are) available, the constraint is not satisfied. After that it is checked if the remaining resource amounts related to time points suffice for the execution of the processes of the path that use some amounts of the resource type r and which execution interval has a non-empty intersection with the interval $[est_p, let_p]$ (step 8). Now let us consider the steps 6-8 in detail.

First, at step 6 the set RPS of resources of the type r produced by the processes of the path is identified. These resources are used and consumed by other processes of the path. Each resource is allowed to be used or consumed before its expiration. Furthermore, it is assumed that resources with the earliest expiration time will be used or consumed first by processes of the path.

Then, at step 7 for each process of the path that consumes some amount of the resource type r it is identified if this amount of not expired resource(s) from RPS is available. Each time when a process that consumes the amount amt of the resource type r is identified, the amount of the resource r1 with the earliest expiration duration decreases: r1.amount = r1.amount - amt. In case amt > r1.amount, then the difference amount amt-r1.amount is taken from another resource(s) that has (have) the earliest expiration time after r. If no such resource is available, the constraint GP3 is not satisfied.

Finally, at step 8 for all time points $t \in TP$ for every process of the path that uses a certain amount of the resource type r it is identified if this amount of not expired resource(s) from RPS is available. Note that only the processes being executed at time points $t \in TP$ should be checked, since all processes that finished before est_p also released the resources that they had used. Note that some processes from PS may share the same resource amount of the type r if they belong to the same process list l such that resource_sharable(r, l). For each process $a \in PS$ at most one list is chosen. Furthermore, any list that contains a may be selected, since the choice of the list does not influence the amount of available resources in RPS. When the total used resource amount is calculated at each time point $t \in TP$, the resource amount of the type r shared by a list of processes being executed at t is taken to be equal to the resource amount of the type r used by any process from this list (step 8.3). To the obtained amount is added the sum of all amounts of resources of type r used by the processes from PS

being executed at t that cannot share resource type r (step 8.4). Then, it is determined if the obtained total used resource amount does not exceed the total available amount of resources of type r at the time point t calculated by the execution of the steps 7 and 8. If it does not exceed, the constraint GC3 is satisfied, otherwise, GC3 is not satisfied.

In order to reduce the number of tuples generated at step 4 and to improve the computational properties, the introduced verification algorithm is extended with tuple reduction steps (the extended version of the algorithm can be found in (Popova and Sharpanskykh 2007e)). This algorithm performs the local elimination of segments within each group that do not contribute to the worst case situation. More specifically, in each group all segments are eliminated, except for the segment that uses the largest amount of resources of type r. This allows reducing the number of execution paths that have to be checked significantly.

Some discussions about the time complexity for the considered algorithm are given below.

The execution of the step 1 takes $O(|P|)$, where P is the set of processes of the workflow $<w, P, C>$. The execution time of the step 2 depends on the number of or-structures of the workflow and the number of branches in each or-structure and takes $O(b)$, b is the overall number of all branches in all or-structures of the workflow. The execution of the step 3 takes also not more than $O(b)$. The time complexity of the execution of the step 4 is dependant on the number of or-structures of the workflow, on the number of branches in each or-structure and on the level of nesting of the or-structures. For the worst case, in a workflow that contains k or-structures with the overall number of branches b, the execution of step 4 takes $(b/k)^k$ time points. For workflows that have nested or-structures, the execution time required for the step 4 is less than $(b/k)^k$, because of workflows with such structures have a smaller number of segments. The steps 5-9 may be repeated $(b/k)^k$ times in the worst case. Each execution of the step 5 in the worst case may take $O(|P|\cdot|C|)$. Each execution of the step 6 in the worst case take not more than $O(|P|)$. Each execution of the steps 7 and 8 may take not more than $O(|P|^2)$. Thus, from the performed complexity analysis it follows that the time complexity of the proposed algorithm is polynomial in the number of the processes and the ordering constraints of a workflow, however exponential in the number of or-structures and segments formed based on these or-structures. An approach to decrease the number of segments, and thus, to decrease the complexity of the proposed algorithm significantly, is briefly described above; more specific details of this extension can be found in (Popova and Sharpanskykh 2007e).

The proposed verification algorithm is computationally much cheaper than standard model checking procedures, which time complexity for analysing specifications in the proposed process-oriented language would be exponential, not only in the number of or-structures and in the number of branches of these structures, but also in the number of the processes in these branches. Furthermore, during model checking, the set of time points TP' for which the satisfaction of the constraint GC3 for the process p should be checked always includes every time point from the interval $[est_p, let_p]$. The set TP' contains in most cases much more elements in comparison to the set TP, obtained by an execution of the proposed algorithm.

6 Related literature

Different aspects of process-oriented modelling and analysis have been investigated in different areas, such as enterprise modelling, artificial intelligence, operations research and others. The following aspects of the process-oriented modelling are usually considered in these areas: functional, behavioural, information-, resource- and organisation-related. Let us briefly discuss each of these aspects.

The functional aspect is usually represented by static task structures (Fox 1992, Menzel and Mayer 1998, Malone, Crowston and Herman 2003), in which characteristics of tasks (activities) (such as input, output and function) and relations between them are defined. Task structures usually serve as templates for process execution structures. To reduce the complexity and to provide means for process-oriented modelling at different levels of abstraction, tasks are structured in hierarchies built on refinement relations as in (Fox 1992, Malone, Crowston and Herman 2003) and in the framework proposed here. Furthermore, in the approach proposed here, special verification means based on constraints are provided, in order to guarantee the correctness of built hierarchical structures, which are absent in other mentioned frameworks.

The behavioural aspect is realized by process execution structures, which are often called control flows. Currently a great variety of languages and frameworks for process-oriented modelling exist. Some of the proposed languages are purely graphical (Menzel and Mayer 1998, Yang and Zhang 2003), whereas others have formal foundations (Fox 1992, van der Aalst 1998). Although process-oriented languages differ in their specification means and expressivity, many of them realize similar control patterns of process execution (or of workflows). In (van der Aalst *et al*. 2003) an extensive overview and a classification of different types of workflow patterns is presented. The graphical process specification languages such as BPMN, BPML, UEML, YAWL (van der Aalst and ter Hofstede 2005) realize these templates to a different extent. Also the process-oriented language presented in this paper supports the most essential and commonly used templates, which are identified in the introduction of the language. Furthermore, more specific templates described in (van der Aalst *et al*. 2003) not addressed in this paper can be easily implemented by an extension of the introduced language. The proposed language allows temporal numerical expressivity for the specification of control flows (e.g. durations of processes and delays, real time constraints), which is deficient in a number of other frameworks (e.g. BPMN, CIMOSA (1993), IDEF3 (Menzel and Mayer 1998)). Furthermore, many of the existing process modelling languages are not formally grounded, and, therefore, can not be used for formal analysis. Although some frameworks propose automated techniques for analysis of process-oriented specification even without properly defined semantics of the modelling language, still the results of such analysis are not completely reliable. Furthermore, the behaviour of such process-oriented specifications may be unpredictable. For example, in the ARIS framework (Scheer and Nuettgens 2000) the control flows are modelled using informal Event-driven Process Chains (EPCs), which limits the possibilities for analysis and its reliability. Similar observations can be made with respect to the frameworks described in (CIMOSA 1993, Yang and Zhang 2003).

However, also a number of formal methods have been applied for modelling and analysing of control flows: process algebra, Petri nets and their extensions and modifications (such as Workflow Nets), and different types of logics.

In (Singh 1996) process algebra is used to represent ordering constraints on processes, however it lacks the expressivity to represent global constraints on processes and (real) numbers (i.e. durations).

Petri-nets and their modifications have been extensively used for formal modelling and analysis of workflows (van der Aalst 1998). This formalism is useful for specifying ordering constraints, however it is difficult to express global constraints over multiple objects, characteristics and relations of the organisation (e.g. many physical world and domain-specific constraints considered in this paper) using Petri Nets. For example, the constraint 'the breaks between the processes allocated to some role should be at least one hour'. In (Adam, Atluri and Huang 1998) it is shown that one can manually construct a Petri net that satisfies a certain set of global constraints, however, then such a representation does not include the information about the constraints themselves. Furthermore, Petri Nets are difficult in use for non-professionals, whereas the introduced approach proposes an intuitive, close to the natural, predicate language, which can be represented graphically.

Different types of logics have been used for modelling and analysis of control flows. One of them is the propositional temporal logic (Attie *et al.* 1993). Although temporal logic is highly suitable for specifying ordering constraints, it has a number of expressivity limitations, e.g. numbers cannot be expressed, in most cases variables and composite structures (such as predicates) cannot be used. Furthermore, most of the existing general-purpose algorithms for checking properties expressed as temporal logic formulae on flow specifications (e.g. model checking (Clarke, Grumberg and Peled. 2000)) have a high computational cost.

The first-order predicate logic has been used for designing ontologically rich process-oriented specifications in (Fox 1992). However, analysis issues of such specifications are not addressed. Different variations of transaction logics (Bonner and Kifer 1998) have been applied for modelling, executing (scheduling) and analysing control flows. Originally, the transition logic has been developed as an extension of the first-order logic for the representation of state changes in databases and logical programs. Therefore, although it allows designing correct flows and performing effective analysis, it still lacks the ontological expressivity to represent the variety of objects and relations that exist in organisations.

A number of dedicated formal techniques have been developed for checking temporal constraints on processes in workflows (Bettini, Wang, and Jajodia 2002, Lu *et al.* 2006).

Information- and resource-related aspects are modelled in a number of informal and semi-formal frameworks as separate flows and in relation to processes (BPML, BPMN, Fox 1992, Barkaoui and Petrucci 1998, Menzel and Mayer 1998, Yang and Chen 2004). In particular, a number of workflow resource patterns are introduced in (Russell *et al.* 2004) that aim to capture the various ways in which resources are represented and utilized in workflows. Whereas these patterns provide an aggregated view on the resource allocation that includes authority-related aspects and the characteristics of roles, the framework proposed in this paper distinguishes different types of organisational aspects into separate views and establishes relations between

these views. Thus, the models of resources and their relations to tasks and processes are specified separately from the role- and authority-related aspects. However, if needed, particular domain-specific constraints can be specified that are based on information from different organisational views to describe different modes of creation/use of resources.

Furthermore, not many frameworks address the verification aspects of resource-based models. Often in formal analysis only a very limited number of aspects of resources and information related to process-oriented models are addressed (Barkaoui and Petrucci 1998, Li, Yang and Chen 2004). In the proposed framework resources are characterized by a type, an expiration time, an amount that may be used, consumed or produced by a process. Furthermore, a process may share a certain amount of some resource with other process(es) and a physical replacement of resources can be specified. Our representation of shared resources is different from (Barkaoui and Petrucci 1998) in three aspects: (1) a certain specified amount of the resource can be shared among processes at the same time; (2) sets of processes that are allowed to share a certain resource can be predefined; (3) different amounts of the same resource can be shared (at the same time). Information is treated as a special kind of a resource. The algorithm for verification of the resource related constraints takes into account all characteristics and modes of use of resources at the same time. To our knowledge there exist no other frameworks that represent and verify all the specified resource characteristics and dependences simultaneously.

Organisational aspects are modelled in many frameworks from the area of artificial intelligence (Horling and Lesser 2005) and enterprise systems (CIMOSA 1993, Bernus et al. 1998, Scheer and Nuettgens 2000). Often such models specify (different types of) relations between tasks (processes) and agents (roles, actors). However, a specification of processes (tasks) and relations between them is often kept simple to enable computationally effective agent-(role-) oriented analysis. The proposed framework establishes relations between concepts from the process-oriented view and the concepts form the organisation-oriented view (e.g. roles and agents), as well as the concepts from the performance-oriented view (e.g. goals, performance-indicators), while keeping the complete ontological expressiveness of each of the views. By doing this different sophisticated methods of analysis across views can be performed.

7 Conclusions

This paper introduces a formal framework for process-oriented modelling and analysis. The framework is based on an expressive sorted predicate logic language L_{PR}, which allows specifying a wide range of concepts and relations of the process-oriented view on organisations. In particular, L_{PR} provides means for the detailed modelling of resources, including different modes of sharing that are distinct from other existing modelling frameworks. Moreover, since the process-oriented view is related to other organisational views, process-oriented specifications may include relations between tasks, processes, resources and other organisational concepts (e.g. roles, goals, agents). Furthermore, L_{PR} is used for the specification of different types of organisational constraints that should be satisfied by process-oriented specifications.

These constraints may express both local (i.e. related to individual objects) and global (i.e. related to multiple objects) properties of an organisation. Also, the paper proposes efficient dedicated analysis techniques for checking the correctness of process-oriented specifications with respect to different sets of constraints, all of which are implemented. The proposed verification algorithms are more (time- and resource-) efficient than general-purpose logical analysis techniques (e.g. model checking and theorem proving), as they do not require checking of properties along all the possible execution paths of process-oriented specifications. To our knowledge there exist no other frameworks that allow the simultaneous verification of different (interdependent) types of constraints based on the extensive set of concepts and relations as can be found in L_{PR}.

The proposed approach differs from constraint satisfaction methods developed in (Tsang 1993). Whereas the main focus of the latter techniques is on finding (optimal) solutions given a consistent and stable set of constraints, our approach addresses both design of a specification and of constraints that should be satisfied by the specification. The designer is free to vary both the specification and the constraint specifications. The designer is supported by the automated tool that allows identifying sources of inconsistencies and mistakes both in the specification and the constraint specifications.

The developed approach allows scalability by performing compositional design of specifications. Using task hierarchies specifications can be built at different levels of abstraction. General constraints defined for high level processes are refined into more specific ones that should be satisfied by processes of lower levels. In such a way, to decrease complexity, specifications of different abstraction levels can be analysed separately keeping relations with each other through task hierarchies and the constraint refinement.

Furthermore, although the introduced predicate language is very intuitive, still a graphical interface for creating and changing specifications would be of help. Such an interface is currently being developed. However, graphics would provide only a little help in the specification of constraints. For this property templates can be used as shown in this paper.

The formal methods discussed in the paper are dedicated for the verification of process-oriented specifications, however, also a number of formal techniques for the analysis of actual execution based on the introduced process-oriented specification, have been developed. These techniques are discussed in (Popova and Sharpanskykh 2007b).

References

Allen, J.F., Maintaining knowledge about temporal intervals. *Communications of the ACM*, 1983, 26, pp. 832–843.

Adam, N.R., Atluri, V., Huang, W.-K., Modeling and analysis of workflows using Petri Nets. *Journal of Intelligent Information Systems*, 1998, 10, pp. 131–158.

Attie, P., Singh, M., Sheth, A., Rusinkiewicz, M., Specifying and enforcing intertask dependencies. In *Proceedings of the 19th VLDB Conference*, 1993.

Barkaoui, K., Petrucci L., Structural analysis of workflow nets with shared resources. In *Workflow Management: Net-based Concepts, Models, Techniques and Tools*, edited by W.M.P. van der Aalst, G. De Michelis, C. A. Ellis, 1998, 98, pp. 82–95.

Bernus, et al. (eds.), *Handbook on architectures of information systems*, Heidelberg, 1998 (Springer-Verlag).

Bettini, C., Wang, X., Jajodia, S., Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 2002, 11(3), pp. 269–306.

Bonner, A. J., Kifer, M., A logic for programming database transactions. In *Logics for Databases and Information Systems*, edited by J. Chomicki, G. Saake, pp. 117–166, 1998 (Kluwer).

Broek, E., Jonker, C., Sharpanskykh, A., Treur, J., and Yolum, P., Formal modeling and analysis of organizations. In: O. Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowsk, J. Sichman and J. Vazquez Salceda (eds.), *Coordination, Organization, Institutions and Norms in Agent Systems I*, LNAI 3913, pp. 18-34, 2006 (Springer)

Business Process Modeling Language (BPML).http://www.bpmi.org.

Business Process Modeling Notation (BPMN) http://www.bpmn.org/

CIMOSA – Open system architecture for CIM, ESPRIT Consortium AMICE, 1993 (Springer-Verlag, Berlin).

Clarke, E.M., Grumberg, O., Peled, D.A., *Model checking*, 2000 (MIT Press).

Cormen, T.H., Leiserson, C. E., Rivest, R. L., Stein, C., Introduction to Algorithms, 2001 (MIT Press)

Fox, M.S., The TOVE project: towards a common-sense model of the enterprise. In *Proceedings of ICIEMT'92*, edited by C.J. Petrie Jr., pp. 310–319, 1992 (MIT Press).

Horling, B., and Lesser, V., A Survey of multi-agent organizational paradigms. The *Knowledge engineering review*, 19(4), pp. 281–316, 2005 (Cambridge University Press).

Li, H., Yang, Y., Chen, T.Y., Resource constraints analysis of workflow specifications. *Journal of Systems and Software*, 2004, 73(2), pp. 271–285.

Lu, R., Sadiq, S., Padmanabhan, V., Governatori, G., Using a temporal constraint network for business process execution. In *Proceedings of 17th Australasian Database Conference*, Australian Computer Science Association, ACS, pp. 157-166, 2006.

Malone, T., Crowston, K., Herman, G. (eds.): *Organizing business knowledge: The MIT Process Handbook*, 2003 (MIT Press, Cambridge, MA).

Manzano, M., *Extensions of First Order Logic*, 1996 (Cambridge University Press).

Menzel, C., Mayer, R.J., The IDEF family of languages. In *Handbook on Architectures of Information Systems*, edited by P. Bernus et al., pp. 209–241, 1998 (Springer-Verlag, Heidelberg).

Popova, V. and Sharpanskykh, A. (2007a), A Formal framework for modeling and analysis of organizations. In *Proceedings of the Situational Method Engineering Conference, ME'07*; edited by Ralyte, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) 2007 (Springer Verlag).

Popova V., and Sharpanskykh, A. (2007b). Formal analysis of executions of organizational scenarios based on process-oriented models. In I. Zelinka, Z. Oplatkova and A. Orsoni (eds.), *Proceedings of 21st European Conference on Modelling and Simulation ECMS*'07, pp. 36-44, 2007 (SCS Press).

Popova, V., and Sharpanskykh, A. (2007c), Formal modelling of goals in agent organizations. In: Proceedings of AOMS workshop joint with IJCAI'07 edited by V. Dignum, F. Dignum, E. Matson, B.Edmonds, 2007, pp.74-86.

Popova, V., and Sharpanskykh, A. (2007d), Modelling organizational performance indicators, In *Proc. of IMSM'07 conference*, edited by F. Barros et al., 2007, pp. 165–170.

Popova, V., and Sharpanskykh, A. (2007e), Process-oriented organization modeling and analysis based on constraints, Technical Report 062911AI, VUA, http://hdl.handle.net/1871/10545.

Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P. Workflow Resource Patterns. BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

Scheer, A.-W., Nuettgens. M., ARIS Architecture and reference models for business process management. In *LNCS* 1806, edited by W.M.P. van der Aalst et al., pp. 366–389, 2000 (Springer-Verlag, Berlin).

Scott, W.R., *Institutions and organizations*. 2nd edn, 2001 (SAGE Publications, Thousand Oaks London New Delhi).

Sharpanskykh, A., Authority and its implementation in enterprise information systems. In *Proceeding of the 1st International Workshop on Management of Enterprise Information Systems, MEIS 2007*, 2007 (INSTICC Press).

Singh, M. P., Synthesizing distributed constrained events from transactional workflow specifications. In *Proc. of the 12th IEEE Intl. Conf. on Data Engineering*, 1996, pp. 616–623.

Tsang, E., *Foundations of Constraint Satisfaction*. 1993 (Academic Press).

Van der Aalst, W. M. P., The application of Petri Nets to workflow management, *The Journal of Circuits, Systems and Computers*, 1998, 8(1), pp. 21–66.

Van der Aalst, W.M.P., and Ter Hofstede, A.H.M., YAWL: Yet another workflow language, *Information Systems* , 2005, 30(4), pp. 245-275.

Van der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., and Barros, A.P., Workflow patterns, *Distributed and Parallel Databases*, 2003, 14(3), pp. 5–51.

Yang, D., and Zhang, S., Modeling workflow process models with statechart. In *Proc. of ECBS*, 2003, pp. 55–61.

Chapter 4

Formal Analysis of Executions of Organizational Scenarios Based on Process-Oriented Models [1]

Abstract. This paper presents formal techniques for analysis of executions of organizational scenarios based on process-oriented models of organizations. A part of these techniques is dedicated to establishing the correspondence between formalized executions (i.e., traces) and process-oriented models. Other techniques provide the analyst with wide possibilities to analyze organizational dynamics and to evaluate organizational performance. For the proposed formal analysis the order-sorted predicate Temporal Trace Language (TTL) is used. The analysis is supported by the dedicated software tool TTL Checker. The analysis approaches are illustrated by a case study in the context of an organization from the security domain.

1 Introduction

Process management in many modern organizations is supported by dedicated software systems, such as Workflow Management Systems (WfMS). WfMSs are used to guide/control the execution of organizational scenarios based on certain internal models. These models describe/prescribe ordering and timing relations on processes, modes of use of resources, allocations of actors to processes etc. WfMS models are expressed using different formalisms: Petri-Nets, Workflow Nets, process algebra, logical specifications. An approach proposed in this paper makes use of models

[1] Part of this chapter appeared as Popova, V., Sharpanskykh, A.: Formal Analysis Of Executions Of Organizational Scenarios Based On Process-Oriented Models. In: I. Zelinka, Z. Oplatkova and A. Orsoni (eds.), Proceedings of 21st European Conference on Modelling and Simulation ECMS 2007, SCS Press, 36-44 (2007) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

specified in an expressive order-sorted predicate language L_{PR} described in (Popova and Sharpanskykh 2006). The actual execution of organizational scenarios may diverge from the dynamics (pre)defined by a process-oriented model. To capture this difference many WfMSs record data about actual executions (e.g., starting and finishing time points of processes, types and amounts of resources used/consumed/produced/broken, names of actors who perform processes).

To guarantee the correct operation of an organization supported by a WfMS (1) a correct formal process-oriented model should be provided and (2) actual executions of organizational scenarios should correspond to this formal model. For establishing the correctness of process-oriented (or workflow) models a number of formal verification techniques exist (e.g., Aalst and Hee 2002) aimed at identifying errors and inconsistencies in models, irrespectively of actual executions of these models. The verification techniques related to models used in this paper are described in (Popova and Sharpanskykh 2006). However, not many formal techniques and tools exist for establishing if the organization actually behaves as it is specified by the model (i.e., for validating a model). In (Barjis et al. 2002; Desel et al. 2003) validation is performed by simulation of organizational scenarios. Although simulation techniques can provide useful insights into relationships and dynamics of an organization, they often abstract from the complexity of dynamics of real organizations. To perform analysis based on the actual organizational execution, data gathered by a WfMS can be used. For example, in (Aalst et al. 2005) it is shown how the analysis based on linear temporal logic (LTL) can be used for establishing the correspondence between the observed and the expected organizational behavior. In this paper different types of formal, automatically supported analysis of actual executions based on process-oriented models will be described. These types include checking the conformity to a formal process-oriented model and to the formal organization, analysis of organizational emergent properties and organizational performance evaluation. The analysis is based on the predicate-based Temporal Trace Language (TTL), which allows more expressivity than LTL used in (Aalst et al 2005).

The presentation is organized as follows. First, the overview of the proposed analysis framework is given. Then, the specification of process-oriented models is briefly discussed and a language used for formalizing executions is introduced. Next, TTL and the dedicated software environment TTL Checker are considered. Finally, different types of trace-based analysis are discussed and illustrated by a case study from the security domain. The paper concludes with a discussion.

2 Trace-based Analysis: Overview

In (Popova and Sharpanskykh 2007a) a general organization modeling and analysis framework is introduced including different views on organizations. In particular, the performance-oriented view describes organizational goal structures, performance indicators structures, and relations between them. Within the organization-oriented view organizational roles, their authority, responsibility and power relations are defined. In the agent-oriented view different types of agents with their capabilities are identified and principles for allocating agents to roles are formulated. Finally,

process-oriented view describes static structures of tasks and resources, the flow of control, and addresses the actual execution of organization processes. The views are related to each other by means of common concepts, which enables different types of analysis across views. This paper describes a part of the process-oriented view related to actual execution of organizational scenarios based on process-oriented models formalized in L_{PR}. Data about actual executions are structured in the form of a *trace* - a formal structure that consists of a time-indexed sequence of states. Each state is characterized by a set of organizational and environmental events that occur in the state. Events are specified by atoms in a sorted predicate language L_{EX}, described in this paper. The formal analysis of actual executions is performed by checking organizational properties expressed in TTL on traces using the TTL Checker tool. The TTL Checker has a graphical interface, using which TTL formulae can be inputted and traces that represent organization executions can be loaded and visualized (see for example Fig.1). The tool generates a positive answer, if the specified property is satisfied by the execution model (i.e., holds w.r.t. the loaded trace(s)). If a formula is not satisfied, a counterexample is provided. The tool also allows performing statistical analysis on multiple traces. More details on the TTL and the tool are given further in the paper. Here we identify the types of trace analysis that can be performed using the TTL Checker.

Each process-oriented model (pre)defines a set of scenarios of organization behavior. The actual execution of an organization may diverge from scenarios described by the model. In some organizations a certain degree of deviation is allowed, whereas other organizations require a strict adherence to the model (e.g., military organizations, nuclear power plants). In the second case the verification of the conformity of an actual execution to a formal organization model is of special importance. This is the first type of analysis considered in this paper.

Every correct process-oriented model guarantees the satisfaction of a set of (global) constraints over processes, resources and agents identified in the organization. These constraints are usually specified based on different organizational and general normative documents (e.g., a strategy description, laws, policies, etc.). In general, if a trace conforms to the corresponding process-oriented model, then all constraints imposed on and satisfied by the model are also satisfied by the trace. However, when the checking of the conformity of the trace to the model fails, then the satisfaction of the constraints by the trace is not guaranteed any more. In this case the analysis of the conformity of a trace to a formal organization (i.e., organizational constraints) should be performed, which is the second type of analysis considered in this paper. Often process-oriented models allow (different degrees of) autonomy of agents in executing organizational scenarios. For example, in many organic organizations processes are defined loosely to ensure flexibility. To analyze the functioning of such organizations, an approach called analysis of the emergent organizational behavior is proposed in the paper.

Finally, the paper proposes a method for the evaluation of organizational performance based on checking the satisfaction of organizational goals related to processes.

The types of analysis described above may be performed both during the execution and after the execution of organizational scenarios.

3 The Specification of the Process-Oriented Model

Process-oriented models are expressed using the L_{PR} language, which is briefly described in this section. For more details see (Popova and Sharpanskykh 2006). The model describes the following objects (represented by sorts in L_{PR}): *tasks*, *processes* (particular instances of tasks in control flows), *resource types* describing information and material artifacts, *resources* (specific instances of resource types having specified amounts), *agents*, *roles* (sets of functionalities that can be assigned to agents), *goals*, *performance indicators* (measures based on which the goals are defined). Each object has a number of characteristics. For example, a task is characterized by a minimum duration (denoted by task_name.min_duration); a resource type has a characteristic expiration duration; resources are characterized by an amount. Furthermore, relations are defined over the objects. For example, the relation task_produces(t:TASK, rt:RESOURCE_TYPE, v:VALUE) specifies that task t produces amount v of resource type rt. Resource types that can be shared by several processes are specified in resource_sharable(rt:RESOURCE_TYPE, L:PROCESS_LIST).

The set of specified processes together with the set of ordering relation defined on them form a workflow. An example of an ordering relation is starts_after(p1: PROCESS, p2:PROCESS). It defines that process p1 starts after process p2. Furthermore, three types of structures specifying the flow of control between processes are defined: and-, or- and loop-structures. Branches of and-structures start simultaneously and are all executed. Only one branch of an or-structure can be executed depending on the or-condition. Loop structures contain processes that can be repeated depending on the loop-condition within a maximum number of iterations. Relations between roles, agents and processes are defined as follows: role_perfoms_process(r:ROLE, p:PROCESS) and agent_plays_role(a:AGENT, r:ROLE). Relations to goals and PIs are defined as follows: is_realized_by(g:GOAL, L:TASK_LIST) defining that goal g can be realized by performing tasks in list L and measures(i:PI, p:PROCESS) specifying that performance indicator i is a measure over some aspect of the performance of process p.

4 Execution Language L_{EX}

For the formalization of a trace, a dedicated sorted predicate language L_{EX} is used, which is based on L_{PR}. Each sort included into L_{EX} represents a set of individual objects of a certain type that occur in the trace (e.g., the sort PROCESS_EX contains all names of processes that have been executed in the trace). To distinguish the names of sorts of L_{EX} from the names of sorts in L_{PR}, all sort names of L_{EX} finish with the EX postfix.

The following sorts are included into L_{EX}:

PROCESS_EX – a set of all process names in a trace;

RESOURCE_EX - a set of all resource names;

RESOURCE_TYPE_EX – a set of all resource types names;

ROLE_EX – a set of all role names;

AGENT_EX – a set of all agent names;

PI_EX – a set of all performance indicators names;

VALUE_EX – an ordered set of numbers;
PROCESS_LIST_EX – a set of all names of process lists;
DECISION_VARIABLE_EX – a set of all names of decision variables;
DECISION_VAR_VALUE_EX – a set of all values of decision variables;
ENV_OBJECT_EX – a set of all environmental objects names;
OBJ_STATE_EX – a set of all names of states of objects;
OBJ_CHAR_EX – a set of all names of object characteristics.

To define events a number of relations are introduced into L_{EX} (see Table 1).

Table 1. Relations defined in L_{EX}

Predicate specification	Informal description
process_started: PROCESS_EX	A process has started
process_finished: PROCESS_EX	A process has finished
resource_used_by: RESOURCE_EX x PROCESS_LIST_EX x VALUE	A certain resource amount is used by a process
resource_consumed_by: RESOURCE_EX x PROCESS_EX x VALUE	A certain resource amount is consumed by a process
resource_produced_by: RESOURCE_EX x PROCESS_EX x VALUE	A certain resource amount is produced by a process
resource: RESOURCE_EX x RESOURCE_TYPE_EX	Identifies a resource of a certain resource type
resource_expired: RESOURCE_EX	A resource is expired
resource_invalid: RESOURCE_EX x VALUE	A certain resource amount became invalid (e.g. broken)
available_resource_amount: RESOURCE_EX x VALUE	Specifies the available amount of the resource
pi_has_value: PI_EX x VALUE	Identifies the value of a PI
agent_is_assigned_to_role: AGENT_EX x ROLE_EX	Specifies the assignment of an agent to a role
agent_performs_process: AGENT_EX x PROCESS_EX	Identifies that an agent performs a certain process
env_object_changed_state_into: ENV_OBJECT_EX x OBJ_STATE_EX	Specifies a changed state of an environmental object
env_object_changed_char_into: ENV_OBJECT_EX x OBJ_CHAR_EX x VALUE	Specifies the value of a certain characteristic of an environmental object
decision_taken: DECISION_VARIABLE_EX x DECISION_VAR_VALUE_EX	Identifies the value of a decision variable

5 Language TTL and TTL Checker Tool

To analyze traces the language TTL is used. TTL is a variant of order-sorted predicate logic, which allows reasoning about dynamic properties of systems. TTL properties considered in this paper are specified based on state properties expressed as formulae in L_{EX}. For enabling dynamic reasoning, TTL includes special sorts: TIME (a set of linearly ordered time points), STATE (the set of all state names of a system), TRACE (the set of all trace names), STATPROP (the set of all state property names). In TTL, formulae of the state language (L_{EX} in this case) are used as objects. Further we shall use t with subscripts and superscripts for variables of the sort TIME; and γ with subscripts and superscripts for variables of the sort TRACE. A state of a system in a trace is denoted using a function symbol state of type TRACE x TIME → STATE. The set of function symbols of TTL includes:

∧, ∨, →, ↔: STATPROP x STATPROP→ STATPROP,

not: STATPROP→ STATPROP,

∀, ∃: VARS x STATPROP→ STATPROP,

which are counterparts to the Boolean propositional connectives and quantifiers.

The states of a system are related to names of state properties via the satisfaction relation denoted by the infix predicate |= (or by the prefix predicate holds): state(γ,t)|= p (or holds(state(γ,t)), which denotes that the state property with a name p holds in trace γ at time point t. For example, state(trace1,10)|= process_started(p2) denotes that the process p2 has started in the trace1 at the time point 10. Both state(γ,t) and p are terms of TTL. All other TTL terms are constructed by induction in the standard predicate logic way.

Transition relations between states are described by dynamic properties, which are expressed by TTL-formulae. The set of *atomic TTL-formulae* is defined as:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds(v_1,u_1) is an atomic TTL formula.
(2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1=\tau_2$ is an atomic TTL formula.
(3) If t_1, t_2 are terms of sort TIME, then $t_1<t_2$ is an atomic TTL formula.

The set of *well-formed TTL-formulae* is defined inductively in a standard way using Boolean propositional connectives and quantifiers. TTL has semantics of the order-sorted predicate logic. A more detailed specification of the syntax and the semantics for the TTL is given in (Sharpanskykh and Treur 2006).

The analysis based on checking of TTL formulae on (one or more) traces is supported by the TTL Checker tool. Besides the logical analysis the tool allows statistical post-processing of the verification results. For this the following functions are used:

case(logical_formula, value1, value2): if logical_formula is true, then the case function is mapped to value1, otherwise – to value2.

sum([summation_variables], case(logical_formula, value1, 0)): logical_formula is evaluated for every combination of values from the domains of each from the summation_variables; and for every evaluation when the logical formula is evaluated to true, value1 is added to the resulting value of the sum function.

To provide support for analysts not skilled in logics, the tool allows defining parameterized templates (macros), which can be instantiated in different ways. Further details about the TTL Checker can be found in (Bosse et al. 2006). Examples of analysis cases that also include statistical processing will be given further in this paper.

6 Trace Conformity to a Model

As described earlier the process-oriented model consists of objects, characteristics and relations defined in L_{PR}. Every such model can be translated to a set of constraints that should be satisfied by actual execution traces. The constraints are represented as properties in TTL using L_{EX} as a state language. Each property is based on a specific combination of language constructs (ordering relations, and-/or-/loop-structures, object characteristics, etc.) In the following we define rules on how to translate different parts of the model specification to TTL properties.

The first property we consider represents the restriction that only processes specified in the model are allowed to be performed. It is formalized in TTL as follows. For specific process names p1, ..., pn:

C1: $\forall t, p{:}PROCESS\_EX\ state(\gamma, t) \models process\_started(p) \Rightarrow p = p1 \mid ... \mid p = pn$

The next properties represent the constraints that processes not part of any or-structure start and finish in the trace. For p1 a process not in any or-branch:

C2: $\exists t1\ state(\gamma, t1) \models process\_started(p1)$
C3: $\exists t1\ state(\gamma, t1) \models process\_finished(p2)$

The execution of processes in or- and loop-structures depends on the evaluation of conditions defined for these structures. In this case it needs to be checked whether the processes that have started also finish in the trace: For p1 a process in a loop-structure/or-branch:

C4: $\exists t1\ state(\gamma, t1) \models process\_started(p1)$
$\Rightarrow \exists t2{:}\ state(\gamma, t2) \models process\_finished(p1)$

Additionally for processes not in loop-structures:
C5: $\exists t1\ state(\gamma, t1) \models process\_started(p1)$
$\Rightarrow (\forall t3\ t3 \neq t1 \Rightarrow state(\gamma, t3) \models \neg process\_started(p1)$

The next property checks if the actual duration of a process is within the range defined by the corresponding task. For a process p1, a task tk, durations d1 and d2 such that [is_instance_of(p, tk), tk.min_duration=d1, tk.max_duration=d2]:

C6: $\exists t1, t2\ state(\gamma, t1) \models process\_started(p1)\ \&\ state(\gamma, t2) \models process\_finished(p2) \Rightarrow d1 \leq t2\text{-}t1\ \&\ t2\text{-}t1 \leq d2$

Ordering relations are translated to constraints in the following way. For p1, p2 such that starts_with(p1, p2):

C7: $\exists t1\ state(\gamma, t1) \models process\_started(p1) \Rightarrow state(\gamma, t1) \models process\_started(p2)$

C8: $\exists t1\ state(\gamma, t1) \models process\_started(p2) \Rightarrow state(\gamma, t1) \models process\_started(p1)$

C9: $\exists t1\ state(\gamma, t1) \models process\_finishes(p1) \Rightarrow state(\gamma, t1) \models process\_finishes(p2)$

C10: $\exists$t1 state(γ, t1) |= process_finishes(p2) $\Rightarrow$ state(γ, t1) |= process_finishes(p1)

For p1, p2 such that starts_during(p1, p2):
C11: $\exists$t1 state(γ, t1) |= process_started(p1)
$\Rightarrow$ $\exists$t2, t3 t2 $\leq$ t1 & t1 $\leq$ t3 $\wedge$ state(γ, t2) |= process_started(p2) &
state(γ, t3) |= process_finished(p2)

For p1, p2, d such that starts_after(p2, p1, d) except for beginning and ending of and-, or-, or loop-structures:

C12: $\exists$t1 state(γ, t1) |= process_finished(p1)
$\Rightarrow$ $\exists$t2: state(γ, t2) |= process_started(p2) & d = t2-t1

Next, and-structures are considered. Firstly, specifications such as [starts_after(p, begin_and(id), d), starts_after(begin_and(id), p1), ..., starts_after(begin_and(id), pn)] are treated as [starts_after(p1, p, d), ..., starts_after(pn, p, d)]. Furthermore the end of the structure should be considered [starts_after(end_and(id), p1), ..., starts_after(end_and(id), pn), starts_after(p, end_and(id))] and it should be checked whether the order of execution at the end of the and-structure matches the specified and-condition.

For p1, ..., pn, p, d such that [starts_after(end_and(id), p1), ..., starts_after(end_and(id), pn), starts_after(p, end_and(id), d), and_cond(id, any)]:

C13: $\exists$t1 state(γ, t1) |= [process_finished(p1) $\vee$ ... $\vee$ process_finished(pn)] & ($\forall$t2: t2 $\leq$ t1 $\Rightarrow$ state(γ, t2) |= [$\neg$process_finished(p1) $\wedge$ ... $\wedge$ $\neg$process_finished(pn)]
$\Rightarrow$ $\exists$t3 state(γ, t3) |= process_started(p) & d = t3-t1

For p1, ..., pn, p, d such that [starts_after(end_and(id), p1), ..., starts_after(end_and(id), pn), starts_after(p, end_and(id), d), and_cond(id, all)]:

C14: $\exists$t1, ..., tn, tp state(γ, t1) |= process_finished(p1) & ...
& state(γ, tn) |= process_finished(pn) & tp $\geq$ t1 & ... & tp $\geq$ tn & (tp = t1 | ... | tp = tn) $\Rightarrow$
$\exists$tt state(γ, tt) |= process_started(p) & d = tt-tp

And-conditions with other expressions are treated similarly taking into account which processes should finish so that the next process can start, for example: [starts_after(end_and(id), p1), ..., starts_after(end_and(id), pn), starts_after(p, end_and(id), d), and_cond(id, finished(p1)$\wedge$finished(p2))] can be checked as follows:

C15: $\exists$t1, t2, t state(γ, t1) |= process_finished(p1) & state(γ, t2) |= process_finished(p2) &
t1 $\leq$ t & t2 $\leq$ t & (t = t1 | t = t2)
$\Rightarrow$ $\exists$t3 state(γ, t3) |= process_started(p) & d = t3-t

For or-structures it should be checked if exactly one of the branches is executed and it matches the specified or-condition. An or-condition is an expression based on a decision variable (related to a decision process), state or a characteristic of an environmental object. For p, p_1,..., p_n, d, and a condition based on the decision variable dv such that [starts_after(begin_or(id),p,d),starts_after(p_1, begin_or(id)), ..., starts_after(p_n, begin_or(id)), or_cond(id, dv), or_branch(p_1, val_1),..., or_branch(p_n, val_n)] (similarly for other conditions):

C16: $\exists$t1 state(γ, t1) |= process_finished(p) $\Rightarrow$ $\exists$t2 (state(γ, t2) |= process_started(p_1) & $\forall$t3 state(γ, t3) |= [$\neg$process_started(p_2) $\wedge$ ... $\wedge$ $\neg$process_started(p_n)] & $\exists$t4 state(γ, t4) |= decision_taken(dv, val_1) & t4 $\leq$ t2 & ($\forall$t5 t5 $\geq$ t4 & t5 $\leq$ t2 & state(γ, t5) |= decision_taken(dv, val) $\Rightarrow$ val = val_1) | ... | (state(γ, t2) |= process_started(p_n) & $\forall$t6 state(γ, t6) |= [$\neg$process_started(p_1) $\wedge$ ... $\wedge$ $\neg$process_started(p_{n-1})] & $\exists$t7 state(γ, t7) |= decision_taken(dv, val_n) & t7 $\leq$ t2 & ($\forall$t8 t8 $\geq$ t7 & t8 $\leq$ t2 & state(γ, t8) |= decision_taken(dv, val) $\Rightarrow$ val = val_n)) & d = t2-t1

In a similar way, formulations can be given for the case of a condition based on the state of an environmental object or a characteristic of an environmental object.

Furthermore, it should be checked that the processes in the branches that did not start also are not executed. For every or-branch such that [starts_after(p1, begin_or(id)), starts_after(p2, p1), ..., starts_after(pn, pn-1), starts_after(end_or(id), pn)] the following property should be checked:

C17: $\forall$t1 state(γ, t1) |= $\neg$process_started(p1)
$\Rightarrow \forall$t2 state(γ, t2) |= [$\neg$process_started(p2) $\wedge$ ... $\wedge \neg$process_started(pn)]

Furthermore it should be checked that the process after the or-structure starts correctly:

For p_1, ..., p_n, p, d such that [starts_after(end_or(id), p_1), ..., starts_after(end_or(id), p_n), starts_after(p, end_or(id), d)]:

C18: $\exists$t1 state(γ, t1) |= [process_finished(p_1) $\vee$ ... $\vee$ process_finished(p_n)] $\Rightarrow \exists$t2 state(γ, t2) |= process_started(p) & d = t2-t1

Next, loop-structures are considered. Specifications such as [starts_after(begin_loop(id), p1), starts_after(p2, begin_loop(id))] are treated as starts_after(p2, p1). Furthermore for every process in a loop-structure the corresponding sequencing relations are checked in a similar way. For the last process p2 in a loop-structure with a condition expression dv = val for a decision variable dv such that [starts_after(p1, begin_loop(id)), ..., starts_after(end_loop(id), p2, d2), starts_after(p3, end_loop(id), d3), loop_cond(id, dv = val), loop_max(m)] the correct execution order w.r.t. a loop condition and a maximal number of iterations should be checked:

C19: $\exists$t1 state(γ, t1) |= process_finished(p2) $\Rightarrow$
$\exists$t2 (state(γ, t2) |= process_started(p1) & $\exists$t3 state(γ, t3) |= decision_taken(dv, val) & t3 $\leq$ t2
& ($\forall$t4 t4 $\geq$ t3 & t4 $\leq$ t2 & state(γ, t4) |= decision_taken(dv, val1) $\Rightarrow$ val = val1) &
$\neg$max_iter(p2) & t2-t1 = d2) & (state(γ, t2) |= process_started(p3) & t2-t1 = d2+d3 &
($\exists$t3 state(γ, t3) |= decision_taken(dv, val1) & val1 $\neq$ val & t3 $\leq$ t2 & ($\forall$t4 t4 $\geq$ t3 & t4 $\leq$ t2 &
state(γ, t4) |= decision_taken(dv, val2) $\Rightarrow$ val2 = val1) | max_iter(p2)))

Property max_iter(p2) can be defined as follows where m is the maximal number of iterations:

$\exists$t1, ..., tm t1 $\neq$ t2 $\wedge$ ... $\wedge$ t1 $\neq$ tm $\wedge$ ... $\wedge$ tm-1 $\neq$ tm $\wedge$ state(γ, t1) |= process_started(p2) & ... & state(γ, tm) |= process_started(p2)

Different types of conditions are treated similarly taking into account the specific condition variable.

The following properties concern resources and resource types and how they are used/consumed/produced/shared by processes.

For resource type rt, task tk, amount v and process p such that [is_instance_of(p, tk), task_consumes(tk, rt, v)]:

C20: sum([r:RESOURCE_EX], case($\exists$t1, t2 state(γ, t1) process_started(p) &
state(γ, t2) |= process_finished(p) & $\exists$t3 t1 $\leq$ t3 & t3 $\leq$ t2 $\wedge$ state(γ, t3) |=
resource_consumed_by(r, p, v1) & $\exists$t4 state(γ, t4) |= resource(r, rt), v1, 0)) = v

For resource type rt, task tk, amount v and process p such that [is_instance_of(p, tk), task_uses(tk, rt, v)] for every time point t in the trace it will be checked that the resource that is used matches the specification:

C21: sum([L:PROCESS_LIST_EX], case(∃t1, t2 state(γ, t1) |= process_started(p) & state(γ, t2) |= process_finished(p) & t1 ≤ t & t ≤ t2 & state(γ, t) |= resource_used_by(r, L, v1) & is_in_list(p, L) & ∃t4 state(γ, t4) |= resource(r, rt), v1, 0)) = v

For resource type rt, task tk, amount v and process p such that [is_instance_of(p, tk), task_produces(tk, rt, v)]:

C22: ∃t1, t2 state(γ, t1) |= process_started(p) & state(γ, t2) |= process_finished(p) ⇒
∃t3 t1 ≤ t3 & t3 ≤ t2 & state(γ, t3) |= resource_produced_by(r, p, v) &
∃t4 state(γ, t4) |= resource(r, rt)

In the model, the resources available at the beginning of the workflow are represented as produced by the BEGIN process. Thus it should be checked if the available amount at the beginning of the trace matches the amount produced by the BEGIN process. For resource r such that [process_output(BEGIN, r), is_resource_type(r, rt), r.amount=v]:

C23: sum([r:RESOURCE_EX], case(state(γ, 0) |= [available_resource_amount(r, v1) ∧ resource(r, rt)], v1, 0)) = v

It should also be checked whether the resources are shared between lists of processes for which this is allowed. For resource type rt and list of processes L such that [resource_sharable(rt, L)]:

C24: ∃t1 ∃L1:PROCESS_LIST_EX state(γ, t1) |= resource_used_by(r, L1, v) & ∃t2 state(γ, t2) |= resource(r, rt) ⇒ is_sublist_of(L1, L)

Finally it should be checked if role/process assignments to agents are correct. For role r, agent a, process p such that [role_performs_process(r,p),agent_plays_role(a, r)]:

C25: ∃t1, t2 state(γ, t1) |= process_started(p) & state(γ, t2) |= process_finished(p) ⇒ ∀t3 t1 ≤ t3 & t3 ≤ t2 & state(γ, t3) |= [agent_performs_role(a, r) ∧ agent_performs_process(a, p)]

The above listed properties are general and can be checked in any order on the execution trace. However in many cases it would be beneficial to enforce certain order of checking. Often when one constraint is violated that causes the violation of others but finding all of them might not add much more information on what went wrong. For example, when one process fails to produce a resource necessary for another process this might cause changes or even failures in the rest of the execution trace. However these are only consequences of the first failure – the production of the resource. It is therefore useful to alert the analyst of the first time point at which a violation of a constraint occurs. The approach proposed here is to consider the events of the trace in their natural temporal order. For each event that represents a starting or finishing point of a process only a selection of the relevant general constraints instantiated for a specific time point(s) and a specific event(s) are checked.

In the following we define the sets of relevant constraints w.r.t. the type of event occurring in the trace. The first constraints to be checked are C23 (available resource at the first time point) and C2 (checks if a process starts) for the first process(es) in the workflow that should start at the first time point unconditionally. If at the first time point an or-structure begins then it should be checked that only one branch is executed and it matches the evaluation of the condition (C16). Afterwards the (partially) ordered list of starting and finishing points of processes is considered. For every starting point the following types of constraints are considered (in this order):

(1) the process is defined in the model (C1);
(2) the process has not been executed before if not in loop-structures (C5);
(3) constraints w.r.t. the conditions for and-structures (C13, C14, C15);
(4) constraints related to starts_with and starts_during (C7, C8, C11);
(5) the process finishes (C3, C4).

For every finishing point the following constraints are checked (in this order):
(1) resource-related constraints (C20, C21, C22, C24);
(2) agent-/role-related constraints (C25);
(3) durations (C6);
(4) constraints related to finishes_with (C9, C10);
(5) constraints on the next process (C12, C16, C17, C18, C19).

From all types of considered constraints those are selected that refer to the specific process to which the starting or finishing point belongs. When more events coincide finishing points are considered before starting points.

The above described approach assumes the availability of the whole execution trace at the beginning of the analysis. In some situations it might be necessary to perform such analysis while the trace is being generated. This gives the possibility to react as soon as an event in the execution deviates from the model and take appropriate measures.

With some adjustments, the generic properties can be used here as well. The analysis process works as follows. The information about events from the trace become available following the order of the time points at which they occur and all events happening at the same time point become available all at once. Depending on the type of the current considered event specific types of constraints are checked or assigned to be checked at specific time points in the future. The system gives a warning when a constraint is violated by the trace. At the first time point again the available resource (C23) and the starting of the first processes is checked (C2, C16). Then for every starting point of a process that appears in the trace the following types of constraints are considered:

(1) the process is defined in the model (C1);
(2) the process has not been executed before in the part of the trace up to the current event (C5);
(3) constraints with respect to the conditions for the end of and-structures (C13, C14, C15);
(4) constraints related to synchronizations starts_with and starts_during (C7, C8, C11);
(5) existence of a finishing point for the process (based on C3, C4) – as this information is not yet available in the trace, the corresponding properties are scheduled to be checked for every time point until such a finishing point occurs. When the maximal duration specified by the model is passed and no finishing point has yet occurred a warning is given that the process exceeds its allowed duration. A warning is also generated if the process finishes before its minimal duration specified by the model has passed;
(6) resource-related constraints of the following types are checked for every time point until the process finishes: resource sharability (C24), resource used by a process (C21), resource produced (C22) or consumed (C20) by the process

up to the current time point is checked not to exceed the specified amount in the model;

(7) agent- and role-related constraints are checked for every time point until the process finishes.

For every finishing point the following types of constraints are considered (in this order):

(1) resource produced (C22) or consumed (C20) by the process for its whole duration is checked to be equal to the pre-specified amount in the model;

(2) constraints related to synchronizations finishes_with (C9, C10);

(3) constraints with respect to the process which should start next (C12, C16, C17, C18, C19) – since the necessary information is not yet available in the trace, the properties are scheduled to be checked in the following way. For every time point it is checked if the expected process has started until information about its starting point arrives. If this starting point is before the pre-specified delay a warning is issued. A warning is also issued when the delay has passed and the process has not started yet. C16 is checked until the starting point of the first process in a branch of the or-structure. Afterwards it is checked for the rest of the incoming trace that none of the other first processes of other branches of this structure start at any later point (C17). At the end of the or-structure a property is scheduled for checking if the process after the or-structure starts for every time point until the process actually starts. For loop-structures a counter is kept for the current number of iterations. It is used to determine the next process together with the current evaluation of the condition. Based on that the appropriate property is scheduled to be checked until the correct process starts.

For most types of constraints the following rule is used: when a specific constraint is checked or scheduled for checking it is marked and is not considered any more at the events occurring later. Exception is made for the loop-related constraints which might need to be considered multiple times.

7 Conformity to a Formal Organization

A formal organization is specified by a fixed set of rules that define (prescribe) organizational structure and behavior and are formalized as predicate logic constraints imposed on a process-oriented model. These rules are usually described in different organizational and general normative documents (e.g., an organizational mission statement, a strategy description, laws, organizational normative acts, different policies, job and procedure descriptions) and are formalized as predicate logic constraints imposed on a process-oriented model.

In (Popova and Sharpanskykh 2006) different types of constraints are described (e.g., domain-specific, physical world constraints). Some of these constraints are strict and should not be violated in any organizational scenario; e.g., “all employees involved in a certain process, which has a risk factor for human health, should be provided with the necessary safety means”. Other rules are less strict and can be

(temporally) violated; e.g., “the average amount of a certain resource produced by an organization is required to be greater than a certain number”.

In general, if a trace conforms to the corresponding organization model (i.e., a trace is in the set of possible executions of the model), then all constraints imposed on and satisfied by the model are also satisfied by the trace. However, when the checking of the conformity of a trace to an organization model fails, then the satisfaction of the constraints by the trace is not guaranteed any more. In this case the analysis of the conformity of a trace to a formal organization should be performed by checking organization-specific properties. Such properties are based on dependencies and characteristics defined in (implied by) an organizational model, or correspond to different types of constraints (e.g., domain-specific, physical world constraints) defined for the model.

In the following several examples of formal organization properties that can be checked on traces are considered.

P1: In the trace $\gamma 1$ the process p1 is executed (after some time) after the process p2 has finished:

$\exists t1, t2\ t1 \leq t2\ state(\gamma, t1) \models process\_finished(p2) \ \&\ state(\gamma, t2) \models process\_started(p1)$

For example, it is required that a product produced by an organization should be eventually delivered to the customer. Other properties expressing ordering relations between processes (also including references to real time) are specified in a similar manner.

P2: For the specified set of traces TR the average overall amount of resources of type r produced by an organization up to a time point t should be at least n:

$sum([\gamma{:}TR, t'{:}between(0, t), r'{:}RESOURCE\_EX], case(\exists a'{:}PROCESS\_EX\ \exists am{:}VALUE\_EX\ state(\gamma, t') \models [\ resource\_produced\_by(r', a', am) \wedge resource(r', r)], am, 0)) / sum([\gamma{:}TR], case(true, 1, 0)) \geq n,$

here between(0, t) represents a set of all natural numbers in the interval [0, t].

P3: In the trace $\gamma 1$ the amount of loss of resources of type r caused by the consumption, usage, and invalidation evaluated at the time point t should be less than m.

$sum([t'{:}\ between(0, t), r'{:}RESOURCE\_EX], case(\exists a'{:}PROCESS\_EX\ \exists am1{:}VALUE\ state(\gamma 1, t') \models [\ resource\_produced\_by(r', a', am1) \wedge resource(r', r)], am1, 0)) - sum([t'{:}between(0, t), r'{:}RESOURCE\_EX], case(\exists a'{:}PROCESS\_EX\ \exists am2{:}VALUE\_EX\ state(\gamma 1, t') \models [\ resource\_consumed\_by(r', a', am2) \wedge resource(r', r)], am2, 0)) - sum([t'{:}between(0, t), r'{:}RESOURCE\_EX], case(\exists am4{:}VALUE\_EX\ state(\gamma 1, t') \models [\ resource\_invalid(r', am4) \wedge resource(r', r)], am4, 0)) - sum([r'{:}RESOURCE\_EX], case(\exists l{:}PROCESS\_LIST\_EX\ \exists am2{:}VALUE\_EX\ state(\gamma 1, t) \models [\ resource\_used\_by(r', l, am3) \wedge resource(r', r)], am3, 0)) < m$

P4: In the trace $\gamma 1$ a resource r produced by an organization required by some other organizational processes should be used or completely consumed before its expiration date.

$\exists t\ \exists p{:}PROCESS\_EX\ \exists am{:}VALUE\_EX\ state(\gamma 1, t) \models resource\_produced\_by(r, p, am)\ \&\ [\forall t'\ t' > t\ state(\gamma 1, t') \models resource\_expired(r)$
$\Rightarrow \exists t''\ \exists pl{:}\ PROCESS\_LIST\_EX\ \exists am2\ t' > t'' \ \&\ t'' > t\ resource\_used\_by(r, pl, am2)]$

P5: In the trace $\gamma 1$ the overall amount of working hours of an agent a at time point t (e.g., a time point in the end of some working period) should not exceed n:

(sum([t': between(0, t), p':PROCESS_EX], case(state(γ1, t')|= [agent_performs_process(a, p') ∧ process_finished(p')], t', 0)) – sum([t'': between(0, t), p': PROCESS_EX], case(state(γ1, t')|= [agent_performs_process(a,p') ∧ process_started(p')], t'',0))) ≤ n

P6: In the trace γ1 no agent executes more than one process at the same time:

∀p1:PROCESS_EX ∀t1 state(γ1, t1) |= [agent_performs_process(a, p1) ∧ process_started(p1)]

⇒ ∃t2 state(γ1, t2) |= process_finished(p1) & ∀t' t'≤t2 & t'≤t1 ∀p'≠p1 state(γ1, t')|= (¬process_started(p') ∧ ¬agent_performs_process(a, p'))

P7: In the trace γ1 at the time point t the amount of available resources of type r is at least a pre-specified minimum amount min.

sum([r':RESOURCE_EX], case(∃am1:VALUE_EX state(γ1, t)|= [available_resource_amount(r', am1) ∧ resource(r', r)], am1, 0)) > min

8 Analysis of Emergent Properties

Emergent properties are not specified and not implied by an organizational model and are related only to (result from) an actual execution(s) of an organization. Such properties may be checked for different reasons: e.g., to optimize the organizational operation by discovering and eliminating bottlenecks. Many emergent properties include a post-processing of the checking results by applying different statistical functions: e.g., sum, average, minimum, maximum, and are often expressed over multiple traces. Consider several examples:

E1: For the specified set of traces TR, determine a frequency of finishing the process p on time (i.e., duration should be within the interval [min_duration, max_duration]).

sum([γ:TR], case(∃t1,t2 state(γ, t1)|= process_started(p) & state(γ, t2)|= process_finished(p) & (t2-t1) ≤ max_duration & (t2-t1) ≥ min_duration], 1, 0)) / sum([γ:TR], case(∃t1 state(γ, t1)|= process_started(p), 1, 0))

E2: In the trace γ1 at the time point t calculate the average workload of agents of an organization:

(sum([t1: between(0, t), p':PROCESS_EX, a':AGENT_EX], case(state(γ1, t1) |= [agent_performs_process(a', p') ∧ process_finished(p')], t1, 0) – sum([t2: between(0, t), p':PROCESS_EX, a':AGENT_EX], case(state(γ1, t2)|=
[agent_performs_process(a', p') ∧ process_started(p')], t2, 0))) / sum([a':AGENT_EX], case(true, 1, 0))

E3: Maximum duration of a process p in all executions:

∃γ1, t1, t2 state(γ1, t1)|= process_started(p) & state(γ1, t2)|= process_finished(p) & ∀γ'≠γ1 ∀t1', t2' [state(γ', t1') |= process_started(p) & state(γ', t2')|= process_finished(p) & (t2'-t1')<(t2-t1)]

E4: In all executions the delay between the end of the process p1 and the beginning of the process p2 should be less than n

∀γ ∀t1, t2 state(γ, t1) |= process_finished(p1) & state(γ, t2) |= process_started(p2) ⇒ (t2-t1)< n

9 Performance Evaluation

The performance of an organization at a certain time point (for a certain period) is evaluated by determining the satisfaction of key organizational goals. These goals range from high-level abstract goals to very specific ones. High-level goals are decomposed to more specific goals which are easier to measure, thus, forming goal decomposition structures. Goals are defined and discussed in (Popova and Sharpanskykh 2006) as part of the performance-oriented view on organizations. Example of goals are: 'It is desired to maintain high degree of product quality', 'It is desired to achieve high customer satisfaction', 'It is desired to maintain number of work-related accidents per year to less than 3', etc.

Goals are formulated based on performance indicators (PIs), which are associated with certain organizational processes. Examples of PIs are: product quality, customer satisfaction, number of accidents, productivity, etc. The values of these PIs are measured (directly or indirectly) during or after the process execution depending on the goal evaluation type and in the end or during a certain period of time (goal horizon). Then, by comparing the measured values with the corresponding goal expressions, the satisfaction of the goals is determined. Further, the obtained goal satisfaction measure is propagated by applying the rules defined in (Popova and Sharpanskykh 2006), upwards in the goal hierarchy for determining the satisfaction of high level goals. An example of this type of analysis is given further in the frames of the case study.

10 Case Study

The application of different types of analysis will be illustrated in the context of an organization from the security domain. The main purpose of the organization is to deliver security services to different types of customers. The organization has well-defined multi-level structure that comprises several areas serving groups of locations (security objects) and has predefined (to a varying degree) job descriptions for employees (approx. 230.000 persons). The allocation of employees to security objects is based on plans created by planning groups.

The planning process consists of the forward (or long-term) planning and the short-term planning. The forward planning is a process of creation of plans describing the allocation of security officers within the whole organization for a long term (4 weeks). Forward plans are created based on customer contracts by forward planners. During the short-term planning, plans that describe the allocation of security officers to locations within an area for a short term (a week) are created and updated based on the forward plan and up-to-date information about the security employees. Based on short term plans, daily plans are created. Within each area the short-term planning is performed by the area planning team that consists of planners and is guided by a team leader.

The position of the forward planners in the organizational structure has changed as a result of a reorganization in the past. Before the reorganization each planning team had a forward planner who was mainly responsible for the creation of long-term plans

for the area. After the reorganization the forward planners were combined into a centralized forward planning group. A number of reasons for such a change are identified in the reorganization reports. In the following it will be shown how the proposed analysis techniques could be used for automated justification of the identified performance bottlenecks and other problems in the organization.

(1) Uneven workload of forward planners in different area planning teams.

This statement can be checked by calculating the workload for the forward planners in different areas and comparing the results. For this the following property can be used with a – the agent name, for whom the workload is calculated, and t – the time point up to which the workload is calculated:

sum([t1: between(0, t), p':PROCESS_EX], case(state(γ1, t1) |= [agent_performs_process(a, p') ∧ process_finished(p')], t1, 0)) - sum([t2: between(0, t), p':PROCESS_EX], case(state(γ1, t2)|= [agent_performs_process(a, p') ∧ process_started(p')], t2, 0)),
here a is an agent name

If multiple traces are available, the average workload of every agent can be calculated as it is demonstrated in property E2. A side-effect of high workload could be the undue execution of some processes assigned to the forward planner. This can be established by verifying the correspondence of the actual execution to the model.

(2) Certain forward planning tasks require collaboration with other forward planners. In the previous organization this has been achieved by informal (i.e., not specified by a formal organizational model) cooperation between forward planners from different areas.

This statement can be justified in two steps. First by performing the analysis of the correspondence of a trace to the model, it can be established that in the trace exist processes performed by agents that are not allocated to the roles, to which these processes are assigned. Then, the number (or frequency) of such processes until the time point t for each role r can be calculated as follows:

sum([p':PROCESS_EX], case(∃t1<t ∃a:F_PLANNER
state(γ1, t1) |= [agent_performs_process(a,p') ∧ ¬agent_performs_role(a1, r)], 1, 0))

For multiple traces (a set TR), the average number of such processes for role r can be calculated as follows:

sum([γ:TR, p':PROCESS_EX], case(∃t1<t ∃a:F_PLANNER state(γ, t1) |= [agent_performs_process(a, p') ∧ ¬agent_performs_role(a1, r)], 1, 0)/sum([γ:TR], case(true, 1, 0))

(3) Planning activities within each area were isolated from each other. Sometimes this led to situations, when customer requests in one area were not satisfied due to lack of security officers, whereas in other areas available employees were in plenty.

Such situations could be identified by calculating the (average) number of customer requests that were not accomplished by the organization until the time point t:

sum([t1: between(0, t)), r': CUSTOMER_REQUEST], case(state(γ1, t1) |= env_object_changed_state_into(r', active) & ∀t2 t2>t1 state(γ1, t2) |= ¬env_object_changed_state_into(r', satisfied), 1, 0))

In the following section we illustrate in more detail the different types of analysis of execution traces using the activities of the short-term planners after the reorganization of the planning departments.

11 Examples of Trace Analysis

Based on company documents such as job descriptions, company policy, procedures, etc., a process-oriented model was created for the planning departments. Part of this model dedicated to the creation of daily plans and short-term plans within one day is considered here. In the first half of the day security employees should provide their data change forms (requests for changes in the allocation schedule) to the unit manager (defined as process p3) who then checks and improves the data (p4) and puts it in the system (p5). At the same time the planners are working on other tasks, for example during the last week of the month they create a new short-term plan (STP) for the next month (p1). In the second half of the day they work on creating a daily plan (p6) for the next day (using the data change information in the system), inputting it in the system (p7) and informing all concerned (p8). Then they update the current short-term plan if necessary (p9) and so on. Part of the specification of the model is shown below:

```
starts_after(begin_and(and1), BEGIN, 0)
starts_after(begin_or(or1)
begin_and(and1), 0)
starts_after(p3, begin_and(and1), 0)
starts_after(p4, p3, 0)
starts_after(p5, p4, 0)
starts_after(p2, begin_or(or1), 0)
or_cond(or1,week_state)
or_branch(last,p1)
or_branch(other,p2)
starts_after(end_or(or1), p1, 0)
starts_after(end_or(or1), p2, 0)
starts_after(begin_and(and1), p5, 0)
starts_after(begin_and(and1), end_or(or1), 0)
and_cond(and1, all)
starts_after(p6, end_and(and1), 0.5)
...
role_performs_process(sec_officer, p3) role_performs_process(planner, p1)
...
is_instance_of(p1, t1)
task_produces(t1, STP, 1)
t1. min_duration = 3.5h
t1.max_duration = 4h
...
```

Based on this specification constraints are generated (as discussed earlier). For example, the first few lines of the specification generate the following constraints for the first time point of an execution trace:

state(γ, 0) |= process_started(p3) (based on C2)
state(γ, 0) |= process_started(p2) & (∀t3 state(γ, t3) |= ¬process_started(p1)) & state(γ, 0) |= ¬env_object_changed_state_into(week, last) | (state(γ, 0) |= process_started(p1) & (∀t3 state(γ,

t3) |= ¬process_started(p2)) & state(γ, 0) |= env_object_changed_state_into(week, last) (based on C17)

∀p:PROCESS_EX state(γ, 0) |= process_started(p) ⇒ p = p1 | p = p2 | p = p3 (based on C1)

Also based on company documents traces were created corresponding to this model. One such trace is used to illustrate the analysis of whether an execution trace agrees with the model. The trace represents a day from the last week of the month. Part of this trace is shown in Fig. 1. In the left part the atoms are listed and in the right part the time line is shown consisting of 12 hours. The time line is relative to the trace and not expressed in absolute date and time stamps. The absolute time line can always be calculated given the time stamp of the beginning of the trace. For each atom, the time interval for which it is true is displayed by a dark-grey bar while a light-grey bar designates that the value is false. For example for the whole duration of the trace agent a1 is assigned to play the role of a security officer and process_started(p1) is only true for time point 0.

The trace in Fig. 1 contains a process that is not in the model, p12. It is executed instead of process p3. According to p3, the security officers should deliver the change forms to the unit manager however on that day the unit manager was unavailable and the forms were brought directly to the planners (p12) who then had to check and improve them and input them in the system. These extra tasks prevented the planners from finishing their work on creating a short-term plan on time. Therefore all other processes during the rest of the day were shifted later than the model specified.

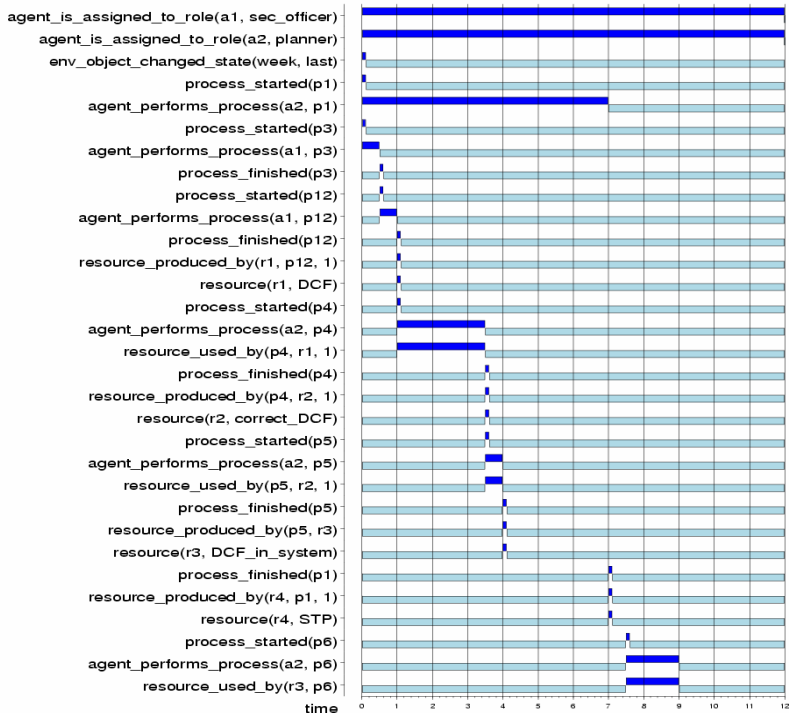


Fig. 1. The execution trace used for illustration

The trace is considered time point by time point taking into account the starting and finishing points of processes. We assume that the analysis is performed in real time, i.e. only the part of the trace up to the current time point is available. At time point 0 the three constraints given above are checked. They are satisfied since the only two processes starting are p3 and p1 and at this time point the state of the object week is indeed 'last'. Next the following properties are scheduled to be checked at every time point t until satisfied:

state(γ, t) |= process_finished(p1)
state(γ, t) |= process_finished(p3)

If that does not happen before the end of the trace then it is considered that this constraint is violated. Also the minimal and maximal duration of the processes should be according to the model:

state(γ, t) |= process_finished(p1) ⇒ t ≥ 3.5
state(γ, t) |= process_finished(p1) ⇒ t ≤ 4
state(γ, t) |= process_finished(p3) ⇒ t = 1

Next resource-related constraints are considered. The only relevant resource is the collection of data change forms DCF which is considered as a whole and only one collection can be produced. Thus C22 is not relevant.

Also agent-/role-related constraint C25 is scheduled for checking at every time point t until the process finishes.

state(γ,t) |= ¬process_finished(p1)
⇒ state(γ,t) |= [agent_plays_role(a2,planner) ∧ agent_performs_process(a2, p1)]
state(γ,t) |= ¬process_finished(p3)
⇒ state(γ,t) |= [agent_plays_role(a1,sec_officer) ∧ agent_performs_process(a1, p3)]

From all the scheduled constraints one fails at time point 0.5 when process p3 finishes – its duration is below the specified minimal duration of 1 hour. At this step the analysis stops – the trace does not agree with the model and the first process that violates the constraints is p3. Then, at this point it can be checked whether and which important organizational properties are satisfied (i.e., conformity to the formal organization). One of the properties extracted from the organizational documents of the company is that a daily plan for the next day is available before the end of the current working day, expressed as follows:

∃t, p:PROCESS_EX, r:RESOURCE_EX
state(γ,t) |= [resource_produced_by(r, p) ∧ resource(r, daily_plan)]

This property is satisfied by the trace.

Another property says that if the planners need to update the short-term plan then this should be performed only after the daily plan is available:

∃t1, t2, p:PROCESS_EX, r:RESOURCE_EX
state(γ, t1) |= [resource_produced_by(r, p) ∧ resource(r, daily_plan)] & state(γ, t2) |= process_started(p9) ⇒ t1 ≤ t2

This property is also satisfied.

Analyzing this trace it can be seen that the reason why the planners get overloaded is because the unit manager was not available to perform the processes assigned to him. Based on this, the analyst might decide to check in what percentage of the traces it happens that the work load of the unit manager is less than 3 hours. This can be checked by the following emergent property:

sum([p:PROCESS_EX], case(∃t1, t2 state(γ, t1) |= [process_started(p) ∧ agent_performs_process(a, p) ∧ agent_performs_role(a, unit_manager)] & state(γ, t2) |= process_finished(p), t2-t1, 0)) < 3

Also it can be determined if the events specified in the trace had an impact on the organizational performance. One of the high-level goals of the organization considered in the case study is the goal G1: 'It is required to maintain good level of satisfaction of the employees'. This general goal is decomposed into more specific goals among which is the goal G1.1: 'It is required to maintain that the level of work load is moderate'. This is again decomposed into even more specific goals among which is the goal G1.1.1: 'It is required to achieve that the number of working hours per day for each employee is not more that 8'. This goal is based on the performance indicator P1: 'working hours per day per employee' which can be evaluated for every trace for the last point t of the trace.

∀v:VALUE state(γ, t) |= pi_has_value(P1, v) ⇒ v ≤ 8

For the trace in Fig. 1 it will be calculated and included at the end of the trace that pi_has_value(P1, 11) which is more than 8. Thus goal G1.1.1 is not satisfied and contributes negatively to the satisfaction of G1.1 which is propagated upwards in the goals structure.

12 Discussion

This paper introduces automated techniques for manifold formal analysis of actual executions based on process-oriented models of organizations. On the one hand these techniques allow identifying errors and inconsistencies in executions of organizational scenarios, on the other hand they provide means for the evaluation and improving of organizational performance.

In order to check the conformity of a trace to the process-oriented specification, the translation of the specification to properties in the language of the execution traces is required. This step is necessary since the trace is specified using a language different from the language of the process-oriented specification (e.g., a time-indexed sequence of events in a log-file). We consider it a necessary step for every process-oriented approach. It might be done implicitly within an algorithm. In this paper, however, all translated properties are formulated explicitly which allows precise feedback on which from these properties are satisfied and which not. The translation is performed only once and the resulting properties can be checked on every new-coming trace. Also, the translation is based on clear translation rules therefore the process can be automated.

Furthermore, for the proposed analysis techniques the TTL language and the environment TTL Checker are used, which allow high expressivity in specification of

properties, including precise timing relations, references to multiple states (execution histories), arithmetical operations and checking properties on multiple traces. All these possibilities make TTL more expressive language than the standard modal logics (e.g., LTL, CTL, ATL) and calculi. Although TTL is an intuitive, close to the natural language, to define complex properties some skills in logics are needed. To support designers (e.g., managers) not skilled in logics, the used tool allows defining parameterized templates (macros) for TTL formulae, which can be instantiated in different ways which can also be used.

The analysis techniques introduced in this paper can be applied to both mechanistic and organic organizations. In particular, since many mechanistic organizations are characterized by a high stability and a large number of routine processes that can be specified with high precision, the verification of the conformity of actual executions of such organizations to a formal process-oriented model is of special importance. At the same time organic organizations are highly dynamic and their processes are very flexible, variable and often unpredictable. Therefore, models for such organizations can be specified only at a high abstraction level, sometimes defining only interface states (i.e., inputs and outputs) of high-level processes, and then the analysis techniques for the evaluation of emergent organizational processes and performance can be applied.

In the proposed approach traces are based on the actual execution of organizational scenarios. Such traces can be obtained in different ways: (1) automatically generated by a WfMS; (2) if data about the execution are represented in the form of informal logs obtained based on a process-oriented model in L_{PR}, they can be formalized (manually or automatically) using the language L_{EX}; (3) in case data about the execution are represented in some other formal language, the translation between this language and L_{EX} (if possible) is performed. Note that the translation and further analysis of traces obtained by (3) is possible only if a model based on which an original trace is generated can be related to an equivalent model in L_{PR}. Traces can be also generated based on a process-oriented model by performing simulations. Such traces can be used for diagnosis of inconsistencies, redundancies and errors in organizational structure and behavior. This type of analysis and the dedicated software are described in (Broek et al. 2006).

In the future it will be investigated how the proposed techniques can be applied for the analysis of inter-organizational processes. Also more analysis cases supported by the proposed techniques will be performed in the context of real organizations.

References

1. Aalst, W. van der; Beer, H.; and Dongen, B. van. 2005. “Process Mining and Verification of Properties: An Approach based on Temporal Logic”. In *On the Move to Meaningful Internet Systems*. Springer-Verlag, Berlin.
2. Aalst, W. van der and Hee, K.M van. 2002. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA.
3. Barjis, J; Shishkov, B and Dietz, J. 2002. “Validation of Business Components via Simulation”. In *Proceedings of the 2002 Summer Computer Simulation Conference*.

4. Bosse, T.; Jonker, C.M.; Meij, L. van der; Sharpanskykh, A. and Treur, J. 2006. "Specification and Verification of Dynamics in Cognitive Agent Models". In Nishida, T. (ed.), *Proceedings of the Sixth International Conference on Intelligent Agent Technology*, IAT'06. IEEE Computer Society Press, pp. 247-254
5. Broek, E.; Jonker, C.; Sharpanskykh, A.; Treur, J. and Yolum, P. 2006. Formal Modeling and Analysis of Organizations. In O. Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowsk, J. Sichman and J. Vazquez Salceda (eds.), *Coordination, Organization, Institutions and Norms in Agent Systems I*, LNAI 3913, Springer, 18-34
6. Desel, J.; Juhas, G.; Lorenz, R. and Neumair, C. 2003. „Modelling and Validation with VipTool". LNCS 2678, 380-389.
7. Popova, V. and Sharpanskykh, A. 2006. "Process-Oriented Organization Modeling and Analysis Based on Constraints". Technical Report 062911AI, VUA, http://hdl.handle.net/1871/10545
8. Popova, V. and Sharpanskykh, A. 2007a. "Formal Modelling of Goals in Agent Organizations". In V. Dignum, F. Dignum, E. Matson (eds.), *Proceedings of the AOMS Workshop (joint with IJCAI 2007)*, 74-86
9. Popova, V. and Sharpanskykh, A. 2007b. "Formal analysis of executions of organizational scenarios based on process-oriented models". Technical Report 071601AI, VUA, http://hdl.handle.net/1871/10643
10. Sharpanskykh, A. and Treur, J. 2006. "Verifying Interlevel Relations within Multi-Agent Systems". In Brewka, G., Coradeschi, S., Perini, A., and Traverso, P. (eds.), *Proceedings of the 17th European Conference on Artificial Intelligence, ECAI'06*. IOS Press, 290-294

Chapter 5

A Framework for Formal Modeling and Analysis of Organizations [1]

Abstract. A new, formal, role-based, framework for modeling and analyzing both real world and artificial organizations is introduced. It exploits static and dynamic properties of the organizational model and includes the (frequently ignored) environment. The transition is described from a generic framework of an organization to its deployed model and to the actual agent allocation. For verification and validation of the proposed model, a set of dedicated techniques is introduced. Moreover, where most computational models can handle only two or three layered organizational structures, our framework can handle any arbitrary number of organizational layers. Henceforth, real-world organizations can be modeled and analyzed, as illustrated by a case study, within the DEAL project line.

1 Introduction

Recently computational modeling and analysis of organizations received a special attention in the areas of social science and artificial intelligence. In particular, organizations have proven to be a useful paradigm for analyzing and designing multi-agent systems [7, 10, 42]. Representation of a multi-agent system as an organization consisting of roles and groups can tackle major drawbacks concerned with traditional multi-agent models; e.g., high complexity and poor predictability of dynamics in a system [10]. As has been shown in [19], organizational structure can be used to limit

[1] This chapter appeared as Jonker, C.M., Sharpanskykh, A., Treur, J., Yolum, P.: A Framework for Formal Modeling and Analysis of Organizations, Applied Intelligence, 27(1), 49-66 (2007) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

the scope of interactions between agents, reduce or explicitly increase redundancy of a system, formalize high-level system goals, of which a single agent may be not aware, or enforce certain coordination mechanisms for efficient task execution.

Moreover, organizational research in social science has recognized the advantages of computational models; e.g., for analysis of structure and dynamics of real organizations. In particular, distributed simulation models were created for analyzing organizational adaptation processes [5, 34], social networks [38] and dynamic processes in different organization types. However, in general formal theories, approaches, and tools for designing computational models of organizations are still rare and most of them are dependant of specific social theoretical background (cf. the OrgCon tool for organizational design based on the contingency theory [4]). In this paper, we propose a new modeling approach for analyzing and formal modeling of real or artificial organizations (e.g., agent-based organizations), independent of any organizational theory from social science. This approach is based on a generic representation of organizations that comprises sets of interrelated roles, which are intentionally organized to ensure a desired (or required) pattern of activities. The approach has the following distinct features: (1) it addresses both organization structure and dynamics; (2) the approach has its formal foundation in an expressive order-sorted predicate language with properly defined syntax and semantics; (3) it allows multiple aggregation levels in an organization model; (4) the environment is explicitly incorporated in an organization model; (5) the approach provides formal techniques and tools for different types of analysis of organization models (by performing simulations and verification).

In the next Section, main principles for modeling and analyzing organizations are discussed and related with the new modeling approach. In Section 3, the basic concepts used for specifying an organization model are introduced. Section 4 discusses how an organization model can be specified in a formal manner. In Section 5, a set of dedicated validation and verification techniques are described. The proposed modeling and verification techniques will be illustrated by a running example from the area of logistics. The paper ends with a discussion in Section 6.

2 Principles for modeling and analyzing organizations

Modern organizations are characterized by their complex structure, dense information flows, and incorporation of information technology. To a large extent, the underlying organization model is responsible for how efficiently and effectively organizations carry out their tasks. In literature on organization theory, a range of theories and guidelines concerning the modeling and design of organizations are present [28, 30]. However, no operational general theories or formal models exist that are known to the authors. Scott [39] even stated that no general principles applicable to organizational modeling can be formulated. However, for certain specific organizational types standard modeling and design techniques may still be identified and formalized. In particular, Minzberg proposed a set of guidelines for modeling mechanistic types of organizations [28]. This type of organizations comprises systems of hierarchically linked job positions with clear responsibilities that use standard well-understood

technology and operate in a relatively stable (possibly complex) environment. In contrast to mechanistic (or functional) organizations, a substantial group of modern organizations are characterized by a highly dynamic, constantly changing, organic structure with non-linear behavior [29]. Although the structure and behavioral rules for such organizations can be hardly identified and formalized, nevertheless by performing agent-based simulations with changing attitudes of proactive agents useful insights into functioning of such organizations can be gained.

2.1 Two perspectives

In this subsection, we will briefly discuss two perspectives from which organizations are analyzed. The first perspective emerges from social sciences and the second originates from computational organization theory and artificial intelligence.

In social science theories, the structure of organizations is frequently specified as informal or semi-formal graphical representations [28, 30]. They can provide a detailed organization structure at an abstract level considered from a certain perspective (e.g., information flows, power and authority relations, allocation of resources). The disadvantages of such models are: (1) lack of generality and relations between different specific types of models, and (2) graphically depicted data can not be effectively processed, combined and analyzed. Furthermore, such approaches lack the means to represent the more detailed dynamics and to relate them to the structures present.

A class of models built based on the system dynamics theory allows formal representation of different aspects of organizational behaviour [12]. Organizational models specified in system dynamics are based on numerical variables and equations that describe how these variables change over time. Although such models can be computationally effective (i.e., used for simulations and computational analysis), nevertheless they still lack the ontological expressivity and the possibility for higher abstract (and, e.g., non-quantitative) representations that are needed to conceptualize wide range of relations and phenomena that exist in different types of organizations.

From computational organization theory and artificial intelligence, approaches have been developed that are able to capture both structural and dynamic aspects of organizations. Some of them are dedicated for analyzing particular aspects of an organization considered from a certain viewpoint (e.g., Petri-nets techniques used for modelling and analyzing business processes [8]). Although such approaches can be useful and efficient, the scope of their application is limited to a particular view on an organization, based on a limited number of concepts. Furthermore, techniques from the area of artificial intelligence have been applied for modelling and analyzing multi-agent organizations [3, 7]. In such organizational models (software, hardware or human) agents are allocated to roles that stand in certain relations to each other and often are described by sets of functionalities performed by an organization. Such models can be used for example for coordinating tasks execution in a multi-agent system [19], or for enforcing certain behaviours (e.g., normative systems) upon an agent system [41]. However, many of such models can handle only two or three levels of abstraction; i.e., the level of an individual role, to which an agent(s) will be eventually allocated, the level of a group composed of roles, and the overall

organization level, as in GAIA [42], MOISE [16, 20], MOCA [1], TOVE [13], Aaladin [11] and OperA [7]. In contrast, multiple levels and relations between them need to be described for the representation of complex hierarchical structures of modern organizations; e.g., mechanistic type of organizations [30]. One of the few exceptions known to authors as capable of representing hierarchical structures is a framework for modeling social structures in UML proposed in [33]. This framework allows the possibility of the iterative inclusion of groups represented by holonic agent structures into other groups as their members, thus building hierarchical structures. However, the framework does not provide a general mechanism for handling interactions between roles and groups of different aggregation levels that often occur in such hierarchical structures. Furthermore, there is no possibility to identify and formally specify how dynamics of a composite group is related to the dynamics of its members, which is a prerequisite for the (formal) analysis of behavior of such composite systems. Another framework that supports the hierarchical representation of a multi-agent system is based on teams of agents [17]. A team is a composite component, similar to a group in [33], which is characterized by a number of roles, enacted by agents and other teams. However, this framework lacks means for elaborated conceptual modeling of social structures, probably because its main focus is on the technical side of programming and implementation of multi-agent systems. By introducing for example a (formal) language for specifying dynamics of individual roles and teams in this framework, different interesting types of analysis of system dynamics could be enabled.

Some models (ISLANDER [9], OperA, [26]) consider organizations as electronic institutions; i.e., norms and global rules that govern an organization are explicitly defined. However, in many modern organic organizations with much individual autonomy, the normative aspects do not play a central role and are of minor importance for the prosperity of an organization. Furthermore, a temporary violation of certain norms is inevitable and even necessary in certain organizations.

Independent of the previous distinction in approaches, the importance of explicit modeling of interactions between agents and the environment is recognized (explicitly considered in SODA [32] and AUML [31]). Since most of the modern organizations are open systems that actively interact with the environment, both an organizational structure and behavior are contingent on the environmental conditions.

Moreover, for modeling in general, verification and validation of the models used or generated is of the utmost importance. This is no different for modeling organizations. However, this aspect of modeling organizations is frequently ignored; two of the exceptions are TROPOS [3] and ISLANDER.

2.2 A new perspective

In this paper, we propose an approach for formal modeling and analysis of organizations. It is highly suitable for mechanistic types of organizations with the explicitly defined structure and behavior (i.e., machine and professional bureaucracy), and divisionalized forms of organizations that consist of autonomous units with specialized and formalized inputs and outputs. Furthermore, this approach can also be

applied for modeling organic types of organizations, when extended with organizational change techniques.

The proposed, formal approach can capture both structural and dynamic aspects of the organization and, subsequently, has four advantages:

(1) Representation of organization structure (including specifications of actors (or roles), relations between them, and information flows) and dynamics by generalized (template) models and more specific instantiated (deployed) models.
(2) The means for simulations of different (agent-based) scenarios on the basis of a model and observing their results.
(3) Organization analysis by means of verifying static and dynamic properties (e.g., based on organizational performance indicators) against (formalized) empirical data, taken from real organizations, or against simulated scenarios.
(4) Diagnosis of inconsistencies, redundancies, conflicts, and errors in an organizational model by means of formal verification techniques (e.g., based on model checking [6]).

In the proposed model, organizations are specified as composite roles that can be refined iteratively into a number of (interacting) composite or simple roles, representing as many aggregation levels as needed. The refined role structures correspond to different types of organization constructs (e.g., groups, units, departments). By considering only role hierarchies we achieve the uniform representation of an organization structural model, which is still able to reflect all the major types of organization constructs. The proposed framework provides formal means for specifying structural relations between roles of the same and different aggregation levels.

Behaviour of roles at each aggregation level is defined by sets of dynamic properties specified using an expressive temporal logical language. In the proposed approach different types of dynamic properties are distinguished, which are capable to capture different aspects of organizational dynamics. Note that the behaviour of a composite role is not simply defined as a list of all dynamic properties of its subroles. The dynamics of a composite role may be characterized by properties that emerge from the dynamics of its subroles or represent a more abstracted view on the lower-level dynamics. Therefore, particularly in the design phase when role dynamic properties are identified and specified, inconsistencies and conflicts between the properties of roles of adjacent aggregation levels may occur. Mechanisms to deal with these conflicts can be found in Section 5 and are further developed in the context of the proposed approach.

It is important to stress that the organizational model can be specified, depicted and analyzed at each aggregation level separately. For example, since the whole organization is considered as one composite role, it can be used as a “black box” with formally specified input and output interfaces for modelling and analyzing of high-level inter-organizational processes. In such a way the scalability of an organization model and the proposed approach is achieved.

Moreover, global normative aspects of an organization that are usually specified by organizational policies are defined by static and dynamic properties of the role at the highest aggregation level, without recognizing them as special concepts and placing them on top of an organization.

In addition, the environment is considered as a special component of the organization model. The environment is populated by agents that under certain conditions may be allocated to organizational roles. Furthermore, the environment serves as a source of events for an organization.

The modeling method introduced in this paper incorporates two types of verification and validation techniques: role-centered and agent-centered, as will be discussed in Section 5. The introduction of these techniques is preceded by the introduction of the model itself in the next section and its formal specification in Section 4.

3 Organization Modeling Concepts

In this section, the concepts are introduced on which the organization modeling approach is founded. First, the specification of the organizational structure is described. A template model is generated, which encapsulates the structure of the organization. On all existing levels of aggregation, the behavior of an organization can be described. Taken together, this provides description of the behavior of an organization. In Section 3.2, it will be explained how such dynamic behavior can be specified. In Section 3.3, the transition from template model to deployed model will be discussed. The introduced modeling concepts will be gradually used to represent different aspects of the organizational structure and behavior of an organization from the area of logistics.

3.1 Organization structure

An organization structure reflects patterns of interactions in an organization and is described by relationships between roles at the same and at adjoining aggregation levels and between parts of the conceptualized environment and roles. The specification of an organization structure that constitutes a template model uses the following elements:

(1) A role represents a subset of functionalities, performed by an organization, abstracted from specific agents (or actors) who fulfill them.

Each role can be composed by several other roles, until the necessary detailed level of aggregation is achieved, where a role that is composed of (interacting) subroles, is called a composite role. At the highest aggregation level, the whole organization can be represented as one role. Such representation is useful both for specifying general organizational properties and further utilizing an organization as a component for more complex organizations. Each role has an input and an output interface, which facilitate in the interaction (communication) with other roles. Graphically, a role is represented as an ellipse with white dots (the input interfaces) and black dots (the output interfaces).

(2) An interaction link represents an information channel between two roles at the same aggregation level. Graphically, it is depicted as a solid arrow, which denotes the direction of possible information transfer.

(3) The conceptualized environment represents a special component of an organization model. The environment can be defined by a set of objects with certain properties and states and by causal relations between objects. On the one hand, agents allocated to organization roles are capable of observing states and properties of objects in the environment; on the other hand, they can act or react and, thus, affect the environment. We distinguish passive and active observation processes. For example, when some object is observable by an agent playing a role and the agent continuously keeps track of its state, changing its internal representation of the object if necessary, passive observation occurs. For passive observation, no initiative of a role or an agent is needed. Active observation is always concerned with the agent's (or role's) initiative. Similarly to roles, the environment has input and output interfaces, which facilitate in the interaction with roles of an organization. Graphically, the environment is depicted as a rectangle with rounded corners. For particular purposes the internal specification for the environment can be conceptualized using one of the existing world ontologies (e.g., CYC, SUMO, TOVE). However, despite the richness and the extensiveness of these ontological bases, more specific and refined types of concepts and relations are required for modeling particular types of organizations and environments.

(4) An environment interaction link represents an information channel between a role of a certain aggregation level and (a part of) the conceptualized environment represented at this aggregation level. Graphically, it is depicted as a dotted arrow, which denotes the direction of possible information transfer.

(5) An interlevel link connects a composite role with one of its subroles. It represents information transition between two adjacent aggregation levels. Graphically, it is depicted as a dashed arrow, which shows the direction of the interlevel transition.

To illustrate the introduced concepts to model the organizational structure and all the following components of an organization model consider a running example based on a case study from the area of logistics. This case study was done within the project DEAL (Distributed Engine for Advanced Logistics). For the project description, we refer to *http://www.almende.com/deal/*. A template organizational model was created, based on the informal description of the structure and functioning of the large Dutch logistics company. Only relevant to the actual delivery process actors (roles) and their properties are specified in this model. To secure anonymity of the company, the real names of the organizational units were substituted by general ones.

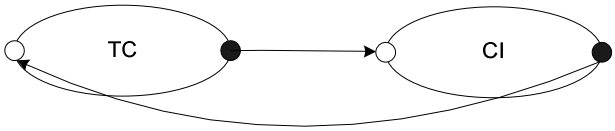


Fig. 1. Representation of the organization at abstraction level 1, which consists of role Transport Company (TC) and role Customer Interaction (CI)

At the highest aggregation level (level 0) the whole organization is represented as one role. At aggregation level 1, the organization consists of two interacting roles: TC and CI (see Fig.1; explanation for this and the following abbreviations and functional

descriptions are given in Table 1). Note, that the organizational model is depicted in a modular way; i.e., components of every aggregation level can be visualized and analyzed both separately and in relation to each other. Consequently, scalability of graphical representation of an organizational model is achieved.

Table 1. Role names, abbreviations, and descriptions for the organizational model in the case study

Role name	Abbreviation	Description
Transport Company	TC	Provides logistic services to customers
Customer Interaction	CI	Identifies interaction rules between a customer and the transport company
Strategy and Tactical Department	ST	Performs analysis and planning of company activities; considers complaints from customers; analyses the satisfaction level of a customer by means of surveys and questionnaires
Custom Relations Department	CR	Handles requests from customers
Operational Department	OP	Responsible for direct fulfillment of the order from a customer
Transport Company Representative	TCR	Mediator role between a customer and the transport company
Customer	C	Generates an order for the transport company; sends inquiries about the delivery status
Sales Person	SP	Assigns an order to a certain load manager, based on the type and the region of a delivery
Load Manager	LM	Assigns orders to suitable trucks and available drivers; assigns fleet managers to drivers; provides CR department with up-to-date information about delivery; provides a driver with instructions in case of a severe problem; informs CR department about possible delays with delivery
Fleet Manager	FM	Keeps constant contact with the assigned drivers; updates automatic support system with actual data on the delivery status; provides consultations for drivers in case of minor problems in transit
Driver	D	Delivers goods; informs a superior fleet manager about the delivery status; interacts (by means of observations and actions) with the conceptualized part of the environment
Environment	Env	Represents the conceptualized environment; in this example only a driver interacts with it

At aggregation level 2 role TC can be refined into three interacting roles: ST, CR, and OP (see Fig.2). All interactions with a customer are conducted within CI role. At aggregation level 2 it consists of two roles: TCR and C (see Fig. 2). Role TCR produces at its output messages from CR and ST departments of the transport company, i.e., CR and ST roles stand as company representatives in certain

interactions with a customer. Therefore, the input state of role TCR has influence on the output state of role CR and vice versa. The same holds for role ST.

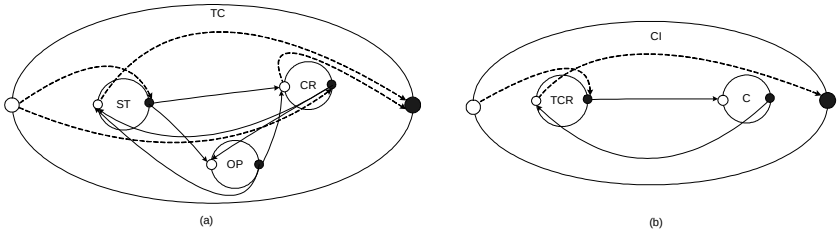


Fig. 2. Representation of (a) the Transport Company (TC) and (b) the Customer Interaction role (CI) at abstraction level 2

The structure of the operational department that is responsible for the direct fulfillment of the order from a customer is depicted at aggregation level 3 in Figure 3. It consists of interacting roles LM, FM, SP and D. Roles LM and SP are able to receive (or transmit) information from (or to) roles outside of role OP by means of interlevel links. Furthermore, in this model only role D interacts with the conceptualized environment.

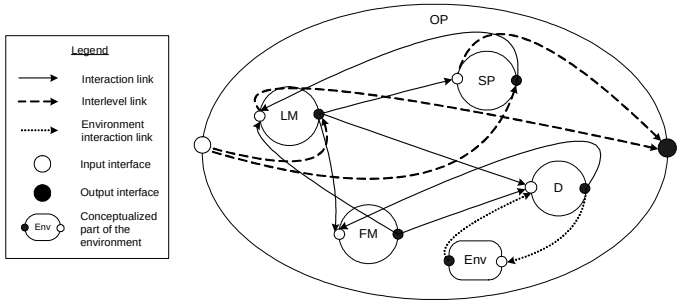


Fig. 3. Representation of the operational department at abstraction level 3

3.2 Organizational dynamics

At each aggregation level, it can be specified how the organization's behavior is assumed to be. To this end, organization dynamics are described by a dynamic representation, for each of the elements in an organization structure. The level of detail for specifying dynamics of an organization depends on its organizational type. Since the behavior of most mechanistic organizations is deterministic, dynamics for such organizations can only be modeled by a set of dynamic properties with high level of detail. In contrast, behavior of many organic organizations is defined loosely. Consequently, the dynamics of models for such organizations can be specified only partially; hence, actors (agents) can act autonomously.

The dynamics of each structural element are defined by the specification of a set of dynamic properties. We define five types of dynamic properties:

(1) ***A role property*** **(RP)** describes the relationship between input and output states of a role, over time. For example, in the settings of the logistics company from the running example, a role property of a truck driver (role D) can be defined as: if role Driver receives a request from his Fleet Manager to provide his coordinates, then role Driver will generate this data for his Fleet Manager.

(2) ***A transfer property*** **(TP)** describes the relationship of the output state of the source role of an interaction link to the input state of the destination role. Again, in the settings of the logistic company an example of a transfer property is the following: if role Customer generates an order to role Transport Company, then Transport Company will receive this order.

(3) ***An interlevel link property*** **(ILP)** describes the relationship between the input or output state of a composite role and the input or output state of its subrole. Note that an interlevel link is considered to be instantaneous: it does not represent a temporal process, but may give a different view on the same information state. Consider an example of such property: if role TCR obtains the customer order data at its input, then at the same time point role CI generates at its output a number assigned to the customer order in the automated information system.

(4) ***An environment property*** **(EP)** describes a temporal relationship between states or properties of objects of interest in the environment. Consider an environment property from the running example: If a severe incident happens with the truck involved in the delivery process, then it will cease the delivery.

(5) ***An environment interaction property*** **(EIP)** describes a relation either between the output state of the environment and the input state of a role (or an agent) or between the output state of a role (or an agent) and the input state of the environment. For example: if the information about a traffic jam on the way of role D is generated at the output of the environment, then role D will receive (observe) this information at its input.

3.3 Deployed model and agent allocation

The generic or template model of an organization provides abstracted information concerning its structure and functioning. However, for a more detailed analysis, a deployed model is needed. It is based on both unfolded generic relations between roles, as defined in the template model, and on creating new role instances. In such a way, role instances from the deployed model can be related to generic roles from a template model by means of the generalization relations. Moreover, different deployed models may be specified using the same template model of an organization for different purposes.

In the deployed model for the considered running example, all roles specified at abstraction levels 1 and 2 have one-to-one mapping to the role instances. While roles LM, FM, and D (defined at abstraction level 3) have multiple instances; e.g., LM and FM are represented differently in different geographical regions and, subsequently, different types of trucks and professional skills of drivers are required for different kinds of deliveries. The deployed model for the considered example (see Fig. 4) is created based on the template model by unfolding assigned_to and in_region relations between roles. For example, assigned_to(D2, FM1) denotes that a middle-size truck and his

driver (D2) are assigned to the fleet manager in eastern Europe (FM1) and the relation in_region(D1, LM1) specifies that both a big-size truck driver (D1) and a load manager (LM1) should belong to the same region in eastern Europe.

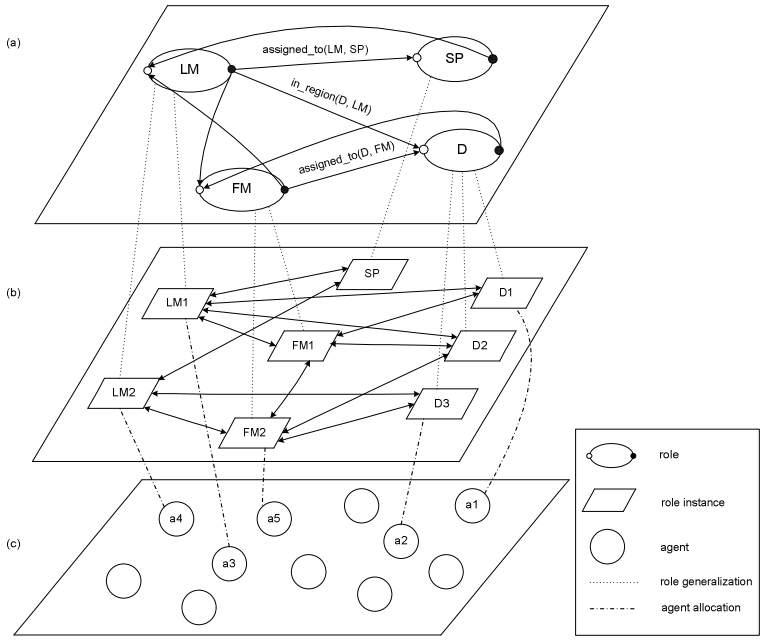


Fig. 4. The operational department of the transport company represented at abstraction level 3, with (a) the template model (b) the deployed model, and (c) agent allocation

The deployed model abstracts from the actual agent allocation but provides the detailed specifications for the behavior of role instances. Based on these specifications, a set of requirements is formulated for each role instance. These requirements (by restricting and defining behavior) are imposed onto the agents, who will eventually enact these roles. In the context of the running example one of the requirements imposed on a driver is that the agent should have a driver license of a certain type and acceptable results of medical tests.

Each agent is characterized by a set of capabilities that describe skills and credentials of an agent. An agent can be allocated to a role only when agent capabilities match the set of role requirements. For example, in order to enact role LM, an agent should have working experience as a senior manager in logistics for at least 3 years.

If, for some reason, an allocated agent is not capable of enacting a certain role anymore, dynamic reallocation of another agent will take place.

In some scenarios, a complex role can act as a single aggregated role and, thus, representing its constituting subroles. In such cases, an (aggregated) agent can be assigned to the complex role. In the literature [36, 37] aggregated (or composite) agents are often called *holons*. A holon is defined by a recursive model of agent

groups and appears as a single entity to the outside world. A holon may impose certain structures (i.e., types of relations) and behaviors on its agents, thus limiting their autonomy in certain aspects. Furthermore, a holon may be allocated to a simple (not composite) role, when the joint set of capabilities of agents of the holon satisfies the role requirements.

4 Formal Specification of the Organization Model

In the previous section, the elements of the organizational model were introduced. The current section provides the formal specification of them.

4.1 Structural properties

Structural properties describe elements of an organization structure introduced in Section 3.1 and relations between them.

As it has been shown above, in an organization model roles interact with other roles and the environment by means of input and output interfaces. These interfaces are described in terms of interaction (input and output) ontologies: a vocabulary or a signature specified in order-sorted logic that comprises finite sets of sorts, constants within these sorts, and relations and functions over these sorts. Generally speaking, an input ontology determines what types of information are allowed to be transferred to the input of a role (or of the environment), and an output ontology predefines what kinds of information can be generated at the output of a role (or of the environment). Roles and relations between them and the environment defined in a template model, as well as role instances and relations between them and the environment defined in a deployed model are specified using sorts and predicates from the structure ontology. This ontology includes sorts for all structural elements of an organization model (such as roles, different types of links, environment). The predicates for specifying organizational structure are defined over these sorts in Table 2. For example, in the settings of the logistics company from the running example, subroles Fleet Manager (FM) and Load Manager (LM) belong to the same composite role Operational department (OP). Formally: has_subrole(OP, FM) & has_subrole(OP, LM). Note that input and output ontologies of role instances are constructed by limiting and refining the ontologies of template roles based on which these role instances have been created.

In order to enable interaction between roles at the same aggregation level it is required that the ontologies of interacting roles contain common (or shared) elements (e.g., to specify the speech act s_act (e.g., inform, request, ask) from role-source r1 to role-destination r2 with the content message the predicate communicate_from_to(r1:ROLE, r2:ROLE, s_act:SPEECH_ACT, message:STRING) may be defined as a part of ontologies for both roles).

However, ontologies of roles connected by an interlevel link may not contain common elements. In this case the interlevel link is described by an ontology mapping between the corresponding elements of ontologies. Moreover, an ontology mapping associated with an interlevel link may be used for representing mechanisms of information abstraction. These mechanisms can be applied for transmitting (or

generating) partial, aggregated or generalized information to the input (or from the output) of a role.

Table 2. Ontology for formalizing organizational structure

Predicate	Description
is_role: ROLE	Specifies a role in an organization
has_subrole: ROLE x ROLE	For a subrole of a composite role
source_of_interaction: ROLE x INTERACTION_LINK	Specifies a source role of an interaction
destination_of_interaction: ROLE x INTERACTION_LINK	Specifies a destination role of interaction
interlevel_connection_from: ROLE x INTERLEVEL_LINK	Identifies a source role of an interlevel link
interlevel_connection_to: ROLE x INTERLEVEL_LINK	Identifies a destination role of an interlevel link
initiator_env_interaction: ROLE x ENVIRONMENT_INTERACTION_LINK	Specifies a role-initiator in interaction with the environment
recipient_env_information: ROLE x ENVIRONMENT_INTERACTION_LINK	Identifies a role-recipient of information from the environment
part_of_env_in_interaction: ENVIRONMENT x ENVIRONMENT_INTERACTION_LINK	Identifies the conceptualized part of the environment involved in interaction with a role
has_input_ontology: ROLE x ONTOLOGY	Specifies an input ontology for a role
has_output_ontology: ROLE x ONTOLOGY	Specifies an output ontology for a role
has_input_ontology: ENVIRONMENT x ONTOLOGY	Specifies an input ontology for the environment
has_output_ontology: ENVIRONMENT x ONTOLOGY	Specifies an output ontology for the environment
has_interaction_ontology: ROLE x ONTOLOGY	Specifies an interaction ontology for a role
has_interaction_ontology: ENVIRONMENT x ONTOLOGY	Specifies an interaction ontology for the environment
has_onto_mapping: INTERACTION_LINK x ONTO_MAPPING	Identifies an ontology mapping
to_be_observed: STATE_PROPERTY	Describes a state property that will be observed in the environment
observation_result: STATE_PROPERTY x BOOLEAN_VALUE	Determines if a certain state property holds in the environment
to_be_performed: ACTION	Specifies an action that will be performed in the environment

Often, structural properties are valid during the whole period of organization existence and can be considered as static. But in rapidly developing and adapting organizations, structural change processes gain special importance. Structural properties for such organizations get a temporal dimension and can be considered as a subclass of dynamic properties.

4.2 State and dynamic properties

The dynamics of an organization are defined by the specification of dynamic properties of its components that are formalized using the dynamic ontology (see Table 3) and belong to the following five classes: role properties, transfer properties, interlevel link properties, environment properties, and environment interaction properties. Each dynamic property represents a relation in time either between (input or output) states of roles or a (input or output) state of a role and a (input or output) state of the environment. States of roles and the environment are defined based on the corresponding ontologies for roles and the environment. More precisely, a state for ontology Ont is an assignment of truth-values to the set At(Ont) of ground atoms expressed in terms of Ont. The set of all possible states for state ontology Ont is denoted by STATES(Ont).

Table 3. Dynamics ontology for formalizing properties of an organization

Sort	**Description**
DYNPROP	Sort for the name of a dynamic property
DPEXPR	Sort for the expression of a dynamic property
Predicate	**Description**
has_dynamic_property: ROLE x DYNPROP	Specifies a role dynamic property
has_dynamic_property: INTERACTION_LINK x DYNPROP	Identifies a dynamic property for an interaction link
has_dynamic_property: ENVIRONMENT x DYNPROP	Identifies a dynamic property for the conceptualizedpart of the environment
has_dynamic_property: ENVIRONMENT_INTERACTION_LINK x DYNPROP	Identifies a dynamic property for an environment interaction link
has_expression: DYNPROPx DPEXPR	Specifies an expression for a dynamic property

A state property is defined by a formula over a state ontology. For example, communicate_from_to(TCR, customer, inform, order_state(ON, delay, customer_report)) is a state formula expressing the informative speech act in form of a customer report from role TCR to role Customer about the delay state of the order with the number ON.

Dynamic properties (e.g., for roles, environment, and links) are specified in the Temporal Trace Language (TTL) [22, 40], which is a variant of order-sorted predicate logic [27], and in the classification in Galton [14, 15] falls in the class of reified temporal logic.

TTL has some similarities with situation calculus [35] and event calculus [24]. To enable reasoning about the dynamic properties the language TTL includes special sorts, such as: TIME (a set of linearly ordered time points), STATE (a set of all state names of a system), TRACE (a set of all trace names; a trace or a trajectory can be thought of as a timeline with for each time point a state), and STATPROP (a set of all state property names).

Role or environment states are related to state properties via the satisfaction relation |=, formally defined as a binary infix predicate (or by holds as a binary prefix predicate): state(γ, t, output(r)) |= p (or holds(state(γ, t, output(r)), p)), which denotes that state property p holds in trace γ at time t in the output state of role r.

Both state(γ, t, output(r)) and p are terms of the TTL language. Here p is used not as a statement, but as a term for an object in the language which refers to a state proposition; this is called reification; cf. Galton [14, 15]. TTL terms are constructed by induction in a standard way for sorted predicate logic from variables, constants and functional symbols typed with TTL sorts. Dynamic properties are expressed by TTL-formulae inductively defined by:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds(v_1, u_1) is an atomic TTL formula.
(2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1 = \tau_2$ is an atomic TTL formula.
(3) If t_1, t_2 are terms of sort TIME, then $t_1 < t_2$ is an atomic TTL formula.
(4) The set of well-formed TTL-formulae is defined inductively in a standard way based on atomic TTL-formulae using boolean propositional connectives and quantifiers.

In the context of the running example consider the information distribution property defined for role OP called RP1(OP), specified at abstraction level 2. Informally, when a severe problem with some delivery occurs, OP should generate a message to CR about possible delay. Formally specified in TTL:

∀γ:TRACE ∀t1:TIME ∀T:TRUCK_TYPE ∀D:DRIVER ∀ON:ORDER_NUM state(γ, t1, environment))|= [truck_state(T, incident, severe_incident) ∧ truck_property(T, operated_by, D) ∧ order_property(ON, assigned_to, D)] ⇒
∃t2:TIME t2>t1 state(γ, t2, output(OP))|=communicate_from_to(OP, CR, inform, order_state(ON, delay, severe_incident)),

where Table 4 provides the description of the predicates.

More examples of dynamic properties formalized in TTL will be given in Section 5.1.

The specification of both structural and dynamic properties in TTL is supported by a dedicated editor [2, 23]. The organizational model for the running example that comprises both static and dynamic aspects has been specified in this software. Furthermore, the software tool enables model execution (simulation) under different environmental conditions (i.e., temporal sequences of events). As a result of simulation, a trace can be generated and visualized. A fragment of the trace generated for the organizational model constructed for the running example is illustrated in Figure 5. Here, the time frame is depicted on the horizontal axis. The names of predicates are shown on the vertical axis. A dark box on top of the line indicates that the predicate is true during that time period.

Table 4. Predicates for formalizing the dynamic properties used in the examples

Predicate	Description
communicate_from_to(r1:ROLE, r2:ROLE, s_act:SPEECH_ACT, message:STRING)	Specifies the speech act *s_act* (e.g., inform, request, ask) from role-source *r1* to role-destination *r2* with the content *message*
deliverable_object(on: ORDER_NUM, desc:STRING)	Assigns the order number *on* with the description *desc* to the object that has to be delivered
truck_property(trt:TRUCK_TYPE, operated_by, d:DRIVER)	Assigns the driver *d* to a truck of the type *trt*

order_property(on:ORDER_NUM, assigned_to, d:DRIVER)	Assignes the order *on* to the driver *d*
order_property(on:ORDER_NUM, deadline, d_value:INTEGER)	Identifies the deadline *d_value* for the order *on*
truck_state(trt:TRUCK_TYPE, st:STATE, descr:STATE_DESCRIPTION)	Denotes the state *st* with the state description *descr* of a truck of the type *trt*
order_state(on:ORDER_NUM, st:STATE, descr:STATE_DESCRIPTION)	Specifies the state *st* with the state description *descr* of the order with the number *on*

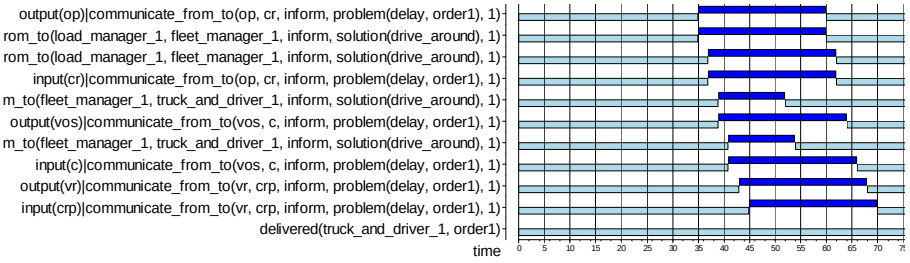


Fig. 5. An example of a visualized trace for the running example

4.3 Formalizing agent allocation principles

The formalization of agent allocation principles is performed in line with the formalization of the template and the deployed models, using the predicates specified in Table 5.

Table 5. Predicates for formalizing agent allocation principles

Predicate	Description
has_allocation_requirement: ROLE x REQUIREMENT	Specifies an allocation requirement for a role
has_capability: AGENT x CAPABILITY	Specifies a capability for an agent
allocated_to: AGENT x ROLE	Specifies an agent allocated to a role
corresponds_to: CAPABILITY x REQUIREMENT	Specifies an agent capability that corresponds to a role requirement

Generally, it is assumed that role requirements and agent capabilities are formulated using the same ontology, i.e., REQUIREMENT = CAPABILITY. However, if these ontologies are different, a necessary ontology mapping should be defined.

An agent can be allocated to a role if for every allocation requirement defined for the role the corresponding (equal in case of the same ontology) agent capability can be found. Formally:

allocated_to(a:AGENT, r: ROLE) ≡ ∀req:REQUIREMENT has_allocation_requirement(r, req) ⇒ [∃c:CAPABILITY has_capability(a, c) & corresponds_to(c, req)]

If after being allocated to the role, the agent looses one of his/her capabilities that correspond to the role requirements, then according to the rule above the current agent allocation will become false and the agent reallocation will be performed.

5 Verification and Validation

The model as introduced in this paper offers the means for both role-centered and agent-centered verification and validation. Role-centered verification techniques are dedicated for checking consistency and integrity of role-based organization models without allocating agents to roles. These techniques are considered in Section 5.1. Whereas agent-centered verification approaches are applied for checking certain (general) dynamic properties on execution of different scenarios with roles of an organization model allocated to (human) agents. These techniques are described in Section 5.2. Both role- and agent-centered verification techniques are illustrated by applying them for checking the organizational model from the running example.

5.1 Role-centered verification techniques

In this paper two types of role-centered verification techniques are considered: (1) verifying consistency of an organizational model by checking relations between dynamic properties of different aggregation levels using model checking techniques [6] and (2) checking if an organizational role-based model complies with certain general requirements (expressed as dynamic properties) in different role-based simulation scenarios. Let us consider these techniques more in detail.

Checking interlevel relations between dynamic properties of different aggregation levels

When an organization model is specified including dynamic properties at different aggregation levels, it is not automatically guaranteed that the properties defined at adjacent aggregation levels fit to each other. A verification process that addreses interlevel relations between properties at one aggregation level and properties of adjacent aggregation level (e.g., as in compositional verification) can reveal incompleteness or inconsistencies in an organization model. The verification approach based on model checking techniques proposed in [40] can be used for justifying such relations. According to this approach dynamic properties of the lower aggregation level components (i.e., roles, links and the environment) expressed in TTL form a model that by means of the techniques described in [40] can be translated into the input format of one of the existing model checkers, and can be further used for automated verification. For practical verification the model checker SMV [6] has been chosen. A property of the higher aggregation level is required by SMV to be represented as a temporal formula in CTL [6]. This property will be automatically checked against all the possible executions of the translated model of the lower aggregation level by performing model checking. In such a manner, it can be proven that a property of the higher aggregation level is a logical consequence of the model that comprises properties of a lower aggregation level.

Let us illustrate this technique by applying it to the running example. The information distribution property RP1(OP) of role OP defined at aggregation level 2 and specified in Section 4.2 is used as a property of the higher aggregation level. For the purpose of verification, this property is expressed in CTL as follows:

AG (truck_state_T_incident_severe_incident & truck_property_T_operated_by_D & order_property_A20_assigned_to_D →
AF
performing_action_preparation_output_OP_communicate_from_to_OP_CR_inform_order_state_A20_delay_severe_incident)

where ***A*** is a path quantifier defined in CTL, meaning "for all computational paths", ***G*** and ***F*** are temporal quantifiers that correspond to "globally" and "eventually" respectively.

The higher level property RP1(OP) can be logically related to the conjunction of dynamic properties of components at the lower aggregation level 3 in the following way:

EP1(Env, T, severe_incident) & EP2(Env, T) & EIP1(Env, D) & RP1(D) & TP1(D, FM) & RP2(FM) & TP2(FM, LM) & RP3(LM) & RP4(LM) & ILP1(LM, OP) ⇒ RP1(OP) (1)

The abbreviations for the dynamic properties and their arguments conform to the specification provided in Section 3. Let us consider the informal and formalized expressions for some of the properties from the relation (1) that hold for any trace γ (the complete specification for the dynamic properties in (1) is given in Appendix A and in [21]):

EP1(Env, T, severe_incident) Incident occurrence

Informal description:

In the environment a severe incident with the truck T occurs

Formalization:

∃t1:TIME state(γ, t1, environment)|= truck_state(T, incident, severe_incident)

EIP1(Env, D) Incident observation

Informal description:

If an incident happens with a truck, then a driver responsible for this truck will observe this incident

Formalization:

∀t1:TIME ∀T:TRUCK_TYPE ∀D:DRIVER ∀ins:INCIDENT state(γ , t1, environment))|= [truck_state(T, incident, ins) ∧ truck_property(T, operated_by, D)] ⇒ ∃t2 t2>t1 state(γ , t2, input(D)) |= observation_result(truck_state(T, incident, ins), true)

RP1(D) Request for incident solution

Informal description:

If a driver observes an incident with his truck, then s/he will react by generating a request for advice to his fleet manager

Formalization:

∀t1:TIME ∀T:TRUCK_TYPE ∀D:DRIVER ∀ins:INCIDENT ∀FM: FLEET_MANAGER state(γ , t1, input(D)) |= observation_result(truck_state(T, incident, ins), true) & state(γ, t1, environment) |= assigned_to(D, FM) ⇒ ∃t2 t2>t1 state(γ , t2, output(D)) |= to_be_performed(communicate_from_to(D, FM, ask, solution_for_problem(ins, T)))

ILP1(LM, OP) Generation of information about the state change of a delivery order object

Informal description:

If a load manager communicates information about the change of a delivery status to the customer relation role, then the operational department role transmits this information to the customer relation department role.

Formalization:

∀t1:TIME ∀LM: LOAD_MANAGER ∀ON:ORDER_NUM ∀st: STATE_TYPE ∀r: REASON state(γ, t1, output(LM))|= to_be_performed(communicate_from_to(LM, CR, inform, order_state(ON, st, r)))
⇒ ∃t2 t2>t1 state(γ, t2, output(OP))|= to_be_performed(communicate_from_to(OP, CR, inform, order_state(ON, st, r)))

By applying the algorithms and the dedicated software described in [40] to the specification that comprises all identified above properties defined at aggregation level 3 is transformed into the finite state transition system format required for performing model checking. Such a format consists of transition rules of the form [P → N], where P is a set of (predicate logic) atoms that are true in a current state and N is a set of atoms that will be true in the next state. For example, one of the transition rules from the obtained specification describes the generation of the memory state based on the observation of driver D at the time point t of the state property expressing that a severe incident happened with truck T:

present_time(t) & observed(input_D_truck_state_T_incident_severe_incident) ⇻
memory(t, observed(input_D_truck_state_T_incident_severe_incident))

The following transition rule expresses the persistency of the created memory state:

memory(t, observed(input_D_truck_state_T_incident_severe_incident)) ⇻
memory(t, observed(input_D_truck_state_T_incident_severe_incident))

The complete specification of the obtained finite state transition system for the considered example is given in Appendix B. The details of the procedure for transformation of a behavioral TTL specification into the finite state transition system format, and its application to the considered example are given in [21].

The automatic verification in the SMV model checking tool of the property RP1(OP) on the considered model showed that the previously identified logical relation (1) indeed holds. In general, the formal verification method of logical relations between dynamic properties of adjacent aggregation levels is useful for revealing missing premises or other shortcomings such as inconsistencies.

Checking global organizational properties with respect to a simulated role-based model

Another role-centered verification method is based on checking global organizational properties (or requirements) with respect to different executions of a role-based model by means of dedicated software. Such global organization properties are usually based on performance indicators of an organization, i.e., quantitative indicators that reflect the state, progress or performance of an organization (e.g., delivery time, customer notification time). By performing such verification inconsistencies and bottlenecks in an organization model can be detected.

Different executions (or execution traces) of a formally defined role-based organization model are obtained by performing simulations of different scenarios using a dedicated software environment [2]. Further the generated traces can be loaded into the verification environment, in which the formalized TTL properties can be checked on these traces.

Based on the formal organization model for the running example a simulation trace has been generated (given in Figure 5 partially), then the customer notification property has been checked on this trace.

Customer notification
Informal description:
Always if a severe problem occurs with the truck and the driver, who was fulfilling the order of some customer, then this customer should be notified about possible delay with delivery.
Formalization:

$\forall\gamma$:TRACE $\forall$t1:TIME $\forall$T:TRUCK_TYPE $\forall$D:DRIVER $\forall$ON:ORDER_NUM state(γ, t1, environment))|= truck_state(T, incident, severe_incident) $\wedge$ truck_property(T, operated_by, D) $\wedge$ order_property(ON, assigned_to, D) $\Rightarrow$ $\exists$t2:TIME t2>t1 $\exists$TCR:ROLE state(γ, t2, input(customer))|=communicate_from_to(TCR, customer, inform, order_state(ON, delay, customer_report))

An automatic verification confirmed that this property holds on the simulation trace.

5.2 Agent-centered verification technique

In this section an agent-centered verification technique is considered that is based on checking dynamic properties on a formalized empirical trace obtained by executing a particular scenario with roles of an organization model allocated to (human) agents. An empirical trace may be obtained from log-files of a company. If an empirical trace is given informally, the first step is to formalize it (by hand), using formal state ontologies. If it is already given in a formal form, the first step is to translate (e.g., automatically) the formal representation into one based on ontologies used in the organization model. Once such a trace is in the right formal form, it is possible to verify dynamic properties of the organization (including structural properties), using dedicated checking software as in the second role-centered verification technique.

As input for the verification software, a formalized trace and a formalized property have to be provided. Given such input, after automatic verification of the given property against the given trace, the software will generate a result (positive or negative). The positive decision confirms that the property holds with respect to the given trace. In case of a negative decision, the software explains why the property does not hold. In order to illustrate this method of verification, let us briefly consider the scenario reconstructed from empirical data of the transport company from the case study:

(1) A Customer places an order by means of a contact with TCR (CR department in this case) in CI.
(2) Inside TC this order is being transmitted from CR to OP.
(3) Within OP the order is distributed by SP to LM1.
(4) LM1 assigns the order to D1, D1 is associated with FM1 (see Fig. 4).
(5) D1 starts delivery, then after some time a severe incident occurs with his truck.
(6) D1 asks for help FM1, who is incapable of making a decision in this case.
(7) FM asks for a solution LM1, who decides to send another truck to proceed with delivery.
(8) Now D1 is reallocated to another truck and driver, who picks up goods and continues delivery.
(9) At the same time LM1 informs CR about possible delay with delivery.
(10) CR, who shares the same knowledge with TCR, informs the Customer about possible delay.
(11) D1 successfully finishes delivery and the Customer is being informed about that.

Using formal state ontologies (see Tables 2 and 3), we formalized this trace in the dedicated software environment. After that we identified several properties of interest that can be automatically verified against the trace. Let us consider two of them.

Delivery successfulness
Informal description:
The order has been fulfilled.

Formalization:

$\exists$t:TIME $\exists$O:ORDER_NUM state(γ, t, environment)|= order_state(O, delivered, final_report)

An automatic verification confirmed that this property holds against the formalized empirical trace.

Delivery accuracy
Informal description:
The order has been fulfilled on time.
Formalization:
$\exists$t:TIME $\exists$O:ORDER_NUM $\exists$d_value:integer state(γ, t, environment)|= order_state(O, delivered, final_report) $\wedge$ order_details(O, deadline, d_value) $\wedge$ d_value $\geq$ t

This property does not hold with respect to the trace. The next logical step in analysis of the causes for property failing would be to check if some incident occurred in transit. In case that a severe incident happened with the truck and the agent (a truck driver) was incapable of performing his role any more, the next step would be to verify whether or not enough time is available for a role reallocation. Subsequently, analysis of organization functioning can be continued until all inquiries about delivery are satisfied.

If an agent allocated to a role possesses individual attitudes and behavioral characteristics that are not explicitly identified in role requirements, however which may influence the execution of functions associated with the role, then dedicated analysis techniques for determining consequences of different agent architectures for role performance can be applied. These techniques are not considered in this paper and will be described elsewhere.

6 Discussion

Both in human society and for software agents, organizational structure provides the means to make complex, composite dynamics manageable. To understand and formalize how exactly organization structure constrains composite dynamics is a fundamental challenge in the area of organizational modeling. The modeling approach presented in this paper addresses this challenge. It concerns a method for formal specification of organizations, which can capture both structural and dynamic aspects of organizations and provides the means for (i) representation of organization structure, (ii) simulations of different scenarios, (iii) analysis of organization, verifying static and dynamic properties against (formalized) empirical data or simulated scenarios, (iv) diagnosis of inconsistencies, redundancies, and errors in structure and functioning. Additionally, the environment is integrated as a special component within the organization model.

Specification of organization structure usually takes the form of pictorial descriptions, in a graph-like framework. These descriptions often abstract from detailed dynamics within an organization. Specification of the dynamic properties of organizations, on the other hand, usually takes place in a completely different conceptual framework; these dynamic properties are often specified in the form of a set of logical formulae in some temporal language. The logical relationships express

the kind of relations between dynamics of parts of an organization, their interaction, and dynamic properties of the organization as a whole, which were indicated as crucial by Lomi and Larsen [25] in their introduction.

This paper shows how pictorial descriptions, in a graph-like framework, and a set of logical formulae in some temporal language can be combined in one organization modeling approach. Inspection can be done on the abstraction level preferred and both the pictorial and formal specifications of the dynamic properties can be inspected. Five essential types of dynamic properties characterizing behavior of main structural components of an organization model (including environment) are identified.

Due to the high expressivity of the introduced modeling (structural and behavioral) languages, the proposed framework creates the formal fundament for developing more specific types of models that describe certain particular aspects of organizations (e.g., goals and tasks). Such models can be built by introducing new particular specifications for these aspects in terms of sorts, predicates, and properties, which represent instantiations of general types of static and dynamic properties described in this paper. In future work different particular perspectives on organizations (e.g., performance-orients, goal-oriented, process-oriented) will be elaborated.

Furthermore, the approach proposed here supports formal specification and verification for both static and dynamic properties. This possibility is especially useful for diagnosis of inconsistencies, redundancies, and errors in structure and functioning of real organizations and providing recommendations for their improvement (e.g., by way of evaluating of performance indicators). Compared to most organization-oriented, multi-agent system, design approaches [1, 10, 11, 42], our model allows any number of aggregation levels in the organization model, which makes it more suitable for modeling and analyzing real organizations. While a role aggregation relation is considered to be crucial for representing an organizational model, other types of relations between roles should also be taken into account. For example, a role specified in a template model and its corresponding role instances defined in a deployed model are related by means of a generalization relation. Furthermore, even more general role templates (or classes), which possess essential characteristics of roles of a certain type (e.g., seller, vendor, customer), independent of any application domain, can be created. Different types of relations between such roles can be identified (e.g., aggregation, generalization, interaction). Then, based on roles classes and their relations libraries can be created that can be used for the specification of a template organizational model. Moreover, such libraries may be employed for constructing templates of different types of organizations. Both structural and dynamic aspects of different types of organizations should be reflected in such templates; for this formal languages introduced in this paper can be used. To identify the distinctive features of different organization types, agent-based models identified in [36] and the literature from organization theory [28, 30] are useful to consider.

Let us now consider a case in which agents show autonomous behavior, independent of (or sometimes conflicting to) organizational rules and goals. To tackle the forthcoming problems from such settings, further investigation of the relationships between formally predefined organizational model and agent autonomous behavior in settings of different types of organizations will be undertaken. The work on holonic structures [36, 37] may be relevant for further investigations on this question. By

applying the approach introduced in this paper the specifications of a (hierarchical) structure and dynamics can be developed, which describe a certain holon, or are imposed on agents within a holon. The specification of autonomous agent behavior takes place in a different conceptual framework, which, nevertheless, can be related (at least in ontological sense) to the modeling framework introduced in this paper. Then, by varying the types and flexibility of the (imposed) structures and behaviors (using for example the types described in [36]), and the level of agent autonomy, different types of organizations represented by multi-agent systems can be investigated. Furthermore, by applying analysis methods described in this paper the behavior of holons can be checked for compliance with the prescribed norms and other (global) properties of an organization.

In the case of highly dynamic organizations (e.g., self-organizing and organic organizations), organizational change is a crucial and frequent process. Due to their high complexity, such organizations are difficult to investigate. However, different simulation techniques can help in providing further insights into mechanisms of functioning of such organizations. For the latter purpose, research has been conducted based on the introduced formal model [18].

In conclusion, this paper introduced a new, formal, fully traceable method on modeling and analyzing (multi-agent) organizations. It comprises both static and dynamic aspects as well as environment representation. Hence, it provides the basis of a formal framework, which provides the means for both the design and for the automatic validation and verification of organizations.

Acknowledgments

This research was partially supported by the Netherlands Organization for Scientific Research (NWO) under project number 612.062.006. SenterNovem is gratefully acknowledged for funding the projects Cybernetic Incident Management (CIM) and Distributed Engine for Advanced Logistics (DEAL) that also funded this research partially. Further, we thank the reviewers for their detailed comments on the original manuscript.

References

1. M. Amiguet, J.-P. Mueller, J.-A. Baez-Barranco, and A. Nagy, “The MOCA Platform”, in Proc. of MABS, 2002, pp. 70-88.
2. T. Bosse, C.M Jonker, L. van der Meij and J.Treur “A Language and Environment for Analysis of Dynamics by Simulation, International Journal of Artificial Intelligence Tools”, 16: pp. 435-464, 2007.
3. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, “Tropos: An Agent-Oriented Software Development Methodology”, Journal of Autonomous Agent and Multi-Agent Systems, vol. 8(3), pp. 203-236, 2004.
4. R. M. Burton and B. Obel, Strategic Organizational Diagnosis and Design: Developing Theory for Application, Kluwer Academic Publishers: Dordrecht, 2004.

5. K. Carley and J.-S. Lee, “Dynamic Organizations: Organizational Adaptation in a Changing Environment”, Disciplinary Roots of Strategic Management Research, vol. 15, pp. 267-295, 1998.
6. E.M. Clarke, O. Grumberg, and D.A. Peled, Model Checking. MIT Press, 2000.
7. M. Dastani, J. Hulstijn, F. Dignum, and J-J. Meyer, “Issues in Multiagent System Development”, in Proc. of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems AAMAS'04, 2004, pp. 922-929.
8. R. David and H. Alla, Discrete, Continuous, and Hybrid Petri Nets. Springer-Verlag, 2005.
9. M. Esteva, D. Cruz, and C. Sierra, “ISLANDER: an electronic institutions editor”, in Proc. of the 1st International Conference on Autonomous Agents and Multiagent systems, 2002, pp. 1045-1052.
10. J. Ferber and O.Gutknecht, “A meta-model for the analysis and design of organizations in multi-agent systems”, in Proc. of Third International Conference on Multi-Agent Systems (ICMAS'98), IEEE Computer Society, 1998, pp. 128-135.
11. J. Ferber, O. Gutknecht, and F.Michel, “From Agents to Organizations: an Organizational View of Multi-Agent Systems”, in Proc. of 4th International Workshop AOSE, 2003, pp. 214-230.
12. J.W. Forrester, Industrial dynamics, Waltham, MA: Pegasus Communications, 1961.
13. M. Fox, M. Barbuceanu, M. Gruninger, and J. Lin, “An Organization Ontology for Enterprise Modelling”, in Simulating Organizations: Computational Models of Institutions and Groups, edited by M. Prietula, K. Carley and L. Gasser, Menlo Park CA: AAAI/MIT Press pp. 131-152, 1997.
14. A. Galton, Temporal Logic, in Stanford Encyclopedia of Philosophy, 2003. URL: http://plato.stanford.edu/entries/logic-temporal/#2.
15. A. Galton, “Operators vs Arguments: The Ins and Outs of Reification”, Synthese, vol. 150, pp. 415-441, 2006.
16. M. Hannoun, J.S. Sichman, O. Boissier, and C. Sayettat, “Dependence Relations between Roles in a Multi-Agent System: Towards the Detection of Inconsistencies in Organization”, in Proc. of MABS, 1998, pp. 169-182.
17. A. Hodgson, R. Roennquist, P. Busetta, and N. Howden, “Team Oriented Programming with SimpleTeam”, in Proc. of SimTecT 2000, Sydney, Australia, 2000, pp. 115-122.
18. M. Hoogendoorn, C.M. Jonker, M. Schut, and J. Treur, “Modelling the Organisation of Organisational Change”, in Proc. of the Sixth International Workshop on Agent-Oriented Information Systems, 2004, pp. 29-46. Extended version: Jounal of Computational and Mathematical Organisation Theory. In press, 2006.
19. B. Horling, V. Lesser, “A Survey of multi-agent organizational paradigms”, The Knowledge Engineering Review, Vol. 19(4), pp. 281-316, 2005.
20. J.F. Hubner, J.S. Sichman, O. Boissier, “A Model for the Structural, Functional and Deontic Specification of Organizations in Multiagent Systems”, in Proc. of SBIA, 2002, pp. 118-128.
21. C.M. Jonker, A. Sharpanskykh, J. Treur, and P. Yolum, “Verifying Interlevel Relations within Organizational Models”, Technical Report #TR-061909AI, Vrije Universiteit Amsterdam, 2006. URL: http://hdl.handle.net/1871/10210
22. C.M. Jonker and J. Treur, “A temporal-interactivist perspective on the dynamics of mental states”, Cognitive Systems Research Journal, 4(3), pp. 137-155, 2003.
23. C.M. Jonker, J. Treur, and W.C.A. Wijngaards, “A temporal-modelling environment for internally grounded beliefs, desires, and intentions”, Cognitive Systems Research Journal, 4(3), pp. 191-210, 2003.
24. R. Kowalski and M. Sergot, “A logic-based calculus of events”, New Generation Computing, 4, pp. 67-95, 1986.
25. A. Lomi and E.R. Larsen, Dynamics of Organizations: Computational Modeling and Organization Theories, AAAI Press, Menlo Park, 2001.

26.F. Lopez y Lopez, M. Luck, and M. d'Inverno, “A Normative Framework for Agent-Based Systems”, Computational and Mathematical Organization Theory, 12(2-3), pp. 227-250, 2005.
27.M. Manzano, Extensions of First Order Logic, Cambridge University Press, 1996.
28.H. Mintzberg, The Structuring of Organizations, Prentice Hall, Englewood Cliffs, 1979.
29.R.E. Miles, C.C. Snow, J.A. Mathews, and H.J. Coleman, “Organizing in the knowledge age: Anticipating the cellular form”, Academy of Management Executive, 11(4), pp. 7-20, 1997.
30.G. Morgan, Images of organizations, SAGE Publications, Thousand Oaks London New Delhi, 1996.
31.J. Odell, H.V.D. Parunak, and B. Bauer, “Extending UML for Agents”, in Proc. of Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, 2000, pp. 3-17.
32.A. Omicini, “SODA: Societies and infrastructures in the analysis and design of agent-based systems”, in Proc. of AOSE, 2000, pp. 185–193.
33.H.V.D. Parunak and J. Odell, “Representing Social Structures in UML”, in Proc. of Agent-Oriented Software Engineering II Workshop, edited by M. Wooldridge, G. Weiss, and P. Ciancarini, Lecture Notes on Computer Science, vol. 2222, Springer-Verlag, Berlin, pp. 1-16, 2002.
34.M. Prietula, L. Gasser, K. Carley, Simulating Organizations, MIT Press, 1997.
35. R. Reiter, Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical System, Cambridge MA: MIT Press, 2001.
36. M. Schillo, “Self-Organization and Adjustable Autonomy: Two Sides of the Same Coin?”, Connection Science, vol. 14 (4), 2003, pp. 345-360.
37. M. Schillo and D. Spresny, “Organization: The Central Concept for Qualitative and Quantitative Scalability”, in Socionics: Contributions to the Scalability of Complex Social Systems edited by K. Fischer and M. Florian, Lecture Notes in Artificial Intelligence, Berlin, vol. 3413, Springer, 2005, pp. 84-103.
38.J. Scott, Social Network Analysis: A Handbook, 2nd Ed. Newberry Park, CA: Sage, 2000.
39.W.R. Scott, Institutions and organizations, SAGE Publications, Thousand Oaks London New Delhi, 2001.
40.A. Sharpanskykh and J. Treur, “Verifying Interlevel Relations within Multi-Agent Systems”, in Proc. of the 17th European Conference on Artificial Intelligence, ECAI'06. IOS Press, 2006, pp. 290-294.
41.J. Vázquez-Salceda, H.M. Aldewereld, and F.P.M. Dignum, “Norms in multiagent systems: From theory to practice”, International Journal of Computer Systems Science & Engineering, vol. 20(4), pp. 225-236, 2005.
42.F. Zambonelli, N. R. Jennings, M. Wooldridge, “Developing multiagent systems: the Gaia Methodology”, ACM Transactions on Software Engineering and Methodology, vol. 12 (3), 2003, pp. 317-370.

Appendix A. The complete specification of dynamic properties from the running example

EP1(Env, T, severe_incident) Incident occurrence

Informal description:

In the environment a severe incident with the truck T occurs

Formalization:

∃t1:TIME state(γ, t1, environment)|= truck_state(T, incident, severe_incident)

EP2(Env, T) Stable information about the environnent

Informal description:

Role D (a driver) operates the truck T and is assigned to deliver the order A20; role D is assigned to the fleet manager FM, and FM is in the region of the load manager LM

Formalization:

∀t1:TIME state(γ, t1, environment) |= [truck_property(T, operated_by, D) ∧ order_property(A20, assigned_to, D) ∧ assigned_to(D, FM) ∧ in_region(FM, LM)]

EIP1(Env, D) Incident observation

Informal description:

If an incident happens with a truck, then a driver responsible for this truck will observe this incident

Formalization:

∀t1:TIME ∀T:TRUCK_TYPE ∀D:DRIVER ∀ins:INCIDENT state(γ , t1, environment))|= [truck_state(T, incident, ins) ∧ truck_property(T, operated_by, D)] ⇒ ∃t2 t2>t1 state(γ , t2, input(D)) |= observation_result(truck_state(T, incident, ins), true)

RP1(D) Request for incident solution

Informal description:

If a driver observes an incident with his truck, then s/he will react by generating a request for advice to his fleet manager

Formalization:

∀t1:TIME ∀T:TRUCK_TYPE ∀D:DRIVER ∀ins:INCIDENT ∀FM: FLEET_MANAGER state(γ , t1, input(D)) |= observation_result(truck_state(T, incident, ins), true) & state(γ, t1, environment) |= assigned_to(D, FM)
⇒ ∃t2 t2>t1 state(γ , t2, output(D)) |= to_be_performed(communicate_from_to(D, FM, ask, solution_for_problem(ins, T)))

TP1(D, FM) Request transfer to Fleet Manager

Informal description:

If a driver sends a request to his fleet manager, the fleet manager will receive this request

Formalization:

∀t1:TIME ∀D:DRIVER ∀FM: FLEET_MANAGER ∀req: REQUEST state(γ, t1, output(D)) |= to_be_performed(communicate_from_to(D, FM, ask, req)) & state(γ, t1, environment) |= assigned_to(D, FM)
⇒ ∃t2 t2>t1 state(γ , t2, input(FM)) |= observation_result(communicate_from_to(D, FM, ask, req))

RP2(FM) Request for solution propagation

Informal description:

If a fleet manager receives a request from a driver for advice to solve a severe problem, then s/he will propagate this request further to the regional load manager

Formalization:

∀t1:TIME ∀D:DRIVER ∀T:TRUCK_TYPE ∀FM: FLEET_MANAGER ∀LM: LOAD_MANAGER state(γ , t1, input(FM)) |= observation_result(communicate_from_to(D, FM, ask, solution_for_problem(severe_incident, T))) & state(γ, t1, environment) |= in_region(FM, LM)

⇒ ∃t2 t2>t1 state(γ , t2, output(FM)) |= to_be_performed(communicate_from_to(FM, LM, ask, solution_for_problem(severe_incident, T)))

TP2(FM, LM) Request transfer to Load Manager

Informal description:

If a fleet manager sends a request to a regional load manager, the regional load manager will receive this request

Formalization:

∀t1:TIME ∀FM: FLEET_MANAGER ∀LM: LOAD_MANAGER ∀req: REQUEST state(γ ,t1, output(FM)) |= to_be_performed(communicate_from_to(FM, LM, ask, req))
⇒ ∃t2 t2>t1 state(γ , t2, input(LM)) |= observation_result(communicate_from_to(FM, LM, ask, req))

RP3(LM) Change of a delivery status

Informal description:

If a load manager receives a request from a fleet manager for advice to solve a severe problem, then s/he officially identifies the incident as severe and changes into "delay" the state of the corresponding delivery order in the information system.

Formalization:

∀D:DRIVER ∀t1:TIME ∀FM: FLEET_MANAGER ∀T:TRUCK_TYPE ∀LM: LOAD_MANAGER ∀ON:ORDER_NUM state(γ , t1, input(LM)) |= observation_result(communicate_from_to(FM, LM, ask, solution_for_problem(severe_incident, T))) & state(γ, t1, environment) |= [order_property(ON, assigned_to, D) ∧ truck_property(T, operated_by, D)]
⇒ ∃t2 t2>t1 state(γ , t2, output(LM)) |= to_be_performed(change(order_state(ON, delay, severe_incident)))

RP4(LM) Informing CR about a delivery status

Informal description:

If a load manager changes a state of a delivery order object, then the information about this change is generated at the output of the load manager role for the customer relation role.

Formalization:

∀t1:TIME ∀LM: LOAD_MANAGER ∀ON:ORDER_NUM ∀st: STATE_TYPE ∀r: REASON state(γ , t1, output(LM)) |= to_be_performed(change(order_state(ON, st, r)))
⇒ ∃t2 t2>t1 state(γ, t2, output(LM))|= to_be_performed(communicate_from_to(LM, CR, inform, order_state(ON, st, r)))

ILP1(LM, OP) Generation of information about the state change of a delivery order object

Informal description:

If a load manager communicates information about the change of a delivery status to the customer relation role, then the operational department role transmits this information to the customer relation department role.

Formalization:

∀t1:TIME ∀LM: LOAD_MANAGER ∀ON:ORDER_NUM ∀st: STATE_TYPE ∀r: REASON state(γ, t1, output(LM))|= to_be_performed(communicate_from_to(LM, CR, inform, order_state(ON, st, r)))
⇒ ∃t2 t2>t1 state(γ, t2, output(OP))|= to_be_performed(communicate_from_to(OP, CR, inform, order_state(ON, st, r)))

Appendix B. The complete specification of the transition system from the running example

present_time(t) &
¬performing_action(preparation(output_LM_communicate_from_to_LM_CR_inform_order_state_A20_delay_severe_incident)) ⟶ present_time(t+1)
truck_state_T_incident_severe_incident & truck_property_T_operated_by_D ⟶
observed(input_D_truck_state_T_incident_severe_incident)
present_time(t) & observed(input_D_truck_state_T_incident_severe_incident) ⟶ memory(t,
observed(input_D_truck_state_T_incident_severe_incident))
memory(t, observed(input_D_truck_state_T_incident_severe_incident)) ⟶ memory(t,
observed(input_D_truck_state_T_incident_severe_incident))
present_time(t) & memory(t, observed(input_D_truck_state_T_incident_severe_incident)) & assigned_to_D_FM
⟶ qcprep1
present_time(t) & qcprep1 ⟶
preparation(output_D_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)
preparation(output_D_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T) ⟶
performing_action(output_D_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)
performing_action(output_D_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T) ⟶
observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)
present_time(t) &
observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T) ⟶
memory(t, observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T))
memory(t, observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T))
⟶ memory(t,
observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T))
present_time(t) & memory(t,
observed(input_FM_communicate_from_to_D_FM_ask_solution_for_problem_severe_incident_T)) &
in_region_FM_LM ⟶ qcprep2
present_time(t) & qcprep2 ⟶
preparation(output_FM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)
preparation(output_FM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T) ⟶
performing_action(output_FM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)
performing_action(output_FM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)
⟶ observed (input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)
present_time(t) & observed
(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T) ⟶ memory(t,
observed(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T))
memory(t, observed(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T))
⟶
memory(t, observed(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T))

present_time(t) & memory(t,
observed(input_LM_communicate_from_to_FM_LM_ask_solution_for_problem_severe_incident_T)) &
order_property_A20_assigned_to_D & truck_property_T_operated_by_D ⟶ qcprep3
present_time(t) & qcprep3 ⟶ preparation(change_order_state_A20_delay_severe_incident)
preparation(change_order_state_A20_delay_severe_incident) ⟶
performing_action(preparation(change_order_state_A20_delay_severe_incident))
performing_action(preparation(change_order_state_A20_delay_severe_incident)) ⟶
performing_action(preparation(output_LM_communicate_from_to_LM_CR_inform_order_state_A20_delay_severe_incident))
performing_action(preparation(output_LM_communicate_from_to_LM_CR_inform_order_state_A20_delay_severe_incident)) ⟶
performing_action(preparation(output_OP_communicate_from_to_OP_CR_inform_order_state_A20_delay_severe_incident))

Chapter 6

Authority and its Implementation in Enterprise Information Systems [1]

Abstract. The concept of power is inherent in human organizations of any type. As power relations have important consequences for organizational viability and productivity, they should be explicitly represented in enterprise information systems (EISs). Although organization theory provides a rich and very diverse theoretical basis on organizational power, still most of the definitions for power-related concepts are too abstract, often vague and ambiguous to be directly implemented in EISs. To create a bridge between informal organization theories and automated EISs, this paper proposes a formal logic-based specification language for representing power- (in particular authority) relations. The use of the language is illustrated by considering authority structures of organizations of different types. Moreover, the paper demonstrates how the formalized authority relations can be integrated into an EIS.

1 Introduction

The concept of *power* is inherent in human organizations of any type. Power relations that exist in an organization have a significant impact on its viability and productivity. Although the notion of power is often discussed in the literature in social studies [1, 2, 4, 5, 6, 7, 12, 13], it is only rarely defined precisely. In particular, power-related terms (e.g., control, authority, influence) are often used interchangeably in this literature. Furthermore, the treatment of power in different streams of sociology differs significantly. One of the first definitions for power in the modern sociology was given

[1] This chapter appeared as Sharpanskykh, A.: Authority and its Implementation in Enterprise Information Systems. In: Sadiq, S., Reichert, M., Schulz, K., Trienekens, J., Moller, C., and Kusters, J. (eds.), Proceeding of the 1st International Workshop on Management of Enterprise Information Systems, MEIS 2007, INSTICC Press, 33-43 (2007)

by Max Weber [20]: *Power is the probability that a person can carry out his or her own will despite resistance*. Weber and his followers (Dahl, Polsby) considered power as an inherently coercive force that implied involuntary submission and ignored the relational aspect of power. Other sociologists (Bierstedt, Blau) considered power as a force or the ability to apply sanctions [2]. Such view was also criticized as restrictive, as it did not pay attention to indirect sources and implications of power (e.g., informal influence in decision making) and subordinate's acceptance of power. Parsons [12] considered power as "*a specific mechanism to bring about changes in the action of organizational actors in the process of social interaction*".

Most contemporary organization theories explore both formal (normative, prescribed) and informal (subjective, human-oriented) aspects of power [4, 13, 17]. Formal power relations are documented in many modern organizations and, therefore, can be explicitly represented in models on which enterprise information systems (EISs) are based. The representation of formal power in EISs has a number of advantages. First, it allows a clear definition of rights and responsibilities for organizational roles (actors) and a power structure. Second, based on the role specifications, corresponding permissions for information, resources and actions can be specified for each role. Third, explicitly defined rules on power enable the identification of violations of organizational policies and regulations. Fourth, data about power-related actions (e.g., empowerment, authorization) can be stored in an EIS for the subsequent analysis.

For modeling of power relations the rich theoretical basis from social science can be used. Notably many modern EISs implement no or very simplified representations of power relations and mechanisms [3, 16]. One of the reasons is that concepts and definitions provided in social theories are often not operational and, therefore, cannot be directly used in automated information systems (EISs). To make use of these theoretical findings in EISs, power-related concepts should be formally grounded.

The first step to make the concept of power operational is to provide a clear and unambiguous meaning for it (or for its specific aspects). In this paper this is done by identifying the most essential characteristics and mechanisms of power described in different approaches and by integrating them into two broad categories: formal power (or authority) and informal power (or influence), which are described in Section 2. Further this paper focuses on the formal representation of authority, for which a formal language is described in Section 3. Moreover, Section 3 illustrates how the introduced formal language can be used to model authority systems of different types of organizations. Section 4 discusses the integration of formal authority relations into an automated EIS. Finally, the paper concludes with a discussion in Section 5.

2 Power, authority and influence

As in many contemporary social theories [4, 13], we assume that power can be practiced in an organization either through (formal) *authority* or through (informal) *influence relations*. Authority represents formal, legitimate organizational power by means of which a regulated normative relationship between a superior and a subordinate is established. Usually authority is attached to positions in organizations.

For example, authority of some managerial positions provides power to hire or to fire; to promote or to demote; to grant incentive rewards or to impose sanctions. In many approaches it is assumed that authority implies involuntary obedience from subordinates. Indeed, as authority has a normative basis that comprises formal, explicitly documented rules, it is expected that subordinates, hired by the organization, should be aware of and respect these rules, which implies the voluntary acceptance of authority.

All manifestations of power that cannot be explained from the position of authority fall into the category of influence. In contrast to authority, influence does not have a formal basis. It is often persuasive and implies voluntary submission. Some of the bases of influence are technical knowledge, skills, competences and other characteristics of particular individuals. Influence is often exercised through mechanisms of leadership; however, possession of certain knowledge or access to some resources, as well as different types of manipulation may also create influence. Influence may be realized in efforts to affect organizational decisions indirectly.

Although authority and influence often stem from different sources, they are often interrelated in organizations. For example, the probability of the successful satisfaction of organizational goals increases, when a strong leader (meaning a leader that has a great value of influence) occupies a superior position of authority. Furthermore, sometimes patterns of influence that frequently occur in an organization may become institutionalized (i.e., may become authority relations).

Modeling methods for authority and influence are essentially different. While authority relations are often prescriptive and explicitly defined, influence relations are not strictly specified and may vary to a great extent. Therefore, whereas authority relations can be generally represented in EISs, the specification of influence relations is dependant on particular (cognitive) models of agents that represent organizational actors. Relations between authority and influence can be studied by performing simulation with different types of agents situated in different organizational environments. The focus of this paper is on modeling of formal authority relations. Influence relations and relations between authority and influence will be considered elsewhere.

3. Authority: a Formal Approach

First, in Section 3.1 a formal language for specifying authority-related concepts and relations is introduced. Then, Section 3.2 discusses how the introduced language can be used for representing authority structures of organizations of different types.

3.1 A Formal Language

Simon [19] describes three contributions of authority for an organization: (1) the enforcement of responsibility, (2) the specialization of decision-making, and (3) the coordination of activity. Based on this and other theoretical findings that describe power, duties and responsibilities of organizational positions [11], a number of relations for the specification of formal authority can be identified. These relations are

defined on positions (or roles), without considering particular agents (individuals). The relations are formalized using the order sorted-predicate language [10].

We represent all activities of an organization (including decision making and personnel-related activities) by processes. Each organizational role is associated with one or more process. Roles may have different rights and responsibilities with respect to different aspects of the process execution. Furthermore, often several roles may potentially execute or manage certain processes. This is represented by the relation

is_authorized_for: r:ROLE x aspect: ASPECT x a:PROCESS, where aspect has one of the values {execution, monitoring, consulting, tech_des (making technological decisions), manage_des (making managerial decisions), user_defined_aspect}.

All types of decisions with respect to a particular process can be divided into two broad groups: *technological* and *managerial decisions* (inspired by [1]). Technological decisions concern technical questions related to the process content and are usually made by technical professionals. Managerial decisions concern general organizational issues related to the process (e.g., the allocation of employees, process scheduling, the establishment of performance standards, provision of resources, presenting incentives and sanctions). Managers of different levels (i.e., from the lowest level line managers to strategic apex (top) managers) may be authorized for making different types of managerial decisions varying from in scope, significance and detail. A particular decision type is specified as an aspect in the is_authorized_for relation. The same holds for technological decisions. Whereas *consulting* has a form of recommendation and implies voluntary acceptance of advices, decisions imposed on a role(s) that execute(s) the process are considered as imperatives with corresponding implications.

Authorization for *execution* implies that a role is allowed to execute the process according to existing standards and guidelines. Whenever a problem, a question or a deviation from the standard procedures occurs, the role must report about it to the role(s) authorized for making technological/managerial (depending on the problem type) decisions and must execute the decision(s) that will follow.

Monitoring implies passive observation of (certain aspects of) process execution, without intervention.

Notice that other aspects of process execution described in the managerial literature (e.g., control, supervision) can be represented as a combination of already introduced aspects. In particular, control can be seen as the conjunction of monitoring and making technological and/or managerial decisions aspects; supervision can be defined as the combination of consulting and control. Furthermore, the designer is given the possibility to define his/her own aspects and to provide an interpretation to them.

Although several roles in an organization may be authorized for a certain aspect related to some process, only one (or some) of them will be eventually (or are) responsible for this aspect. For example, the responsibility of a certain role with respect to the process execution means that the role is actually the one who will be performing the process and who holds accountability of the process execution. Furthermore, responsibility for the process execution implies allowance to use resources required for the process performance. The responsibility relation is specified as:

is_responsible_for: r:ROLE x aspect:ASPECT x a:PROCESS: process a is under responsibility of role r with respect to aspect (defined as for authorized_for)

Some roles are authorized to make managerial decisions for authorizing/disallowing other roles for certain aspects with respect to process execution. The authorization/ disallowance actions are specified by the following relations:

authorizes_for: r1:ROLE x r2:ROLE x aspect: ASPECT x a:PROCESS: role r1 gives the authority for aspect of process a to role r2.

disallows: r1:ROLE x r2:ROLE x aspect: ASPECT x a:PROCESS: role r1 denies the authority for aspect of process a for role r2.

However, to make a role actually responsible for a certain aspect of the process, another role besides the authority to make managerial decisions should also be the superior of the role with respect to the process. Superior-subordinate relations with respect to organizational processes are specified by: is_subordinate_of_for: r1: ROLE x r2: ROLE x a:PROCESS. Then, responsibility is assigned/retracted using the following relations:

assigns_responsibility_to_for: r1: ROLE x r2:ROLE x aspect: ASPECT x a:PROCESS: role r1 assigns the responsibility for aspect of process a to role r2.

retracts_responsibility_from_for: r1: ROLE x r2:ROLE x aspect: ASPECT x a:PROCESS: role r1 retracts responsibility from role r2 for aspect of process a.

Using these relations superiors may delegate/retract (their) responsibilities for certain aspects of processes execution to/from their subordinates, and may restrict themselves only to control and making decisions in exceptional situations.

In [7] control over resources is identified as an important source of power. Therefore, it is useful to identify explicitly which roles control resources by means of the relation has_control_over: r1: ROLE x res:RESOURCE. In the proposed modeling framework the notion of resource includes both tangible (e.g., materials, tools, products) and abstract (information, data) entities.

Our treatment of authority is different from both formal approaches that consider authority as an attribute or a property inherent in an organization [6, 20] and from the human-relation view that recognizes authority as an informal, non-rational and subjective relation (e.g., Follett, Mayo [4]). As many representatives of converging approaches (e.g., C.I. Barnard, Simon [19]) we distinguish between the formal authority prescribed by organizational policies and actual authority established between a superior and his/her subordinate in the course of social interactions. In the latter case a special accent lies on the acceptance of authority by a subordinate. In [4] different cases of the authority acceptance are discussed: orders anticipated and carried out (anticipation); acceptance of orders without critical review; conscious questioning but compliance (acceptance of authority); discusses but works for changes; ignores, evades, modifies orders (modification and evasion); rejection of authority (appeals to co-workers or higher rank for support). Depending on the organizational type, varying administrative sanctions may be applied in case an employee does not accept an authoritative communication, when he/she: (a) correctly understands/interprets this communication; (b) realizes that this communication complies with formal organizational documents and/or is in line with organizational goals; (c) is mentally and physically able to perform the required actions. In many

modern organizations rewards and sanctions form a part of authority relation, thus, explicitly defined:

grants_reward_to_for: r1: ROLE x r: REWARD x r2: ROLE x reason: STRING: role r1 grants reward r to role r2 for reason

imposes_saction_on_for: r1: ROLE x s: SANCTION x r2: ROLE x reason: STRING: role r1 imposes sanction s to role r2 for reason

Specific conditions (e.g., temporal, situational) under which authority relations may be created/maintained/dissolved are defined by executable rules expressed by logical formulae. The format and specification of these rules will be discussed in Section 4.

3.2 Modeling Authority Relations in Different Types of Organizations

Authority is enforced through the organizational structure and norms (or rules) that govern the organizational behavior. In general, no single authority system can be equally effective for all types of organizations in all times. An organizational authority system is contingent upon many organizational factors, among which organizational goals; the level of cohesiveness between different parts of an organization, the levels of complexity and of specialization of jobs, the level of formalization of organizational behavior, management style (a reward system, decision making and coordination mechanisms), the size of an organization and its units. Furthermore, the environment type (its uncertainty and dynamism; the amount of competitors), as well as the frequency and the type of interactions between an organization and the environment exert a significant influence upon an organizational authority structure.

In the following it will be discussed how authority is realized in some types of (mostly industrial) organizations and how it can be modeled using relations introduced in the previous Section 3.1.

Authority in small firms of the early industrial era was completely exercised by their owners through mechanisms of direct personal control. Firm owners were managers and technical professionals at the same time, and, therefore, had authority and responsibility for all aspects related to processes, except for their execution, responsibility for which was assigned to hired workers. The owners controlled all resources. Currently similar types of organizations can be found in family business and small firms.

With the growth of industry, which caused joining of small firms into larger enterprises, owners were forced to hire subcontractors, who took over some of their managerial functions. This can be modeled as assigning responsibility to subcontractors by the owner for some managerial and technological decisions, as well as monitoring and consulting of workers with respect to some processes execution. The owner reserved often the right to control for himself, which included granting rewards and imposing sanctions to/on subcontractors and workers, realized through superior-subordinate relations. Organizational resources were usually controlled by the owner.

Large industrial enterprises of XX century are characterized by further increase in number of managerial positions structured hierarchically by superior-subordinate

relations. Such organizations are often defined as mechanistic [17] and have the following typical characteristics: strong functional specialization, a high level of processes formalization, a hierarchical structure reinforced by a flow of information to the top of the hierarchy and by a flow of decisions/orders from the top. Responsibilities were clearly defined for every position in a hierarchy. In most organizations of this type responsibility for execution was separated from responsibilities to make decisions. Managerial positions differed in power to make decisions depending on the level in the hierarchy. Often, technological decisions were made by managers of lower levels (or even by dedicated positions to which also execution responsibilities were assigned), whereas managerial decisions were made by managers at the apex. In many of such organizations managers at the apex shared responsibility for making (some) decisions with lower-level managers. Therefore, decisions that were usually proposed by lower level managers had to be approved by the apex managers. Initially such enterprises operated in relatively stable (however, sometimes complex) environmental conditions that reinforced their structure. However, later in the second half of XX century to survive and to achieve goals in the changed environmental conditions (e.g., a decreased amount of external resources; increased competition; diversification of markets) enterprises and firms were forced to change their organizational structure and behavior. In response to the increased diversity of markets, within some enterprises specialized, market-oriented departments were formed. Such departments had much of autonomy within organizations. It was achieved by assigning to them the responsibility for most aspects related to processes, which created products/services demanded by the market. Although department heads still were subordinates of (apex) manager(s) of the organization, in most cases the latter one(s) were restricted only to general performance control over departments. Often departments controlled organizational resources necessary for the production and had the structure of hierarchical mechanistic type.

Although a hierarchical structure proved to be useful for coordination of activities of organizations situated in stable environments, it could cause significant inefficiencies and delays in organizations situated in dynamic, unpredictable environmental conditions. For example, in the domain of incident management unforeseen environmental circumstances may require a quick reaction from roles at a lower level of a power hierarchy without involvement of the established authority channels. Furthermore, the formalization and excessive control over some (e.g., creative and innovative) organizational activities often can have negative effects on productivity. Nowadays, large enterprises often create project teams or task forces that are given complex, usually innovative and creative tasks without detailed descriptions/prescriptions. As in the case with departments, teams are often assigned the responsibility to make technological and (some) managerial decisions and are given necessary resources to perform their tasks. Usually teams have highly cohesive plain structures with participants selected from different organizational departments based on knowledge, skills and experience required for the processes assigned to these teams. Although many teams implement informal communication and participative decision making principles [9], also formal authority relations can be found in teams. In particular, in some project teams superior-subordinate relations exist between the team manager and team members. In this case, whereas

responsibility for making technological decisions is given to team members, the responsibility for most managerial decisions is assigned to the team manager. Then, the members of such teams, being also members of some functional departments or groups, have at least two superiors. In other teams the team manager plays the integrator role and does not have formal authority over team members. In this case the responsibility for decisions made by a team lies on all members of the team. Sometimes to strengthen the position of a team manager, s/he is given control over some resources (e.g., budgets) that can be used, for example, to provide material incentives to the team members.

The principles on which teams are built come close to the characteristics of the organic organizational form [17]. Some of such organizations do not have any formal authority structure, other allow much flexibility in defining authority relations between roles. In the former case formal authority is replaced by socially created informal rules. In the latter case, authority may be temporally provided to the role that has the most relevant knowledge and experience for current organizational tasks. In many organic organizations formal control and monitoring are replaced by informal mutual control and audit. For the investigation of dynamics of organic organization, informal aspects such as influence, leaderships, mental models of employees are highly relevant, which will be discussed elsewhere. Often interactions between organic organizations (e.g., of network type) are regulated by contracts. Usually contracts specify legal relationships between parties that explicitly define their rights and responsibilities with respect to some processes (e.g., production, supply services). Several organizations may be involved in the process execution (e.g., supply chains for product delivery); therefore, it is needed to identify particular aspects of responsibility in contracts for such processes. The introduced language may be used for specifying such responsibilities and their legal consequences through reward/sanctions mechanisms.

4. Integration of Authority Relations into an EIS

In our previous work a general framework for formal organizational modeling and analysis is introduced [15]. It comprises several perspectives (or views) on organizations. In particular, *the performance-oriented view* [15] describes organizational goal structures, performance indicators structures, and relations between them. *The process-oriented view* [14] describes task and resource structures, and dynamic flows of control. In *the agent-oriented view* different types of agents with their capabilities are identified and principles for allocating agents to roles are formulated. Concepts and relations within every view are formally described using dedicated formal predicate-based languages. The views are related to each other by means of sets of common concepts. The developed framework constitutes a formal basis for an automated EIS.

To incorporate the authority relations introduced in this paper into this framework, both syntactic and semantic integration should be performed. The syntactic integration is straightforward as the authority relations are expressed using the same formal basis (sorted predicate logic) as the framework. Furthermore, the authority

relations are specified on the concepts defined in the framework (e.g., tasks, processes, resources, performance indicators). For the semantic integration rules (or axioms) that attach meaning, define integrity and other types of organization constraints on the authority relations should be specified. A language for these rules is required to be (1) based on the sorted predicate logic; (2) expressive enough to represent all aspects of the authority relations; (3) executable, to make constraints (axioms) operational. Furthermore, as authority relations are closely related to dynamic flows of control that describe a temporal ordering of processes, a temporal allocation of resources etc., a language should be temporally expressive. A language that satisfies all these requirements is the Temporal Trace Language (TTL) [8]. In [18] it is shown that any TTL formula can be automatically translated into executable format that can be implemented in most commonly used programming languages.

TTL allows specifying a temporal development of an organization by a trace. A trace is defined as a temporally ordered sequence of states. Each state corresponds to a particular time point and is characterized by a set of state properties that hold in this state. State properties are formalized in a standard predicate logic way [10] using state ontologies. A state ontology defines a set of sorts or types (e.g., ROLE, RESOURCE), sorted constants, functions and predicates.

States are related to state properties via the formally defined satisfaction relation |=: state(γ, t) |= p, which denotes that state property p holds in trace γ at time t. Dynamic properties are specified in TTL by relations between state properties. For example, the first axiom on the authority relations expresses that roles that are responsible for a certain aspect related to some process should be necessarily authorized for this:

Ax1: $\forall$r ROLE $\forall$a:PROCESS $\forall$aspect:ASPECT $\forall\gamma$:TRACE $\forall$t:TIME state(γ, t) |= [responsible_for(r, aspect, a) $\Rightarrow$ authorized_for(r, aspect, a)]

Another axiom expresses the transitivity of the is_subordinate_of_for relation: r1: ROLE x r2: ROLE x a:PROCESS:

Ax2: $\forall$r1, r2, r3: ROLE $\forall$a:PROCESS $\forall\gamma$, t state(γ, t) |= [is_subordinate_of_for(r2, r1, a) $\wedge$ is_subordinate_of_for(r3, r2, a)] $\Rightarrow$ is_subordinate_of_for(r3, r1, a)]

One more axiom (**Ax3**) that relates the interaction (communication) structure of an organization with its authority structure based on superior-subordinate relations expresses that there should be specified a communication path between each superior role and his/her subordinate(s). Such a path may include intermediate roles from the authority hierarchy and may consist of both interaction and interlevel links.

The following axiom expresses that only roles that have the responsibility to make managerial decisions with respect to some process are allowed to authorize other roles for some aspect of this process:

Ax4: $\forall$r1,r2:ROLE $\forall$a:PROCESS $\forall$asp:ASPECT $\forall\gamma$, t state(γ, t) |=
[authorizes_for(r1, r2, asp, a) $\Rightarrow$ is_responsible_for(r1, manage_des, a)]

In general, rules that describe processes of authorization, assigning/retracting of responsibilities may have many specific conditions. However, to assign responsibility for some aspect of a process a role should necessarily have at least the responsibility to make managerial decisions and be the superior (with respect to this process) of a role, to which the responsibility is assigned. All other conditions may be optionally specified by the designer. Responsibility may be assigned on a temporal basis. To

specify that the responsibility of role r1 for aspect asp of process a assigned by role r2 holds in all states that correspond to time points in the time interval limit, a responsibility persistency rule should be defined:

C1: ∀asp: ASPECT ∀r1,r2:ROLE ∀a:PROCESS ∀γ, ∀t1, t2:TIME state(γ, t1) |= is_responsible_for(r1, asp, a) & state(γ, t2) |= assigns_responsibility_to_for(r2, r1, asp, a) & (t1-t2) < limit
⇒ state(γ, t1+1) |= is_responsible_for(r1, asp, a)

Using concepts and relations from other organizational views, more complex constraints related to formal authority can be described. For example, "the total amount of working hours for role r1 should be less than a certain limit":

C2: sum([a:PROCESS], case(∃t1 state(γ, t1) |= is_responsible_for(r1, execution, a), a.max_duration, 0)) < limit

This property can be automatically verified every time when roles are assigned additional responsibilities for some processes. This is particularly useful in matrix organizations [17], in which roles often combine functions related to different organizational formations (departments, teams), and, as a result, their actual workload may not be directly visible.

Another constraint expresses that when the execution of a process begins, for each of the basic aspects for this process (execution, tech_des, and manage_des) a responsible role should be assigned:

C3: ∀a:PROCESS ∀γ, t state(γ, t) |= process_started(a)
⇒ ∃r1,r2,r3: ROLE state(γ, t) |= [is_responsible_for(r1, manage_des, a) ∧
is_responsible_for(r2, tech_des, a) ∧ is_responsible_for(r3, execution, a)]

Another example is related to rewards/sanctions imposed on a role depending on the process execution results. As shown in [15], performance indicators (PIs) may be associated with organizational processes that represent performance measures of some aspects of the tasks execution. Depending on the PIs values, a company may have regulations to provide/impose some rewards/sanctions for roles (agents) responsible for the corresponding processes. Although such rules are rarely completely automated, still an EIS may signal to managers about situations, in which some rewards/sanctions can be applied. For example, the system may detect and propose a reward granting action to the manager, when a role has been keeping the values of some PI(s) related to its process above a certain threshold for some time period [period_start, period_end]. In TTL:

C4: ∀γ, t1 t1 ≥ period_start & t1 ≤ period_end & state(γ, t1) |= [is_responsible_for(r2, execution, a1) ∧ measures(PI1, a1) ∧ is_subordinate_of_for(r2, r1, a1) ∧ PI1.value > limit]
⇒ state(γ, period_end+1) |= grants_reward_to_for(r1, bonus_5_procent, r2, excellent_performance_of_a1)

The axioms Ax1-Ax4 can be checked on a specification of organizational formal authority relations. To this end, simple verification algorthims have been implemented. Whereas the constraints C1-C4 and similar to them need to be checked on actual executions of organizational scenarios (e.g., traces obtained from an EIS). An automated method that enables such types of analysis is described in [14].

Furthermore, the identified rules can be used to determine for each user of an EIS relevant to him/her information and a set of allowed actions that are in line with

his/her (current) responsibilities defined in the system. Moreover, (possible) outcomes of each action of the user can be evaluated on a set of (interdependent) authority-related and other organizational constraints, and based on this evaluation the action is either allowed or prohibited.

5. Discussion

This paper makes the first step towards defining the formal operational semantics for power-related concepts (such as authority, influence, control), which are usually vaguely described in organization theory. In particular, this paper addresses formal authority, different aspects of which are made operational by defining a dedicated predicate logic-based language. It is illustrated how the introduced relations can be used for representing authority structures of organizations of different types.

Further, the paper addressed the integration of the formalized authority relations into an EIS. To this end, both syntactic and semantic integration have been considered. Syntactic integration is performed in a straightforward way, whereas for semantic integration rules (or axioms) have been defined using Temporal Trace Language. In particular, these rules attach meaning to different concepts and relations of authority and define the consistency of an authority structure. Thus, based on these rules verification of specifications of organizational authority relations can be performed.

Modern enterprises can be described along different dimensions/views: e.g., human-oriented, process-oriented and technology-oriented. However, most of the existing EISs focus particularly on the process-oriented view. An extension of the models on which EISs are built with concepts and relations defined within the human-oriented view allows conceptualizing more static and dynamic aspects of organizational reality, thus, resulting in more feasible enterprise models. Among the relations between human actors authority deserves a special attention, as it is formally regulated and may exert a (significant) influence on the execution of enterprise processes. This paper illustrates how the concepts and relations of authority can be formally related to other organizational views, thus resulting into an expressive and versatile enterprise model.

In the future it will be investigated how the proposed authority modeling framework can be applied for the development of automated support for a separation task (i.e., maintaining a safe distance between aircrafts in flight) in the area of air traffic control. Originally this task was managed by land controllers, who provided separation instructions for pilots. With the increase of air traffic, the workload of controllers rose also. To facilitate the controllers's work, it was proposed to (partially) delegate the separation task to pilots. This proposal found supporters and opponents both among controllers and pilots. The resistance to a large extent was (is) caused by ambiguity and vagueness of issues related to power mechanisms. Such questions as "whom to blame when an incident/accident occurs?", "which part of the task may be delegated?", "under which environmental conditions the task can be delegated?" still remain open. By applying the framework proposed in this paper one can precisely define responsibilities of both controllers and pilots and conditions under which the

responsibility can be assigned/retracted. Notice that these conditions may include relations from different views on organizations (e.g., “current workload is less than x”, “has ability a”), which allows a great expressive power in defining constraints.

References

1. Bacharach, S.B. and Aiken, M.: Communication in administrative bureaucracies. Academy of Management Journal, 18 (1977) 365-377
2. Blau, P.M. and Scott, W.R.: Formal Organizations.Chandler Publishing (1962)
3. CIMOSA – Open System Architecture for CIM; ESPRIT Consortium AMICE, Springer-Verlag, Berlin (1993)
4. Clegg, S.R.: Frameworks of Power. London: Sage (1989)
5. Friedrich, C.J. (ed.) Authority. Cambridge: Harvard University Press (1958)
6. Gulick, L.H. and Urwick, L.F. (eds.) Papers on the Science of Administration. Institute of Public Administration, New York (1937)
7. Hickson, D.J., Hinings,C.R., Lee,C.A., Schneck, R., Pennings, J.M.: A strategic contingency theory of intraorganizational power, Administrative Science Quarterly, 16 (1971) 216-229.
8. Jonker, C.M., Treur, J.: A temporal-interactivist perspective on the dynamics of mental states. Cognitive Systems Research Journal 4 (2003) 137-155
9. Lansley, P., Sadler, P.J., Webb, T.D.: Organization Structure, Management Style and Company Performance, Omega, London (1975)
10. Manzano, M.: Extensions of First Order Logic. Cambridge University Press (1996)
11. Mintzberg, H.: The Structuring of Organizations. Prentice Hall, Englewood Cliffs (1979)
12. Parsons T.: The institutionalization of Authority. In: Max Weber, The Theory of Social and Economic organization. Oxford University Press, New York (1947)
13. Peabody, R.L.: Organizational authority: superior-subordinate relationships in three public service organizations. Atherton Press, New York (1964)
14. Popova, V., Sharpanskykh, A.: Process-Oriented Organization Modeling and Analysis Based on Constraints. Technical Report 062911AI, VUA, http://hdl.handle.net/1871/10545
15. Popova, V., Sharpanskykh, A.: Modelling Organizational Performance Indicators. In: Barros, F. et al. (eds) Proc. of IMSM’07 conference (2007) 165–170
16. Scheer, A.-W.; Nuettgens. M.: ARIS Architecture and Reference Models for Business Process Management. In: van der Aalst, W.M.P. et al. (eds.), LNCS 1806, Berlin (2000) 366-389
17. Scott, W.R.: Institutions and organizations. SAGE Publications, Thousand Oaks (2001)
18. Sharpanskykh, A. and Treur, J.: Verifying Interlevel Relations within Multi-Agent Systems. In Proc. of the 17th European Conf. on AI, ECAI'06. IOS Press, 2006, pp. 290-294.
19. Simon, H.A.: Administrative Behavior. 2nd edn. Macmillan Co., New York (1957)
20. Weber, M.: From Max Weber: Essays in Sociology. In: Gerth, H.H. and Wright Mills, C. (ed.) Oxford University Press, New York (1958)

Chapter 7

Agent-based Modeling of Human Organizations [1]

Abstract. At present the agent paradigm is often used for computational modeling of human behavior in an organizational setting. However, in many developed models only a limited number of (unrelated) organizational aspects are represented. Furthermore, some of these models make little use of a rich theoretical basis developed in social science. This may undermine the practical feasibility of such models. This paper proposes a formal approach for modeling of characteristics and behavior of agents in organizations, diverse aspects of which are represented using an expressive formal framework. The approach is based on the theoretical findings from social science and enables analysis of how different organizational and environmental factors influence the behavior and performance of agents. The approach is illustrated by a simulation case.

1 Introduction

The agent paradigm has been extensively used for modeling and analysis of both human and artificial organizations. In particular, in the area of Multi-Agent Systems (MAS) the representation of a system as an organization consisting of roles and groups can help to handle high complexity and poor predictability of the system dynamics [11]. Although organizational models of MASs can be computationally effective, nevertheless most of them have a limited ontological expressivity required for modeling of human organizations. Furthermore, such models only rarely make use of an extensive theoretical basis developed in social science.

[1] A part of this chapter appeared as Sharpanskykh, A.: Modeling of Agents in Organizational Context. In: H-D. Burkhard, G. Lindeman, L. Varga, R. Verbrugge (eds.), Proceedings of the 5th International Central and Eastern European Conference on Multi-Agent Systems, LNAI 4696, Springer Verlag (2007)

Modeling of individuals in a social setting using the agent has gained popularity in the area of computational social science [3]. In contrast to the traditional methods (e.g., based on system dynamics [8]) that abstract from individual events and entities and take an aggregate view on the social dynamics, the agent-based approaches take into account the local perspective of a possibly large number of separate agents and their specific behaviors in (formal) organizational structures. Agent-based social simulation has been used for investigating organizational structures and dynamics at macro- (e.g., market fluctuations [1]), meso- (e.g., interactions between organizations [5]) and micro- (e.g., personal traits and organizational performance [25]) levels. In many approaches that identify and exploit relations between different levels much attention has been devoted to analyzing, predicting and improving the effectiveness and efficiency of the allocation and the execution of organizational tasks to/by different types of agents. In particular, the frameworks TAEMS [6] and VDT [14] provide the elaborated models for (collaborative) task environments and the computational means to analyze the performance of agents and of a whole organization with respect to the task execution. The agents in these and other similar frameworks are represented as autonomous entities with such characteristics as skills, competences, experience, and, sometimes, goals. In task-oriented agent-based modeling it is often assumed that agents comply with organizational goals and will perform tasks in such a way that a high level of organizational performance is ensured. However, in some cases such an assumption may not be valid. In particular, for feasible modeling of human organizations various (sometimes conflicting) interests of different organizational actors should be explicitly considered, as they often (significantly) influence the organizational performance. In general, to stimulate productive work of employees, an organization should reconcile (or align) its goals with the (key) goals of its employees. Furthermore, the organization should arrange work and provide incentives to its employees in such a way that they are constantly motivated to adopt the behavior that ensures the satisfaction of the essential organizational goals. The topic of work motivation has received much attention in Organization Theory [10, 13, 15, 18, 19]. Also, different computational motivation models and the mechanisms for manipulating them have been proposed [4]. However, only a little research has been done on the computational modeling of motivation and intentional attitudes of agents situated in the organizational context. Organizational factors that influence the behavior of agents are diverse: e.g., norms and regulations related to the tasks execution, to communication, a power (authority) system, a reward/punishment system etc. Furthermore, many of these factors are interrelated (e.g., a power structure influences the execution of tasks). However, often models that are used in social simulations consider only a limited number of organizational aspects and do not reveal (inter-) dependencies that exist between these aspects. This results into limited evaluation possibilities of effects of different organization processes and may undermine the practical feasibility of such models.

In this paper, a formal agent-based approach for modeling of characteristics and behavior of individuals in the organizational context is proposed. The approach makes use of a rich theoretical basis developed in Organization Theory. In particular, the motivation modeling of agents is based on the expectancy theory (the version of Vroom) [26] that has received good empirical support. The formal motivation modeling has an advantage that automated tools can be developed using which

(human resource (HR)) managers can make estimations of how different organizational factors influence the motivation and performance of different types of employees (agents). Agents are situated in a formal organization modeled using the general organization modeling and analysis framework proposed in [12]. This framework comprises several interrelated views: the performance-oriented view [21] describes organizational and individual goal and performance indicators structures; the process-oriented view [20] describes task and resource structures and dynamic flows of control; within the organization-oriented view [12, 24] organizational roles, their power and communication relations are defined. Concepts and relations within every view are formally described using dedicated languages based on an order sorted predicate logic [16]. Temporal relations within and across the views are formalized using the Temporal Trace Language (TTL) [23], which is an extension of an order sorted predicate logic that allows reasoning about dynamic properties of systems. Both the order sorted predicate logic and TTL are also used for specifying the structural and temporal aspects of agent-based models correspondingly.

The paper is organized as follows. Section 2 introduces the proposed modeling approach. The application of the approach is illustrated by a simulation case study in Section 3. Section 4 concludes the paper.

2 An Agent-based Modeling Approach

Using the general modeling framework an organizational model that comprises concepts and relations from different views is specified. The elements of the model are related as follows: Organizational goals are structured into a hierarchy using the refinement relations. Goals are satisfied by execution of certain tasks. Different sets of organizational tasks are associated with roles. Interaction (e.g., communication) and authority structures are defined on organizational roles with respect to tasks. To enable effective and efficient execution of tasks, agents with appropriate characteristics should be allocated to roles. In this Section, a description of professional, psychological, and intentional agent characteristics is provided (Section 2.1), followed by the introduction of a motivation model of an agent (Section 2.2).

2.1 Characteristics of agents and allocation to roles

For each role a set of requirements on agent *capabilities* (i.e., knowledge and skills) and *personal traits* is defined. Requirements related to knowledge define facts and procedures with respect to organizational tasks, confident understanding of which is required from an agent. Skills describe developed abilities of agents to use effectively and readily their knowledge for tasks performance. In the literature [18] four types of skills relevant in the organizational context are distinguished: technical (related to the specific content of a task), interpersonal (e.g., communication, cooperation), problem-solving/decision-making and managerial skills (e.g., budgeting, scheduling, hiring). More specific requirements may be defined on skills reflecting their level of development, experience, the context in which these skills were attained. To enable testing (or estimation) of skills and knowledge, every particular skill and knowledge

is associated with a performance indicator(s) (PI) (e.g., the skill 'typing' is associated with the PI "the number of characters per minute"). Notice that some indicators may be soft (not directly measurable) (such as the level of flexibility); the value of such indicators may be established by indirect evidences (e.g., from the agent's history and achievements). Moreover, a skill may be associated with a compound PI built as a weighed expression on simple PIs.

Personal traits may also influence the successfulness of the execution of tasks. The traits are divided into five broad categories discovered in psychology [13]: openness to experience, conscientiousness, extroversion, agreeableness, and neuroticism. In some cases agent personal traits may be evaluated through psychological tests and by consultations with agents' referees. Some agent's traits may mediate the attainment of agent's skills. For example, extroversion and agreeableness play an important role in building interpersonal skills.

Agent capabilities and traits can have different levels of importance. Whereas required for a role capabilities and traits are compulsory for taking the role, desired capabilities and traits considered as an advantage. In some cases an organization may tolerate the deficiency in (or insufficient level of development of) some skills if a feasible guarantee is provided that this gap will be filled during a certain time period.

Most of the approaches on personnel management used currently are based on the HR-models [19]. In contrast to the traditional scientific management models [17], the HR-based approaches pay a special attention to the needs, desires and goals of employees and to the alignment of the individual goals with the organizational ones. Therefore, during the evaluation of agents-candidates for a role, also the goals of the agents should be taken into consideration (to a possible extent) to identify similarities and conflicts with the organizational goals.

In modern social science behavior of individuals is considered as goal-driven. A goal is defined as an objective to be satisfied describing a desired state or development of the individual. It is recognized that high level goals of individuals are largely dependant on their needs. These needs are to a great extent determined by the individual behavioral and biological history (i.e., biological and social background). Currently the following division of needs is identified in social science: (1) *extrinsic needs* associated with biological comfort and material rewards; (2) *social interaction* needs that refer to the desire for social approval, affiliation and companionship; (3) *intrinsic needs* that concern the desires for self-development, self-actualization, mastery and challenge. Such a categorization has some similarities with the hierarchy of needs proposed by Maslow [10]. However, a number of empirical studies showed that the Maslow's key hypothesis that the high-order (intrinsic) needs cannot motivate behavior of an individual until the lower-order (extrinsic) needs are satisfied does not hold for all individuals. Empirical evidences confirmed that the importance (or the priority) of different types of needs (and the associated goals) often changes over time in different life phases of an individual. The characteristics of an agent can be formalized using the sorted first-order predicate logic as it will be shown in Section 3.

In general, the efficiency of allocation of an agent to a role is dependant on how well the agent's characteristics (i.e., capabilities and traits) and goals fit with the role specification and the requirements. However, modern organizations implement very diverse allocation principles (e.g., based on equality, seniority or stimulation of

novices) [10]. Such principles can be formalized as allocation policies comprising executable (temporal) rules. An example of such a policy is given in Section 3.

When an individual is allocated to a role, the identification of his/her specific lower level goals is performed in cooperation with a managerial representative of the organization. During this process, the high level goals, based on the agent's needs are refined into more specific goals aligned with organizational goals using AND- and OR- relations as it is shown in [21]. Many authors argue that the lower level goals should as detailed and specific as possible [9, 19]; furthermore, such goals should be attainable by agents. Often two types of such goals are distinguished: development (or learning) and performance goals. Development goals reflect wishes of agents to gain certain knowledge or some skills that are also useful for the organization. For example, the attainment of the skills required to perform task(s) interrelated with the task(s) already assigned to the agent may enable the allocation of the agent to a more global and essential (composite) task. Individuals vary in the abilities and desires to learn; therefore, this type of goals is particularly dependent on the individuals' traits and goals. Performance goals usually concern the effectiveness and efficiency of the execution of the tasks already allocated to the agent. Both development and performance goals may change over time.

Within the performance-oriented view of the modeling framework [21] the formal specification of a goal is based on a mathematical expression over a PI(s). The characteristics of a goal include, among others: *priority*; *horizon* – for which time point/interval should the goal be satisfied; *hardness* – hard (satisfaction can be established) or soft (satisfaction cannot be clearly established, instead degrees of *satisficing* are defined); *negotiability*. For example, the hard performance goal "it is required to maintain the time for the generation of a plan < 24 hours" is based on the PI "the time for the generation of a plan". Another example is the development goal "it is desired to achieve the state in which the framework JADE is mastered". In the latter example the goal is desirable, which points at its low priority.

The satisfaction of goals in the organizational context is associated directly or indirectly with the performance of tasks. In particular, goals associated with intrinsic needs are often satisfied by intrinsic rewards that are a natural consequence of the agent behavior related to the execution of a task. While externally provided rewards (e.g., salary, bonuses, group acceptance) serve to the satisfaction of goals related to extrinsic and social interaction needs. At any time point the (level of) satisfaction of a lower level goal may be established by the evaluation of the PI expression, on which the goal is based. Further, using the rules defined in [21] information about the satisfaction of lower-level goals is propagated to determine the satisfaction of high-level goals.

Many organizations have reward/sanction systems contingent on the satisfaction of goals. Furthermore, besides general organizational policies also particular individual policies (e.g., concerning promotions, bonuses etc.) can be defined. Such policies can be also formalized by sets of executable rules. Many studies showed that making explicit rules based on which rewards and sanctions are provided increases the motivation of an agent to perform certain actions (tasks) [18]. The motivation of agents to perform certain tasks is important to ensure the satisfaction of both individual and organizational goals related (directly or indirectly) to these tasks.

Therefore, the motivational aspect of the agent behavior should be explicitly represented in the models of organizational agents.

2.2 Modeling the motivation of an agent

The topic of motivation in work organizations has received much attention in social science. In [26] the motivation is defined as a *process governing choice made by persons among alternative forms of voluntary activity*. There exist many different theories of motivation [15, 18, 19]. In this paper we adopt the Vroom's version of the expectancy theory [26] that has received a good empirical support.

According to this theory, when an individual evaluates alternative possibilities to act, s/he explicitly or implicitly makes estimations for the following factors: *expectancy*, *instrumentality*, and *valence* (see Fig.1).

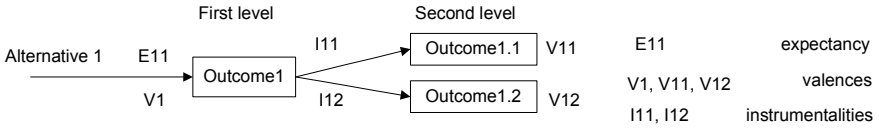


Fig. 1. An example of the motivation model by Vroom [26]

Expectancy refers to the individual's belief about the likelihood that a particular act will be followed by a particular outcome (called a first-level outcome). In the organizational context expectancy of an agent related to successful task execution is determined by the characteristics of the task and the agent, and by the organizational and environmental conditions. Tasks can be characterized along multiple dimensions: (a) complexity and predictability; (b) specialization; (c) formalization; (d) interrelation with other tasks; (e) collaboration required from agents. Usually agents that possess knowledge and the skills required for some task have a high level of expectancy of the successful task execution. Furthermore, agents with highly developed skills tend to assign a high value to expectancy associated with complex and not completely predictable tasks. On the opposite, inexperienced agents decrease their expectancy when dealing with complex tasks and especially with tasks with low predictability. For such agents the formalization of a task (e.g., by detailed procedure descriptions and guidelines) will increase their expectancy level. If a task requires from an agent a contribution from or collaboration with other agents, then the agent's belief about reliability and trustworthiness of these agents will play an important role in his/her expectancy estimation. Furthermore, other organizational factors, such as internal policies, rules and constraints (e.g., temporal, authority-related constraints) may influence expectancy of the task execution. Many modern organizations actively interact with the environment, which is often highly dynamic and unpredictable. The less certainty and knowledge about the environment an agent has (e.g., market fluctuations, resource availability), the less his/her expectancy level. As expectancy is defined as a subjective perception (or a belief) of an agent, the agent's personal traits also have influence on his/her expectancy.

Instrumentality is a belief concerning the likelihood of a first level outcome resulting into a particular second level outcome; its value varies between -1 and +1. A

second level outcome represents a desired (or avoided) by an agent state of affairs that is reflected in an agent's goal(s) (e.g., bonus receipt, imposition of a sanction). Although the notion of instrumentality can be perceived as probability, in contrast to the latter instrumentality may take negative values, in case a second-level outcome does not follow a particular first-level outcome. If an organizational reward system is defined explicitly, instrumentality between a performance level and the corresponding material reward/sanction is perceived as high (>0.5) by agents.

Note that the agent's experience gained by the execution of tasks influences the values of expectancies and instrumentalities associated with these tasks. For example, if despite high performance the agent did not get the promised/expected (amount of) rewards, then his/her instrumentality between the level of efforts and the previously identified reward will decrease. Similarly, the agent adjusts the expectancy value associated with a task based on his/her actual amount of efforts put into the task execution.

Valence refers to the strength of the individual's desire for an outcome or state of affairs. While second level outcomes are directly related to the agent's goals, the valence values associated with these outcomes refer to priorities of these goals. Thus, similarly to goal priorities, the values of valence change over time (e.g., depending on the satisfaction of goals).

While in most cases the correspondences between actions of agents and rewards provided externally can be specified in a straightforward way, the prerequisites for obtaining intrinsic rewards are less obvious. One of the conditions for intrinsic rewards identified in literature [9] is that a task assigned to an agent should represent a reasonably complete piece of work, to the outcomes of which the agent could attribute his/her efforts. Some agents receive intrinsic rewards from the very process of task execution irrespectively of the execution results. While intrinsic rewards for other agents are contingent upon the execution outcomes. In the latter case if the actual task result equates to or exceeds the agent's expectation, then the agent receives an intrinsic reward. Furthermore, as follows from [9] the amount of intrinsic reward is dependent on the task complexity.

In the Vroom model *the force on an individual to perform an act is a monotonically increasing function of the algebraic sum of the products of the valences of all outcomes and the strength of his expectancies that the act will be followed by the attainment of these outcomes* [26]. Hence, the motivational force to perform act i can be calculated as:

$$F_i = f\left(\sum_{j=1}^{n} E_{ij} \times V_j\right), \quad V_j = \sum_{k=1}^{m} V_{jk} \times I_{jk} \tag{1}$$

Here E_{ij} is the strength of the expectancy that act i will be followed by outcome j; V_j is valence of first-level outcome j; V_{jk} is valence of second-level outcome k that follows first-level outcome j; I_{jk} is perceived instrumentality of outcome j for the attainment of outcome k.

3 A Simulation Case Study

In this Section we shall investigate the behavior of the employees of a small firm that develops web graphics by request from external clients. Such an organization manages all its activities using a cohesive team structure. Teams have a flat power structure, which allows achieving high responsiveness to the environmental dynamics. Although the role of a leader (or manager) is identified, all important decisions are made with the assistance of all team members. The manager is responsible mostly for organizing tasks: e.g., searching for clients, distribution of orders, monitoring of the order execution. The firm consists of highly motivated members and has a very informal and open working style. The risky, environment-dependant nature of the firms of such type may cause financial insecurity and deficiency for their members. In the following the model used for the simulation is introduced. Subsequently, the simulation results are presented.

Modeling tasks and the environment

Tasks received by the firm are characterized by: (1) *name*; (2) *type*; (3) *required level(s) of development of skill(s)*; (4) *average / maximum duration*; (5) *extra time delay per unit of each skill development*; (6) *material reward*; (7) *intrinsic reward*; (8) *development level increment per skill*. For this case study the generalized PI "the development level" for each skill is used, which is an aggregated quantity (a real number in the range 0-5) reflecting the skill-related knowledge, experience, task execution context etc. The task average duration is the time that an agent that possesses the skills satisfying the task requirements will spend on the task execution. Agents with insufficient development levels of skills will require additional time for the execution. This is specified by the extra time delay characteristic per deficient unit of each required skill. The maximum task duration specifies the maximal time allowed for the task execution. For the successful performance of tasks agents are granted with material rewards; also the development level(s) of their skill(s) is (are) increased by the experience increment amount(s). Note that for simplicity the intrinsic rewards associated with the tasks in this case study are made independent of the specific characteristics of the agents who execute these tasks.

The task types used in the simulation are specified in Table 1. When detailed data about the task execution are available, more precise dependencies between task durations, extra delays and the skill development levels and traits can be established.

In the simulation we suppose that tasks arrive in accordance with a nonhomogeneous Poisson process $\{N(t),\ t \geq 0\}$ with a bounded intensity function $\lambda(t)$. Here $N(t)$ denotes the number of events that occur by time t and the quantity $\lambda(t)$ indicates how likely it is that an event will occur around the time t. We use the thining or random sampling approach [22], which assumes that $\lambda(t) \leq \lambda$ for all $t \leq T$, where T is the simulation time (2000 working hours (1 year) for this case study). Furthermore, for $T \leq 1000$: $\lambda_{A1}=\lambda_{A2}=\lambda_{B1}=\lambda_{B2}=0.05$ and for $T > 1000$: $\lambda_{A1}=\lambda_{A2}=2 * 10^{-5}$; $\lambda_{B1}=\lambda_{B2}=0.05$.

Organization modeling

The firm has two high level long-term goals with the same priority: "it is required to maintain a high profit level" and "it is required to maintain a high level of satisfaction of the employees". These goals are imposed on the organizational structure that comprises the role Manager and the generalized role Task Performer. The latter is instantiated into

specific roles-instances associated with the tasks received by the firm. An instantiated role is assigned to one of the agents representing the employees using the following policy: Agents that can be potentially allocated to a role should be *qualified* for this role. An agent is qualified for a role under two conditions: (1) the agent is not involved into the execution of any other tasks; (2) agent possesses the skills required for the task associated with the role; and the level of development of these skills will allow to the agent to finish the task before the task deadline (i.e., maximum duration).

Table 1. The characteristics of the task types A1/A2 (create a simple/complex web illustration) and B1/B2 (create a simple/complex Flash animation) used in the simulation

Type	**A1**	**A2**	**B1**	**B2**
Required skill(s)	S1: 2	S1: 4	S2: 1	S2: 4
Average (max) duration (hours)	14 (18)	30 (38)	12 (15)	50 (60)
Extra time delay per skill (hours)	S1: 2	S1:4	S2: 3	S2: 8
Material reward	10	20	7	25
Intrinsic reward	1	3	1	4
Development increment	S1:0.1	S1:0.2	S2: 0.08	S2: 0.2

To formalize these conditions, for each task and agent characteristic a predicate is introduced. Some of these predicates are given in Table 2. To express the temporal aspects of the agent qualification rule the language TTL is used [23]. TTL specifies the dynamics of a system by a trace, i.e. a temporally ordered sequence of states. Each state corresponds to a particular time point and is characterized by a set of state properties that hold in this state. State properties are defined as formulae in a sorted predicate logic using state ontologies. A state ontology defines a set of sorts or types (e.g., TASK, AGENT), sorted constants, functions and predicates (see Table 2). States are related to state properties via the satisfaction relation |=: state(γ, t) |= p, which denotes that state property p holds in trace γ at time t. Dynamic properties are specified in TTL by relations between state properties.

The agent qualification rule is formally expressed in TTL as follows:

∀γ ∀t:TIME ∀a1:TASK ∀ag:AGENT ∀r1:ROLE ∀tp1: TASK_TYPE
state(γ, t)|= [task_arrived(a1) ∧ role_for_task(r1, a1) ∧ task_type(a1, tp1) ∧
¬∃r2:ROLE r2≠r1 ∧ agent_allocated(ag, r2) ∧ sum([sk:SKILL], ∃VALUE:n, m, k case(state(γ, t)|=
task_requires_skill(a1, sk, n) ∧ agent_possesses_skill(ag, sk, m) ∧ m≥0.5∧ task_extra_delay(tp1,
sk, k), k • (n-m), 0)) < (task_max_duration(tp1) - task_average_duration(tp1))
⇒ ∀t1:TIME t1>t state(γ, t1)|= agent_qualified_for(ag, r1)

Here in sum([summation_variables], case(logical_formula, value1, 0)) logical_formula is evaluated for every combination of values from the domains of each from the summation_variables; and for every evaluation when logical_formula is true, value1 is added to the resulting value of the sum function.

Table 2. Predicates for the formalization of agent-based models

Predicate	Description
task_arrived, task_started, task_finished: TASK	Specifies the arrival, start and finish of a task
role_for_task: ROLE x TASK	Identifies a role for a task
agent_allocated: AGENT x ROLE	Specifies an agent allocated to a role
agent_qualified_for: AGENT x ROLE	Specifies an agent qualified for a role
agent_requested: AGENT x ROLE	Identifies an agent that requested a role

Further, since the firm recognizes the importance of wishes of its employees, a role can be only allocated, when a qualified agent has voluntarily requested the role. Furthermore, the firm established the rule that in case several qualified agents requested a role, then the agent with the most distant (i.e., the earliest) previous allocation time among these agents will be allocated to the role. This rule is also formalized using TTL:

$\forall\gamma\ \forall$t, t1:TIME $\forall$ag:AGENT $\forall$r1:ROLE $\forall$a1:TASK
state(γ, t) |= [agent_qualified_for(ag, r1) $\wedge$ agent_requested(ag, r1) $\wedge$ role_for_task(r1, a1) $\wedge$ latest_allocation(ag, t1) $\wedge$ $\forall$ag1:AGENT $\forall$t2:TIME ag1$\neq$ag $\wedge$ agent_requested(ag1, r1) $\wedge$ latest_allocation(ag1, t2) $\wedge$ t1 < t2
$\Rightarrow$ agent_allocated(ag, r1) $\wedge$ task_started(a1)]

Here latest_allocation(ag1, t1) is a short notation for:
$\exists$t1:TIME $\exists$a2: TASK $\exists$r2: ROLE state(γ, t1)|= task_finished(a2) $\wedge$ role_for_task(r2, a2) $\wedge$ agent_allocated(ag1, r2) $\wedge$ $\forall$t2:TIME t2> t1 $\forall$r3:ROLE state(γ, t2)|= $\neg$agent_allocated(ag1, r3)

For the successful execution of tasks the agents are provided with material rewards on the following basis: 50% of the reward is given to the agent who performed the task and the rest is divided equally among all other employees.

Modeling agents

The firm consists of three members and the manager modeled as agents. As in the most firms of such type, the employees are intrinsically motivated by their work, and pursuit high performance standards. For each agent two high level long-term hard goals are defined that also comply with the organizational goals: g1: it is required to maintain the level of income not less than 50; g2: it is required to maintain the level of intrinsic satisfaction not less than 5. It is assumed that the goal g1 when unsatisfied has higher priority than the goal g2. When g1 is satisfied, g2 becomes more important.

Two agents ag1 and ag2 possess the skill S1 to perform purely graphical work: agent_possesses_skill(ag1, S1, 4) and agent_possesses_skill(ag2, S1, 3). Here the third argument denotes the level of the skill development. The agent ag3 has the skill S2 to make Flash animations: agent_possesses_skill(ag3, S2, 4). Furthermore, ag1 has the general knowledge related to S2 (agent_possesses_skill(ag1, S2, 0.1)), which however is insufficient for the performance of tasks that require S2. By mutual consent of the firm and ag1 the development goal for ag1 without a strict deadline has been set: it is desired to achieve the level of development of S2 $\geq$ 0.5. When ag1 decides to gain the minimum level of the skill S2 development that is necessary for the task execution (0.5), s/he will be given one week for the training, during which no other tasks will be assigned to him/her. The motivation of the agents to attain their goals is represented by the motivation models, two examples of which for ag1 are given in Fig. 2.

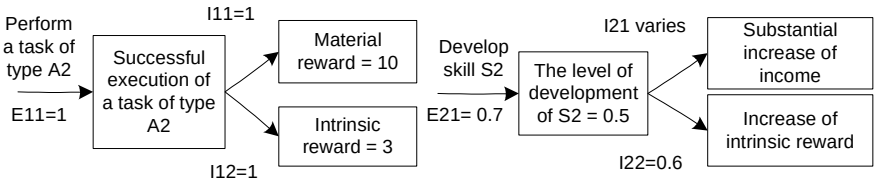
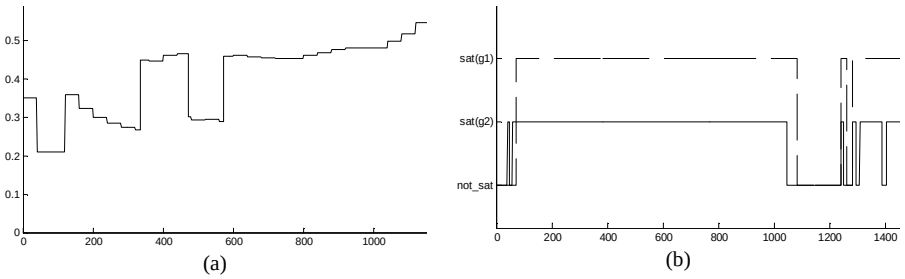


Fig. 2. The examples of two motivation models for the agent ag1 used in the case study

The parameters of the motivation models are defined as follows: Expectancy of an agent ag for the successful execution of a task tk is defined as a weighed average of the quotients $pos(sk_i)/req(sk_i)$ for each skill sk_i required for tk; here $pos(sk_i)$ is the development level of the skill sk_i possessed by ag and $req(sk_i)$ is the level required by tk. Instrumentality for each second level outcome associated with the successful execution of a task is equal to 1 for every agent qualified for this task. This is because the reward system is defined explicitly and the qualified agents have a clear estimation of the intrinsic reward associated with the task. The instrumentality value of ag1 for the skill S2 development is reevaluated in the end of each month and is equal to 1, when $n/m > 50$, and is equal to $n/(m*50)$ otherwise; here n is the amount of the material rewards provided by the tasks of types B1 and B2 received by the firm up to the current time point, and m is the amount of months of the simulation time (the initial instrumentality value is 0.35). The valence values of second level outcomes change over time. In particular, when the goal g1 of an agent ag is not satisfied, then the valence values of ag for all outcomes related to material rewards will become 1, and the valence values of outcomes related to intrinsic rewards will become 0.5. When g1 is satisfied, then the valence values for material outcomes will decrease to 0.5, and for intrinsic outcomes will increase to 1. An agent generates a request to perform an action specified in a motivation model (e.g., request for a role), when the motivational force associated with this action calculated using the formula (1) is greater than 0.5.
The initial income value is 20 for all agents, and the initial intrinsic satisfaction level is 3. Each agent consumes 0.05 units of the received material rewards per day and the amount of the received intrinsic rewards decreases by 0.03 each day.



(a) (b)

Fig. 3. (a) The change of the motivation force (the vertical axis) of agent ag1 for the attainment of skill S2 over time. (b) The change of the satisfaction of the goals of agent ag1 over time.

The simulation is performed using the dedicated tool [2]. Fig. 3a shows how the motivational force of ag1 to attain the skill S2 changes over time. After the time point 1000, when the amount of tasks of type A diminishes significantly, the force transgresses the threshold 0.5, and ag1 begins the attainment of S2. After some time

ag1 possesses the skills required to perform the tasks of both types A and B and both his/her goals g1 and g2 become satisfied (see Fig. 3b).

4 Conclusion

The paper proposes a formal approach for modeling of agents situated in an (formal) organization that accentuates the intentional and motivational aspects of agent behavior. The proposed quantitative motivation model of an agent based on the expectancy theory allows estimating the agent's motivational force to attain certain (organizational or individual) goals. Since the goal expressions are based on performance measurements, using the proposed approach it is possible to analyze how different organizational factors that affect the parameters of the motivation model influence the organizational or agent performance. An example of such analysis is demonstrated by a simulation case study in this paper.

Based on a large corpus of empirical social studies a great number of dependencies between organizational and environmental factors and the agent's motivation have been identified. In general, to create a feasible and valid model for a complex organization, a large number of variables and functions representing these factors and dependencies should be specified. This causes such undesirable properties of a model as a high complexity and the loss of tractability [7]. Therefore, it is recommended that an organization analyst depending on the organizational type and the purpose of analysis should choose only the most relevant organizational and environmental factors that have a direct impact on the agent behavior in the considered organizational setting. Such a choice may be based on the results of empirical studies for organizations of the considered type.

In the future research the behavior of various types of agents situated in organizations of different types (e.g., mechanistic, organic [17]) will be investigated.

References

1. Bertels, K., Boman, M.: Agent-Based Social Simulation in Markets. Electronic Commerce, 1(1-2) (2001) 149 – 158
2. Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J.: A Language and Environment for Analysis of Dynamics by Simulation. International Journal of Artificial Intelligence Tools, 16: 435-464 (2007)
3. Carley, K.M.: A comparison of artificial and human organizations. Journal of Economic Behavior & Organization, 31(2) (1996) 175-191
4. Coddington, M. and Luck, M.: A Motivation Based Planning and Execution Framework, International Journal on Artificial Intelligence Tools, 13(1) (2004) 5-25
5. Davidsson, P., Henesey, L., Ramstedt, L., Tornquist, J., Wernstedt, F.: An Analysis of Agent-Based Approaches to Transport Logistics. Transportation Research Part C: Emerging Technologies, Vol. 13(4) (2005) 255-271
6. Decker, K.: TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. Foundations of Distributed Artificial Intelligence, Chapter 16, O'Hare, G. and Jennings, N. (eds.), Wiley Inter-Science (1996) 429-448

7. Dooley, K.: Simulation research methods. In: Baum, J. (ed.): Companion to Organizations. Blackwell, London (2002) 829-848
8. Forrester, J. W.: Industrial dynamics, Waltham, MA: Pegasus Communications (1961)
9. Galbraith, J.R.: Designing organizations. Jossey-Bass, San Francisco California (1995)
10. Hackman, J.R.: Work redesign and motivation. Professional Psychology, 11 (1980) 445-455.
11. Horling, B. and Lesser, V. A Survey of multi-agent organizational paradigms. The Knowledge Engineering Review, Vol. 19(4) (2005) 281-316
12. Jonker C.M., Sharpanskykh, A., Treur, J., Yolum, P.: A Framework for Formal Modeling and Analysis of Organizations, Applied Intelligence, 27(1), 49-66 (2007)
13. Katz, D and Kahn, R.: The social psychology of organizations. Wiley, New York (1966)
14. Kunz, J.C., Levitt, R.E., and Jin Y. The Virtual Design Team: A computational simulation model of project organizations. Communications of the Association for Computing Machinery 41(11) (1999) 84-92.
15. Lawler, E.E.: Motivation in Work Organisations, Cole Publishing (1973)
16. Manzano, M.: Extensions of First Order Logic. Cambridge University Press (1996)
17. March, J.G. and Simon, H.A.: Organizations. Wiley, New York (1958)
18. Pinder, C. C. (1998). Work motivation in organizational behavior. Upper Saddle River, NJ: Prentice-Hall.
19. Porter, L.W., Bigley, G.A., Steers, R.M. (eds.): Motivation and Work Behavior, 7th edition. New York: McGraw-Hill (2003)
20. Popova, V., Sharpanskykh, A.: Formal analysis of executions of organizational scenarios based on process-oriented models. In I. Zelinka, Z. Oplatkova and A. Orsoni (eds.), Proceedings of 21st European Conference on Modelling and Simulation ECMS'07, SCS Press (2007) 36-44.
21. Popova, V., Sharpanskykh, A.: Formal Modeling of Goals in Agent Organizations. In: V. Dignum, F. Dignum, E. Matson, B.Edmonds (eds.), Proceedings of AOMS workshop at IJCAI'07 (2007) 74-80
22. Ross, S.: Simulation. 2nd edn. Harcourt Academic Press, London Boston New York (1998)
23. Sharpanskykh, A., Treur, J.: Verifying Interlevel Relations within Multi-Agent Systems. In: Brewka, G., Coradeschi, S., Perini, A., and Traverso, P. (eds.), Proceedings of the 17th European Conference on Artificial Intelligence, Riva del Garda, September, IOS Press (2006) 290-294
24. Sharpanskykh, A.: Authority and its Implementation in Enterprise Information Systems. In: Sadiq, S., Reichert, M., Schulz, K., Trienekens, J., Moller, C., and Kusters, J. (eds.), Proceeding of the 1st International Workshop on Management of Enterprise Information Systems, MEIS 2007, INSTICC Press (2007) 33-43
25. Tyler G.P., Newcombe P.A.: Relationship Between Work Performance and Personality Traits in Hong Kong Organizational Settings. International Journal of Selection and Assessment 14 (1) (2006) 37–50
26. Vroom, V.H.: Work and motivation. Wiley, New York (1964)

Chapter 8

On the Complexity Monotonicity Thesis for Environment, Behaviour and Cognition[1]

Abstract. Development of more complex cognitive systems during evolution is sometimes viewed in relation to environmental complexity. In more detail, growth of complexity during evolution can be considered for the dynamics of externally observable behaviour of agents, for their internal cognitive systems, and for the environment. This paper explores temporal complexity for these three aspects, and their mutual dependencies. A number of example scenarios have been formalised in a declarative temporal language, and the complexity of the structure of the different formalisations was measured. Thus, some empirical evidence was provided for the thesis that for more complex environments, more complex behaviour and more complex mental capabilities are needed.

1 Introduction

Behaviour of agents (both living organisms and artificial (software or hardware) agents) can occur in different types and complexities, varying from very simple behaviour to more sophisticated forms. Depending on the complexity of the externally observable behaviour, the internal mental representations and capabilities required to generate the behaviour also show a large variety in complexity. From an evolutionary viewpoint, for example, Wilson [16], p. 187 and Darwin [3], p. 163 point out how the

[1] This chapter appeared as Bosse, T., Sharpanskykh, A., Treur, J.: On the Complexity Monotonicity Thesis for Environment, Behaviour and Cognition. In: Baldoni, M., Son, T.C., Riemsdijk, M.B. van, and Winikoff, M. (eds.): Proceedings of the 5th International Workshop on Declarative Agent Languages and Technologies, DALT 2007, 17-32 (2007) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author). The post-proceedings of the workshop will be published by Springer Verlag.

development of behaviour relates to the development of more complex cognitive capabilities. Godfrey-Smith [4], p. 3 assumes a relationship between the complexity of the environment and the development of mental representations and capabilities. He formulates the main theme of his book in condensed form as follows: 'The function of cognition is to enable the agent to deal with environmental complexity' (the *Environmental Complexity Thesis*). In this paper, this thesis is refined as follows:

- the more complex the environment, the more sophisticated is the behaviour required to deal with this environment,
- the more sophisticated the behaviour, the more complex are the mental representations and capabilities needed

This refined thesis will be called the *Complexity Monotonicity Thesis*. The idea is that to deal with the physical environment, the evolution process has generated and still generates a variety of organisms that show new forms of behaviour. These new forms of behaviour are the result of new architectures of organisms, including cognitive systems with mental representations and capabilities of various degrees of complexity. The occurrence of such more complex architectures for organisms and the induced more complex behaviour itself increases the complexity of the environment during the evolution process. New organisms that have to deal with the behaviour of such already occurring organisms live in a more complex environment, and therefore need more complex behaviour to deal with this environment, (to be) realised by an architecture with again more complex mental capabilities. In particular, more complex environments often ask for taking into account more complex histories, which requires more complex internal cognitive representations and dynamics, by which more complex behaviour is generated.

This perspective generates a number of questions. First, how can the Complexity Monotonicity Thesis be formalised, and in particular how can the 'more complex' relation be formalised for (1) the environment, (2) externally observable agent behaviour and (3) internal cognitive dynamics? Second, connecting the three items, how to formalise (a) when does a behaviour fit an environment: which types of externally observable behaviours are sufficient to cope with which types of environments, and (b) when does a cognitive system generate a certain behaviour: which types of internal cognitive dynamics are sufficient to generate which types of externally observable agent behaviour?

In this paper these questions are addressed from a dynamics perspective, and formalised by a declarative temporal logical approach. Four cases of an environment, suitable behaviour and realising cognitive system are described, with an increasing complexity over the cases. Next, for each case, complexity of the dynamics of environment, externally observable agent behaviour and internal cognitive system are formalised in terms of structure of the formalised temporal specifications describing them, thus answering (1) to (3). Moreover, (a) and (b) are addressed by establishing formalised logical (entailment) relations between the respective temporal specifications. By comparing the four cases with respect to complexity, the Complexity Monotonicity Thesis is tested.

2 Evolutionary Perspective

The environment imposes certain requirements that an agent's behaviour needs to satisfy; these requirements change due to changing environmental circumstances. The general pattern is as follows. Suppose a certain goal G for an agent (e.g., sufficient food uptake over time) is reached under certain environmental conditions ES1 (Environmental Specification 1), due to its Behavioural Specification BS1, realised by its internal (architecture) CS1 (Cognitive Specification 1). In other words, the behavioural properties BS1 are sufficient to guarantee G under environmental conditions ES1, formally ES1 & BS1 $\Rightarrow$ G, and the internal dynamics CS1 are sufficient to guarantee BS1, formally CS1 $\Rightarrow$ BS1. In other environmental circumstances, described by environmental specification ES2 (for example, more complex) the old circumstances ES1 may no longer hold, so that the goal G may no longer be reached by behavioural properties BS1. An environmental change from ES1 to ES2 may entail that behaviour BS1 becomes insufficient. It has to be replaced by new behavioural properties BS2 (also more complex) which express how under environment ES2 goal G can be achieved, i.e., ES2 & BS2 $\Rightarrow$ G.

Thus, a population is challenged to realise such behaviour BS2 by changing its internal architecture and its dynamics, and as a consequence fulfill goal G again. This challenge expresses a redesign problem: the given architecture of the agent as described by CS1 (which entails the old behavioural specification BS1) is insufficient to entail the new behavioural requirements BS2 imposed by the new environmental circumstances ES2; the evolution process has to redesign the architecture into one with internal dynamics described by some CS2 (also more complex), with CS2 $\Rightarrow$ BS2, to realise the new requirements on behaviour.

Based on these ideas, the Complexity Monotonicity Thesis can be formalised in the following manner. Suppose $< E_1, B_1, C_1 >$ and $< E_2, B_2, C_2 >$ are triples of environment, behaviour and cognitive system, respectively, such that the behaviours B_i are adequate for the respective environment E_i and realised by the cognitive system C_i. Then the Complexity Monotonicity Thesis states that

$$E_1 \leq_c E_2 \Rightarrow B_1 \leq_c B_2 \quad \& \quad B_1 \leq_c B_2 \Rightarrow C_1 \leq_c C_2$$

Here $\leq_c$ is a partial ordering in complexity, where $X \leq_c Y$ indicates that Y is more complex than X. A special case is when the complexity ordering is assumed to be a total ordering where for every two elements X, Y either $X \leq_c Y$ or $Y \leq_c X$ (i.e., they are comparable), and when some complexity measure cm is available, assigning degrees of complexity to environments, behaviours and cognitive systems, such that

$$X \leq_c Y \Leftrightarrow cm(X) \leq cm(Y)$$

where $\leq$ is the standard ordering relation on (real or natural) numbers. In this case the Complexity Monotonicity Thesis can be reformulated as

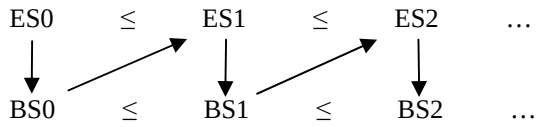
$$cm(E_1) \leq cm(E_2) \Rightarrow cm(B_1) \leq cm(B_2) \quad \&$$
$$cm(B_1) \leq cm(B_2) \Rightarrow cm(C_1) \leq cm(C_2)$$

The Temporal Complexity Monotonicity Thesis can be used to explain increase of complexity during evolution in the following manner. Make the following assumption on Addition of Environmental Complexity by Adaptation, as described above:

- adaptation of a species to an environment adds complexity to this environment

Suppose an initial environment is described by ES0, and the adapted species by BS0. Then this transforms ES0 into a more complex environmental description ES1.

Based on ES1, the adapted species will have description BS1. As ES1 is more complex than ES0, by the Complexity Monotonicity Thesis it follows that this BS1 is more complex than BS0: ES0 ≤ ES1 ⇒ BS0 ≤ BS1. Therefore BS1 again adds complexity to the environment, leading to ES2, which is more complex than ES1, et cetera[1]:



This argument shows that the increase of complexity during evolution can be related to and explained by two assumptions: the Complexity Monotonicity Thesis, and the Addition of Environmental Complexity by Adaptation assumption. This paper focuses on the former assumption.

3 Variations in Behaviour and Environment

To evaluate the approach put forward, a number of cases of increasing complexity are analysed, starting from very simple *stimulus-response behaviour* solely depending on stimuli the agent gets as input at a given point in time. This can be described by a very simple temporal structure: direct associations between the input state at one time point and the (behavioural) output state at a next time point. A next class of behaviours, with slightly higher complexity, analysed is *delayed response behaviour*: behaviour that not only depends on the current stimuli, but also may depend on input of the agent in the past. This pattern of behaviour cannot be described by direct functional associations between one input state and one output state; it increases temporal complexity compared to stimulus-response behaviour. For this case, the description relating input states and output states necessarily needs a reference to inputs received in the past. Viewed from an internal perspective, to describe mental capabilities generating such a behaviour, often it is assumed that it involves a memory in the form of an internal model of the world state. Elements of this world state model mediate between the agent's input and output states.

Other types of behaviour go beyond the types of reactive behaviour sketched above. For example, behaviour that depends in a more indirect manner on the agent's input in the present or in the past. Observed from the outside, this behaviour seems to come from within the agent itself, since no direct relation to current inputs is recognised. It may suggest that the agent is motivated by itself or acts in a goal-directed manner. For a study in *goal-directed behaviour* and foraging, see, for example, [5]. Goal-directed behaviour to search for invisible food is a next case of behaviour analysed. In this case the temporal description of the externally observable behavioural dynamics may become still more complex, as it has to take into account more complex temporal relations to (more) events in the past, such as the positions already visited during a search process. Also the internal dynamics may become more

[1] Note that this argument can also be applied to multiple species at the same time, i.e., species A increases the complexity of the environment, which causes another species B to adapt to this more complex environment.

complex. To describe mental capabilities generating such a type of behaviour from an internal perspective, a mental state property *goal* can be used. A goal may depend on a history of inputs. Finally, a fourth class of behaviour analysed, which also goes beyond reactive behaviour, is *learning behaviour* (e.g., conditioning). In this case, depending on its history comprising a (possibly large) number of events, the agent's externally observable behaviour is tuned. As this history of events may relate to several time points during the learning process, this again adds temporal complexity to the specifications of the behaviour and of the internal dynamics.

To analyse these four different types of behaviour in more detail, four cases of a food supplying environment are considered in which suitable food gathering behaviours are needed. These cases are chosen in such a way that they correspond to the types of behaviour mentioned above. For example, in case 1 it is expected that stimulus-response behaviour is sufficient to cope with the environment, whilst in case 2, 3 and 4, respectively, delayed response behaviour, goal-directed behaviour, and learning behaviour is needed). The basic setup is inspired by experimental literature in animal behaviour such as [6], [14], [15]. The world consists of a number of positions which have distances to each other. The agent can walk over these positions. Time is partitioned in fixed periods (days) of a duration of d time units (hours). Every day the environment generates food at certain positions, but this food may or may not be visible, accessible and persistent at given points in time. The four different types of environment with increasing temporal complexity considered are:

(1) Food is always visible and accessible. It persists until it is taken.
(2) Food is visible at least at one point in time and accessible at least at one later time point. It persists until it is taken.
(3) Food either is visible at least at one point in time and accessible at least at one later time point, or it is invisible and accessible the whole day. It persists until it is taken.
(4) One of the following cases holds:
    a) Food is visible at least at one point in time and accessible at least at one later time point. It persists until it is taken.
    b) Food is invisible and accessible the whole day. It persists until it is taken.
    c) Food pieces can disappear, and new pieces can appear, possibly at different positions. For every position where food appears, there are at least three different pieces in one day. Each piece that is present is visible. Each position is accessible at least after the second food piece disappeared.

Note that there is an accumulating effect in the increase of complexity of these types of environment. For example, the behaviour of environment (3) is described as the disjunction of the behaviour of environment (2) and another type of behaviour. For this reason, it is expected that agents that survive in environment n will also survive in environment n-1.

4 Modelling Approach

To express formal specifications for environmental, behavioural and cognitive dynamics for agents, the Temporal Trace Language (TTL, see [2]) is used. This language is a variant of order-sorted predicate logic. In dynamic property expressions, TTL allows explicit references to time points and traces. If a is a state property, then, for example state(γ, t, input(agent)) |= a denotes that this state property holds in trace γ at time point t in the input state of the agent. Here, a *trace* (or trajectory) is defined as a time-indexed sequence of states, where time points can be expressed, for example, by

real or integer values. If these states are input states, such a trace is called an *input trace*. Similarly for an *output trace*. Moreover, an *input-output correlation* is defined as a binary relation C : Input_traces x Output_traces between the set of possible input traces and the set of possible output traces.

In the following sections, the four variations in behaviour and environment as introduced above are investigated in more detail. For formalising dynamic properties in TTL that will be used to specify these cases, the following state properties are used:

at(o, p)	object o is at position p
visible(sp)	an object occurring in the state property sp is visible (e.g. as it is not covered by a large object)
accessible(p)	position p is accessible (e.g. because there is no enemy at the position)
distance(p1, p2, i)	the distance between positions p1 and p2 is i
max_dist	a constant indicating the maximum distance the agent can travel in one step
observed(sp)	the agent observes state property sp
performing_action(a)	the agent performs action a

For example, a property that describes stimulus-response behaviour of an agent that goes to food, observed in the past can be expressed and formalised as follows:

At any point in time t,
if the agent observes itself at position p
and it observes an amount of food x at position p'
and position p' is accessible
then at the next time point after t the agent will go to position p'

Formalisation:

$\forall t\ \forall x\ \forall p\ \forall p'$
$[\ state(\gamma, t, input(agent)) \models observed(at(agent, p)) \wedge observed(at(food(x), p')) \wedge$
$observed(accessible(p'))\ \Rightarrow\ state(\gamma, t+1, output(agent)) \models performing\_action(goto(p'))\ ]$

5 Behavioural Cases

Using the introduced approach to formalise dynamic properties, the four variations in behaviour and environment are addressed in this section: stimulus-response, delayed-response, goal-directed, and learning behaviour.

5.1 Stimulus-Response Behaviour

As a first, most simple type of behaviour, stimulus-response behaviour is analysed in more detail. For this and the following cases of behaviour the following basis properties EP1-EP5 are used to describe the behaviour of the environment. They are specified both in a structured semi-formal temporal language, and in the formal temporal language TTL. Additionally, for every case specific properties of the environment will be specified.

Environmental properties

EP1 Sufficient food within reach

At the beginning of every day n (d is the duration of a day), the agent is positioned at a position p, and a sufficient amount x of food (c is the minimum) is provided at some position p' within reachable distance from p.

∀n ∃p ∃p' ∃x ∃i x>c & i≤max_dist &
state(γ, n*d, environment) |= at(agent, p) ∧ at(food(x), p') ∧ distance(p, p', i)

EP2 Complete observability
If the agent is at position p, and a(p, p') is a visible state property involving p and a position p' within reachable distance, then this is observed by the agent. This property is to be applied to food, distance, accessibility, agent position, and the absence of these.
∀t ∀x ∀p ∀p' ∀i
[[i≤max_dist & state(γ, t, environment) |= at(agent, p) ∧ a(p, p') ∧ visible(a(p, p')) ∧
distance(p, p', i)] ⇒ state(γ, t, input(agent)) |= observed(a(p, p'))]

EP3 Guaranteed effect of movement
At any point in time t, if the agent goes to position p, then it will be at position p.
∀t ∀p state(γ, t, output(agent)) |= performing_action(goto(p))
⇒ state(γ, t+1, environment) |= at(agent, p)

EP4 Guaranteed effect of eating
At any point in time t, if the agent takes food and the amount of food is sufficient for the agent then the agent will be well fed
∀t [[∀x state(γ, t, output(agent))|= performing_action(take(food(x))) & x≥c]
⇒ state(γ, t+1, environment) |= agent_well_fed]

EP5 Reachability of environment
The distances between all positions p in the agent's territory are smaller than max_dist. Here, p and p' are variables over the type TERRITORY_POSITION, which is a subtype of POSITION.
∀t ∀p ∀p' ∀I state(γ, t, environment) |= distance(p, p', i) ⇒ i ≤ max_dist

The following environmental properties hold for the stimulus-response case and some of the other cases considered.

EP6 Food persistence
Food persists until taken by the agent.
∀t1 ∀t2 ∀x ∀p [t1<t2 & state(γ, t1, environment) |= at(food(x), p) &
[∀t t1 ≤ t ≤ t2 ⇒ state(γ, t, output(agent)) |= not(performing_action(take(food(x))))]
⇒ state(γ, t2, environment) |= at(food(x), p)]

EP7 Food on one position
Per day, food only appears on one position.
∀n ∀x ∀p ∀p' ∀t state(γ, n*d, environment) |= at(food(x), p) &
state(γ, t, environment) |= at(food(x), p') & n*d < t ≤ (n+1)*d ⇒ p = p'

EP8 Complete accessibility
Each position is accessible for the agent (i.e., never blocked by enemies).
∀t ∀p state(γ, t, environment) |= accessible(p)

EP9 Complete visibility
All state properties a(p, p') that are true, are visible (which means that they will be observed by agents that are close enough, according to EP2). This property is to be applied to food, distance, accessibility, agent position, and the absence of these.
∀t ∀p ∀p' state(γ, t, environment) |= a(p, p') ⇒ state(γ, t, environment(agent)) |= visible(a(p, p'))

Note that the property of an agent being well fed is assumed to be a state property of the environment, since it refers to the agent's body state.

For the case of stimulus-response behaviour the environment is characterised by the following conjunction ES1 of a subset of the environmental properties given above:

ES1 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP8 & EP9

Behavioural Properties

The agent's stimulus-response behaviour is characterised by the following behavioural properties.

BP1 Going to observed food
At any point in time t, if the agent observes itself at position p and it observes no food at position p and it observes that an amount of food x is present at position p' and it observes that position p' is accessible and it observes that position p' is within reachable distance then it will go to position p'.
$\forall t\ \forall x\ \forall p\ \forall p'$ [[state(γ, t, input(agent)) |= observed(at(agent, p)) $\wedge$ observed(not(at(food(x), p))) $\wedge$ observed(at(food(x), p')) $\wedge$ observed(accessible(p')) $\wedge$ observed(distance(p, p', i)) & i$\leq$max_dist]
$\Rightarrow$ state(γ, t+1, output(agent)) |= performing_action(goto(p'))]

BP2 Food uptake
At any point in time t, if the agent observes itself at position p and the agent observes food at p then it will take the food
$\forall t\ \forall x\ \forall p$ [[state(γ, t, input(agent)) |= observed(at(agent, p)) $\wedge$ observed(at(food(x), p))]
$\Rightarrow$ state(γ, t+1, output(agent))|= performing_action(take(food(x)))]

Vitality property VP
The animal gets sufficient food within any given day.
$\forall n\ \exists t1$ [n*d $\leq$ t1 $\leq$ (n+1)*d & state(γ, t1, environment) |= agent_well_fed]

Logical relations
Given the dynamic properties specified above, the *environmental* and *behavioural* specifications (in short, ES1 and BS1) for case 1 (stimulus-response behaviour) are as follows:

ES1 $\equiv$ EP1 & EP2 & EP3 & EP4 & EP5 &EP6 & EP7 & EP8 & EP9
BS1 $\equiv$ BP1 & BP2

Given these specifications, the question is whether they are logically related in the sense that this behaviour is adequate for this environment, i.e., whether indeed the following implication holds:

BS1 & ES1 $\Rightarrow$ VP

To automatically check such implications between dynamic properties at different levels, model checking techniques can be used. To this end, first the dynamic properties should be converted from TTL format to a finite state transition format. This can be done using an automated procedure, as described in [11]. After that, for checking the implications between the converted properties, the model checker SMV is appropriate (see URL: http://www.cs.cmu.edu/~modelcheck/smv.html; see also [8]). SMV has been used to verify (and confirm) the above implication, as well as a number of other implications shown in this paper.

Concerning the relation between the specification of the *cognitive* and the *behavioural* dynamics: in this case CS1 = BS1. Thus, CS1 $\Rightarrow$ BS1 also holds.

5.2 Delayed Response Behaviour

In delayed response behaviour, previous observations may have led to maintenance of some form of memory of the world state: a model or representation of the (current) world state (for short, *world state model*). This form of memory can be used at any point in time as an additional source (in addition to the direct observations). In that case, at a given time point the same input of stimuli can lead to different behavioural

output, since the world state models based on observations in the past can be different. This makes that agent behaviours do not fit in the setting of an input-output correlation based on a direct functional association between (current) input states and output states. Viewed from an external viewpoint, this type of behaviour, which just like stimulus-response behaviour occurs quite often in nature, is just a bit more complex than stimulus-response behaviour, in the sense that it adds complexity to the temporal dimension by referring not only to current observations but also to observations that took place in the past.

This leads to the question what kind of complexity in the environment is coped with this kind of behaviour that is not coped with by stimulus-response behaviour. An answer on this question can be found in a type of environment with aspects which are important for the animal (e.g., food or predators), and which cannot be completely observed all the time; e.g., food or predators are sometimes hidden by other objects:

Environmental properties

For this case the environment described sometimes shows the food, but not always as in the previous case. It is characterised by the following conjunction ES2 of a subset of the environmental properties given above, extended with the properties EP10, EP11 and EP12 given below:

ES2 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP10 & EP11 & EP12

EP10 Temporary visibility of food
Per day, all food that is present is visible for at least one time point, and is accessible for at least one later time point[1].

EP11 Complete visibility of non-food
All state properties that are true, except the presence of food, are visible. Thus, this property is applied to distance, accessibility, and agent position.

EP12 Complete local observability of food
For all time points, if the agent is at the position p with food then the agent observes the food (no matter if it is visible, e.g., by smell)

Behavioural properties

Next, dynamic properties are identified that characterise the input-output correlation of delayed response behaviour, observed from an external viewpoint. Such a dynamic property has a temporal nature; it can refer to the agent's input and output in the present, the past and/or the future. In semi-formal and formal notation, for the case considered, the input-output correlation for delayed response behaviour can be characterised by:

BP3 Going to food observed in the past
At any point in time t, if the agent observes itself at position p and it observes no food at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and at some earlier point in time t1 the agent observed that an amount of food x was present at position p' and at every point in time t2 after t1 up to t, the agent did not observe that no food was present at p' then at the next time point after t the agent will go to position p'

$\forall t\ \forall x\ \forall i\ \forall p\ \forall p'$
$[\ [\ state(\gamma, t, input(agent)) \models observed(at(agent, p)) \wedge observed(not(at(food(x), p))) \wedge$
$observed(accessible(p')) \wedge observed(distance(p, p', i)) \ \&\ i \leq max\_dist\ ] \ \&$
$\exists t1 < t\ [state(\gamma, t1, input(agent)) \models observed(at(food(x), p'))\ \&$

[1] Formal expressions for all properties can be found in the Appendix at http://www.cs.vu.nl/~tbosse/complexity.

$\forall t2$ [$t \geq t2 > t1 \Rightarrow$ state(γ, t2, input(agent))|= not(observed(not(at(food(x), p'))))]]
$\Rightarrow$ state(γ, t+1, output(agent)) |= performing_action(goto(p'))]

Cognitive properties
Since the external characterisations of delayed response behaviour refer to the agent's input in the past, it is assumed that internally the agent maintains past observations by means of persisting internal state properties, i.e., some form of memory. These persisting state properties are sometimes called *beliefs*. For the example case, it is assumed that an internal state property b1(p) is available, with the following dynamics:

CP1 Belief formation on food presence
At any point in time t, if the agent observes that food is present at position p then internal state property b1(p) will hold (i.e., a belief that food is present at p)

CP2 Belief b1 persistence
At any point in time t, if internal state property b1(p) holds and the agent does not observe the absence of food at position p then at the next time point internal state property b1(p) still holds

CP3 Going to food believed present
At any point in time t, if the agent observes itself at position p and it observes no food at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and $p \neq p'$ and internal state property b1(p') holds then the agent will go to position p'

Logical relations
ES2 $\equiv$ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP10 & EP11 & EP12
BS2 $\equiv$ BP2 & BP3
CS2 $\equiv$ BP2 & CP1 & CP2 & CP3
BS2 & ES2 $\Rightarrow$ VP

$$CS2 \Rightarrow BS2$$

5.3 Goal-Directed Behaviour

A next, more complex type of behaviour considered is goal-directed behaviour. This behaviour is able to cope with environments where visibility can be more limited than in the environments considered before.

Environmental properties
For this case the environment is characterised by the following expression ES3 based on a subset of the environmental properties given earlier, extended with property EP13, given below:

ES3 $\equiv$ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP11 & EP12 &
(EP10 OR (EP8 & EP13))

EP13 Complete invisibility of food
Food is always invisible for the agent (e.g., always covered), unless the agent is at the same position as the food.

Behavioural properties
The agent's behaviour exploring positions in order to discover food is characterised by the following behavioural property:

BP4 Searching for food
At any point in time t, if the agent observes itself at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and it did not visit position p' yet and p' is the position closest to p which the agent did not visit and it did not observe any food at all yet then at the next time point after t the agent will go to position p'

$\forall t\ \forall p\ \forall p'$
state(γ, t, input(agent)) |= observed(at(agent, p)) $\wedge$ observed(accessible(p')) $\wedge$
observed(distance(p, p', i)) & i$\leq$max_dist &
  not [$\exists t'\ t'<t$ & state(γ, t', input(agent)) |= observed_at(agent, p')] &
  $\forall p''$ [[not [$\exists t'\ t'<t$ & state(γ, t', input(agent)) |= observed_at(agent, p'')]]
    $\Rightarrow \exists d1\ \exists d2$ state(γ, t, input(agent)) |= observed(distance(p, p', d1)) $\wedge$
    observed(distance(p, p'', d2)) & d1<d2] &
    not [$\exists t'\ \exists p''\ \exists x\ t'\leq t$ & state(γ, t', input(agent)) |= observed(at(food(x), p''))]
      $\Rightarrow$ state(γ, t+1, output(agent)) |= performing_action(goto(p'))

Cognitive properties

To describe the internal cognitive process generating this type of behaviour, the mental state property *goal* is used. In particular, for the case addressed here, when the agent has no beliefs about the presence of food, it will generate the goal to find food. If it has this goal, it will pro-actively search for food in unexplored positions. This is characterised by the following dynamic properties:

CP4 Goal formation
At any point in time t, if the agent does not believe that food is present at any position p then it will have the goal to find food

CP5 Non-goal formation
At any point in time t, if the agent believes that food is present at position p then it will not have the goal to find food

CP6 Belief formation on visited position
At any point in time t, if the agent observes itself at position p then internal state property b2(p) will hold (i.e., the belief that it visited p)

CP7 Belief b2 persistence
At any point in time t, if internal state property b2(p) holds then at the next time point internal state property b2(p) still holds

CP8 Belief formation on distances
At any point in time t, if the agent observes that the distance between position p and p' is d then internal state property belief(p, p', d) will hold

CP9 Belief persistence on distances
At any point in time t, if internal state property belief(p, p', d) holds then at the next time point internal state property belief(p, p', d) still holds

CP10 Going to closest position
At any point in time t, if the agent observes itself at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and it has the goal to find food and it believes it did not visit p' yet and p' is the position closest to p of which the agent believes it did not visit it then at the next time point after t the agent will go to position p'

Logical relations

ES3 $\equiv$ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP11 & EP12 &
(EP10 OR (EP8 & EP13))
BS3 $\equiv$ BP2 & BP3 & BP4
CS3 $\equiv$ BP2 & CP1 & CP2 & CP3 & CP4 & CP5 & CP6 & CP7 & CP8 & CP9 & CP10
BS3 & ES3 $\Rightarrow$ VP

CS3 ⇒ BS3

5.4 Learning Behaviour

A final class of behaviour analysed is learning behaviour. In this case, depending on its history comprising a (possibly large) number of events, the agent's externally observable behaviour is tuned to the environment's dynamics. In the case addressed here, in contrast to the earlier cases, the environment has no guaranteed persistence of food for all positions. Instead, at certain positions food may come and go (e.g., because it is eaten by competitors). The agent has to learn that, when food often appears (and disappears) at a certain position, then this is an interesting position to be, because food may re-appear at that position (but soon disappear again).

Environmental properties

For this case the environment is characterised by the following expression ES4 based on a subset of the environmental properties given earlier, extended with property EP14, given below.

ES4 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & ((EP6 & EP7 & EP10 & EP11 & EP12)
OR (EP6 & EP7 & EP8 & EP11 & EP12 & EP13) OR (EP9 & EP14))

EP14 Food reoccurrence

Every piece of food disappears and reappears at least 2 times per day, of which at least after the second disappearance its position will be accessible.

Behavioural properties

The agent's behaviour for this case should take into account which positions show reoccurence of food. The following behavioural property characterises this.

BP5 Being at useful positions

At any point in time t, if the agent observes itself at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and for all positions p'' that the agent observed food in the past, the agent later observed that the food disappeared and at some earlier point in time t1 the agent observed that food was present at position p' and after that at time point t2 before t the agent observed no food present at position p' and after that at time point t3 before t the agent again observed the presence of food at position p' and after that at a time point t4 before t the agent again observed no food present at position p' and p' is the closest reachable position for which the above four conditions hold then at the next time point after t the agent will go to position p'

∀t ∀p ∀p' ∀x
state(γ, t, input(agent)) |= observed(at(agent, p)) ∧
observed(accessible(p')) ∧ observed(distance(p, p', i)) & i≤max_dist &
∀t' ∀p'' ∀x' [t'<t & state(γ, t', input(agent)) |= observed(at(food(x'), p''))
⇒ ∃t'' t'<t''≤t &
state(γ, t'', input(agent)) |= observed(not(at(food(x'), p'')))]
& ∃t1 ∃t2 ∃t3 ∃t4 [t1<t2<t3<t4<t &
state(γ, t1, input(agent)) |= observed(at(food(x), p')) &
state(γ, t2, input(agent)) |= observed(not(at(food(x), p'))) &
state(γ, t3, input(agent)) |= observed(at(food(x), p')) &
state(γ, t4, input(agent)) |= observed(not(at(food(x), p')))]
& ∀p" [∃t1 ∃t2 ∃t3 ∃t4 [t1<t2<t3<t4 &
state(γ, t1, input(agent)) |= observed(at(food(x), p")) &
state(γ, t2, input(agent)) |= observed(not(at(food(x), p"))) &

state(γ, t3, input(agent)) |= observed(at(food(x), p")) &
state(γ, t4, input(agent)) |= observed(not(at(food(x), p")))] ⇒
∃d1 ∃d2
state(γ, t, input(agent)) |= observed(distance(p, p', d1)) ∧
observed(distance(p, p", d2)) & d1<d2]
⇒ state(γ, t+1, output(agent)) |= performing_action(goto(p'))

Cognitive properties

The internal cognitive dynamics has to take into account longer histories of positions and food (re)appearing there. This is realised by representations that are built up for more complex world properties, in particular, not properties of single states but of histories of states of the world. For example, at a certain time point, it has to be represented that for a certain position in the past food has appeared twice and in between disappeared. The state properties b3(p, q) play the role of representations of world histories on food (re)occurrence.

CP11 Initial mental state
At the beginning of every day n, for all positions p, internal state property b3(p, 0) holds (i.e. a belief that there is no food at p)

CP12 Belief update on food presence
At any point in time t, for q ∈ {0,2}, if internal state property b3(p, q) holds and the agent observes food at position p then internal state property b3b(p, q+1) will hold

CP13 Belief update on food absence
At any point in time t, for q ∈ {1,3}, if internal state property b3(p,q) holds and the agent observes no food at position p then internal state property b3(p,q+1) will hold

CP14 Belief b3 persistence
At any point in time t, for all q, if internal state property b3(p,q) holds then at the next time point internal state property b3(p,q) still holds

CP15 Going to interesting position
At any point in time t, if the agent observes itself at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and it has the goal to find food and p' is the position closest to p of which the agent believes that it is an attractive position then at the next time point after t the agent will go to position p'

Here, b3(p,4) represents the belief that food was twice present at p, and subsequently disappeared (in other words, a belief that p is an attractive position, since food might show up again). Note that, although the mechanism described here is quite different from, e.g., machine learning, this type of behaviour nevertheless can be qualified as learning behaviour. The reason for this is that the behaviour can be split into two distinct phases: one in which nothing was learned, and one in which the agent has learned which positions are useful by maintaining a history of previous observations.

Logical relations

ES4 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & ((EP6 & EP7 & EP10 & EP11 & EP12)
OR (EP6 & EP7 & EP8 & EP11 & EP12 & EP13) OR (EP9 & EP14))
BS4 ≡ BP2 & BP3 & BP4 & BP5
CS4 ≡ BP2 & CP1 & CP2 & CP3 & CP4 & CP5 & CP6 & CP7 & CP8 & CP9 & CP10 &
CP11 & CP12 & CP13 & CP14 & CP15
BS4 & ES4 ⇒ VP
CS4 ⇒ BS4

6 Formalisation of Temporal Complexity

The Complexity Monotonicity Thesis discussed earlier involves environmental, behavioural and cognitive dynamics of living systems. In Section 2 it was shown that based on a given complexity measure cm this thesis can be formalised by:

$cm(E_1) \leq cm(E_2) \Rightarrow cm(B_1) \leq cm(B_2)$ &
$cm(B_1) \leq cm(B_2) \Rightarrow cm(C_1) \leq cm(C_2)$

What remains is the existence or choice of the complexity measure function cm. To measure degrees of complexity for the three aspects considered, a temporal perspective is chosen: complexity in terms of the temporal relationships describing them. For example, if references have to be made to a larger number of events that happened at different time points in the past, the temporal complexity is higher. The temporal relationships have been formalised in the temporal language TTL based on predicate logic. This translates the question how to measure complexity to the question how to define complexity of syntactical expressions in such a language. In the literature an approach is available to define complexity of expressions in predicate logic in general by defining a function that assigns a *size* to every expression [7]. To measure complexity, this approach was adopted and specialised to the case of the temporal language TTL. Roughly spoken, the complexity (or size) of an expression is (recursively) calculated as the sum of the complexities of its components plus 1 for the composing operator. In more details it runs as follows.

Similarly to the standard predicate logic, predicates in the TTL are defined as relations on terms. The size of a TTL-term t is a positive natural number $s(t)$ recursively defined as follows:

(1) $s(x)=1$, for all variables x.
(2) $s(c)=1$, for all constant symbols c.
(3) $s(f(t1,\ldots, t_n))= s(t1) + \ldots + s(t_n) + 1$, for all function symbols f.

For example, the size of the term observed(not(at(food(x), p))) from the property BP1 (see the Appendix) is equal to 6.

Furthermore, the size of a TTL-formula ψ is a positive natural number $s(\psi)$ recursively defined as follows:

(1) $s(p(t_1,\ldots, t_n))= s(t_1) + \ldots + s(t_n) +1$, for all predicate symbols p.
(2) $s(\neg\varphi)=s((\forall x)\ \varphi)= s((\exists x)\ \varphi) = s(\varphi)+1$, for all TTL-formulae φ and variables x.
(3) $s(\varphi\&\chi) = s(\varphi|\chi) = s(\varphi\Rightarrow\chi) = s(\varphi)+ s(\chi)+1$, for all TTL-formulae φ, χ.

In this way, for example, the complexity of behavioural property BP1 amounts to 53, and the complexity of behavioural property BP2 is 32. As a result, the complexity of the complete behavioural specification for the stimulus-response case (which is determined by BP1 & BP2) is 85.

Using this formalisation of a complexity measure as the size function defined above, the complexity measures for environmental, internal cognitive, and behavioural dynamics for the considered cases of stimulus-response, delayed response, goal-directed and learning behaviours have been determined. Table 1 provides the results (see the Appendix for all properties).

Table 1. Temporal complexity of environmental, behavioural and cognitive dynamics.

Case	Environmental dynamics	Behavioural dynamics	Cognitive dynamics
Stimulus-response	262	85	85
Delayed response	345	119	152
Goal-directed	387	234	352
Learning	661	476	562

The data given in Table 1 confirm the Complexity Monotonicity Thesis put forward in this paper, that the more complex the environmental dynamics, the more complex the types of behaviour an agent needs to deal with the environmental complexity, and the more complex the behaviour, the more complex the internal cognitive dynamics.

7 Discussion

In this paper, the temporal complexity of environmental, behavioural, and cognitive dynamics, and their mutual dependencies, were explored. As a refinement of Godfrey-Smith's Environmental Complexity Thesis [4], the Complexity Monotonicity Thesis was formulated: for more complex environments, more complex behaviours are needed, and more complex behaviours need more complex internal cognitive dynamics. A number of example scenarios were formalised in a temporal language, and the complexity of these formalisations was measured. Complexity of environment, behaviour and cognition was taken as temporal complexity of dynamics of these three aspects, and the formalisation of the measurement of this temporal complexity was based on the complexity of the syntactic expressions to characterise these dynamics in a predicate logic language, as known from, e.g., [7]. The outcome of this approach is that the results support the Complexity Monotonicity Thesis.

Obviously, the results as reported in this paper are no generic proof for the correctness of the Complexity Monotonicity Thesis. Instead, the paper should rather be seen as a case study in which the thesis was tested positively. However, the approach taken for this test was not completely arbitrary: the used complexity measure is one of the standard approaches to measure complexity of syntactical expressions [7]. Moreover, the formal specifications were constructed very carefully, to ensure that no shorter specifications exist that are equivalent. Although no formal proof is given that the used specifications are indeed the shortest possible ones, the construction of these specifications has been an iterative process in which multiple authors have participated. To represent the specifications, the language TTL was just used as a vehicle. Various similar temporal languages could have been used instead, but we predict that this would not significantly influence the results.

Nevertheless, there are a number of alternative possibilities for measuring complexity that might in fact influence the results. Among these is the option to use complexity measures from information theory based on the amount of entropy of a system, such as [1]. In future work, such alternatives will be considered as well.

Another challenging direction for future work is the possibility to establish a uniform approach for specification of dynamic properties for environment, behaviour, and cognition. Such an approach may, for example, prescribe a limited number of predefined concepts that can be used within the dynamic properties.

Another issue that is worth some discussion is the fact that the Complexity Monotonicity Thesis can also be considered in isolation of Godfrey-Smith's Environmental Complexity Thesis. Although it was used as a source of inspiration to explore for the more refined Complexity Monotonicity Thesis, the Environmental Complexity Thesis as such was not investigated in this paper. Doing this, again from an agent-based modelling perspective, is another direction for future work. To this end, techniques from the area of Artificial Life may be exploited, e.g., to perform social simulations and observe whether more complex agents evolve in a way that supports the Environmental Complexity Thesis.

Furthermore, organizations can be also considered as structures that often create additional complexity in the environment, with which organisms need to cope. Therefore, some results presented in this paper can be also applied in organizational context. However, a more detailed investigation is still required.

In [4], in particular in Chapters 7 and 8, mathematical models are discussed to support the Environmental Complexity Thesis, following, among others [9] and [12]. These models are made at an abstract level, abstracting from the temporal dimension of the behaviour and the underlying cognitive architectures and processes. Therefore, the more detailed temporal complexity as addressed in this paper is not covered. Based on the model considered, Godfrey-Smith [4] concludes that the flexibility to accommodate behaviour to environmental conditions, as offered by cognition, is favoured when the environment shows (i) unpredictability in distal conditions of importance to the agent, and (ii) predictability in the links between (observable) proximal and distal. This conclusion has been confirmed to a large extent by the formal analysis described in this paper. Comparable claims on the evolutionary development of learning capabilities in animals are made in work such as [13] and [10]. According to these authors, learning is an adaptation to environmental change. All these are conclusions at a global level, compared to the more detailed types of temporal complexity considered in our paper, where cognitive processes and behaviour extend over time, and their complexity can be measured in a more detailed manner as temporal complexity of their dynamics.

References

1. Berlinger, E. (1980). An information theory based complexity measure. In Proceedings of the Natural Computer Conference, pp. 773-779.
2. Bosse, T., Jonker, C.M., Meij, L. van der, Sharpanskykh, A., and Treur, J. (2006). Specification and Verification of Dynamics in Cognitive Agent Models. In: *Proceedings of the Sixth International Conference on Intelligent Agent Technology, IAT'06*. IEEE Computer Society Press, 2006, pp. 247-254.
3. Darwin, C. (1871). *The Descent of Man*. John Murray, London.
4. Godfrey-Smith, P., (1996). *Complexity and the Function of Mind in Nature*. Cambridge University Press.

5. Hills, T.T. (2006). Animal Foraging and the Evolution of Goal-Directed Cognition. *Cognitive Science*, vol. 30, pp. 3-41.
6. Hunter, W.S. (1912). The delayed reaction in animals. *Behavioral Monographs*, 2, 1912, pp. 1-85
7. Huth, M. and Ryan, M. (2000). Logic in Computer Science: Modelling and reasoning about computer systems, Cambridge University Press.
8. McMillan, K.L. (1993). Symbolic Model Checking: An Approach to the State Explosion Problem. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1992. Published by Kluwer Academic Publishers, 1993.
9. Moran, N. (1992). The evolutionary maintenance of alternative phenotypes. *American Naturalist*, vol. 139, pp. 971-989.
10. Plotkin, H. C. and Odling-Smee, F. J. (1979). Learning, Change and Evolution. *Advances in the Study of Behaviour 10*, pp. 1-41.
11. Sharpanskykh, A., Treur, J. (2006). Verifying Interlevel Relations within Multi-Agent Systems. In: *Proceedings of the 17th European Conference on Artificial Intelligence, ECAI'06*. IOS Press, 2006, pp. 290-294.
12. Sober, E. (1994). The adaptive advantage of learning versus a priori prejustice. In: *From a Biological Point of View*. Cambridge University Press, Cambridge.
13. Stephens, D. (1991). Change, regularity and value in evolution of animal learning. *Behavioral Ecology*, vol. 2, pp. 77-89.
14. Tinklepaugh, O.L. (1932). Multiple delayed reaction with chimpanzees and monkeys. *Journal of Comparative Psychology*, 13, 1932, pp. 207-243.
15. Vauclair, J. (1996). *Animal Cognition*. Harvard Univerity Press, Cambridge, MA.
16. Wilson, O. (1992). *The Diversity of Life*. Harvard University Press, Cambridge, MA.

Part IV

Supporting Organization Design

This part describes principles of an organization design process using the framework based on the modeling methods for particular views described in Part III.

More specifically, Chapter 1 describes a set of general methodological guidelines for using the framework in the process of organization design. These guidelines link and generalize the specific methodological principles for (re)designing specifications of particular views introduced in Part III by defining a set of steps of an organization design process along all modeling views. The general guidelines help an organization designer create an organizational specification from scratch as well as offer the possibility to revise existing specifications of organizations.

To automate the process of organization design, the identified general methodological guidelines should be made formal and operational. Chapter 2 formalizes the design process of specifications from the organization-oriented view by introducing a set of formal design operators. These operators can be combined into complex operators that can serve as patterns for larger steps in an organization design process. Furthermore, the contribution of this Chapter provides a solid basis for the development of a software environment supporting interactive organization design processes.

Chapter 1

General Approaches to Organization Design [1]

The general approaches to organization design differ with respect to the presence and involvement of the concerned agents. The design can be performed without having in mind specific agents - the necessary agent profiles are composed at the later design stages based on the considered/designed tasks. Organizational design can also be performed with respect to a (partially) known set of agents who will take roles in the organization. Thus agents' skills and traits can be taken into account. Sometimes the agents are not only known but they have some degree of power to steer the design process.

The design process often starts with the identification of one or more high-level goals which play the role of the driving force behind the design process. These goals (initially still informally defined) should answer the question: why should the organization exist and what purpose will it serve? Such goals can be identified by the designer or emerge through communication and/or negotiation between the involved agents. In the second case the resulting organizational goals reflect to some extent the individual goals of the participating agents. In this way some possible future conflicts between individual and organizational goals are prevented early. If conflicts do appear, they can be dealt with through negotiation and redesign at the later stages.

The higher-level goals are often more abstract and, through refinement, more specific, easier to evaluate, goals are formulated. Also, often the higher-level goals are long-term, strategic goals while their sub-goals are shorter-term tactical or operational goals. The leaves of the hierarchies should be goals formulated so that the corresponding PIs can clearly be associated to the processes in the workflow. In this way the satisfaction of every goal in the hierarchies can be evaluated.

[1] This chapter appeared as a part of the paper: Popova, V., Sharpanskykh A.: A Formal Framework for Modeling and Analysis of Organizations. In: Ralyte, J., Brinkkemper, S., Henderson-Sellers, B. (eds.), Proceedings of the Situational Method Engineering Conference, ME'07, Springer Verlag (2007) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

Also at the earlier stage of the design process one or more general tasks are identified giving an answer to the question: what should the organization do? For identifying these tasks sometimes only the defined goals are considered. However when the involved agents are (partially) known, the definition of tasks can be based on the available skills and experience as well. These tasks are later refined to task hierarchies. For the tasks, the used / produced resource types are identified which can also form hierarchies. Based on the tasks, processes are defined and organized into workflows that can represent different levels of abstraction. The level of elaboration of these structures can depend on the type of the organization. In mechanistic organizations [4] the procedures are prescribed to a great degree of detail which should result in more elaborate structures refined to simple tasks and processes. In organic organizations (e.g., adhocracies) the procedures are described at a higher level of abstraction leaving more freedom to the agents to choose how to perform them which should result in less deep task hierarchies and less elaborate workflows.

The design process can follow different paths through the views and concepts but several general guidelines can be formulated. When an informally defined goal is being formalized and made more precise this should be reflected on the PI structure - often this means that a new PI is defined or an existing one is revised. A change in the goal hierarchy should also be reflected on the task hierarchy by identifying new or existing tasks that can realize the new or revised goals. A change in the task hierarchy often brings changes to the current workflow design. Adding or revising processes in the workflow might give rise to new PIs that need to be monitored. When a PI is proposed it should be decided on its level of importance in order to find out if a new goal should be formulated based on it. The definition of roles is based on the currently defined tasks and processes. Fig.1 shows the main dependencies between concepts and structures in the framework which guide the design process.

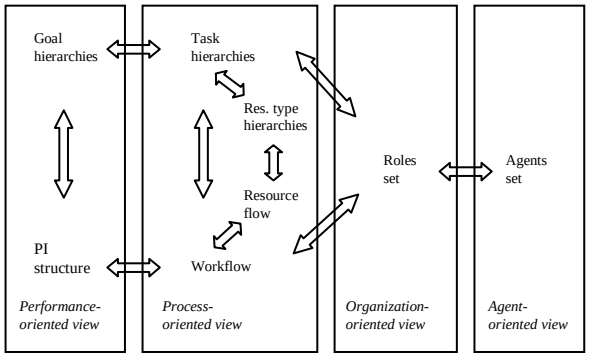


Fig. 1. Dependencies between the structures of the four views

Power and authority relations between the defined roles are usually assigned at the later stages of the design process. However different general schemes can be predefined and committed to by the designer at the earlier stages as well leaving the details for later. Such schemes reflect different types of organizations identified in

organization theory such as: hierarchical, flat or team-based organizations which differ in the way the power is distributed, granted or accepted by the roles (agents).

The choice of scheme should be driven by an analysis of the environment in which the organization should operate. For example a relatively stable environment tolerates a well-defined hierarchical structure which can help the organization to operate more efficiently. A changing environment can be addressed by designing a lighter, more flexible and dynamic structure that can easily adapt to the changes. Obviously the environment in which the organization will be situated plays an important role not only in defining power and authority relations. It needs to be taken into account at every step of the design process and in every view of the framework.

Sometimes instead of designing an organization from scratch, a specification of an existing one needs to be created. Here a wide range of internal or external documents are used, e.g., company policies, job descriptions, mission statement, business plans, procedure descriptions, laws. However even the richest documentation leaves some information unspecified thus it is essential to involve domain experts and managers. In organizational redesign, the issue of maintaining the consistency and correspondence between the structures of different views becomes more complex and the tools for automatic analysis become indispensable.

The framework allows reuse in a number of ways. Libraries of commonly appearing parts of structures (goals and tasks hierarchies, PI-structures, workflow graphs, etc.) can be stored and reused for organizations in the same domain. The research in identifying and classifying important PIs for different domains [e.g. 1, 2] can easily be applied here. Reuse can also be supported by predefined templates for various aspects of different types of organizations (mechanistic, organic, etc.). For example templates for domain-specific constraints can be provided for each view to be customized by the designer. The used tool allows defining parameterized templates (macros) for TTL formulae that can be instantiated in different ways which can also be used as support for designers not skilled in logics. For more details see [3].

References

1. F.T.S. Chan, Performance measurement in a supply chain, *International Journal of Advanced Manufacturing Technology* 21(7), 534-548 (2003).
2. E. Krauth, H. Moonen, V. Popova, and M. Schut, Performance Measurement and Control in Logistics Service Providing, in: *Proceedings of Seventh International Conference on Enterprise Information Systems, ICEIS 2005*, edited by C.-S. Chen et al., pp. 239-247 (2005).
3. V. Popova and A. Sharpanskykh, Process-Oriented Organization Modeling and Analysis Based on Constraints. Technical Report 062911AI, VUA, http://hdl.handle.net/1871/10545
4. W.R. Scott, *Institutions and organizations* (SAGE Publications, Thousand Oaks 2001).

Chapter 2

A Formal Framework to Support Organization Design [1]

Abstract. Organizational design is an important topic in the literature on organizations. Usually the design principles are addressed informally in this literature. This paper makes a first attempt to formally introduce design operators to formalize the design steps in the process of designing organizations. These operators help an organization designer create an organization design from scratch as well as offer the possibility to revise existing designs of organizations. The operators offer both top-down refinements and bottom-up grouping options. Importantly, the operators can be combined into complex operators that can serve as patterns for larger steps in an organization design process. The usability of the design operators is demonstrated in a running example. The contribution of this paper provides a solid basis for the development of a software environment supporting interactive organization design processes. This is demonstrated by an implemented prototype example tool.

1 Introduction

Organizations play a key role in the modern society. To a large extent vitality and productivity of an organization situated in the environment of a certain type depend on kinds of structure and behavior of the organization that should conform to the environmental conditions. Organizational structures and processes are studied in social sciences, where organizational design is a special topic. Organization design is

[1] Part of this chapter appeared as: Jonker, C.M., Sharpanskykh, A., Treur, J., Yolum, P.: Design Operators to Support Organizational Design. In: J. Gero (ed.), Proc. of the Second International Conference on Design Computing and Cognition, DCC'06, Springer Verlag, 203-222 (2006) (the names of the authors are ordered alphabetically reflecting the comparable contribution of each author).

concerned "with what an organization is ought to be" (Pfeffer 1978). More specifically, Galbaith (1978) stated that organization design "is conceived to be a decision process to bring about a coherence between the goals or purposes for which the organization exists, the patterns of division of labor and interunit coordination and the people who will do the work". Further Galbaith argues that design is an essential process for "creating organizations, which perform better than those, which arise naturally".

In literature, a range of theories and guidelines concerning the design of organizations are present (Galbraith 1978; Duncan 1979; Minzberg 1993; Blau and Schoenherr 1971). For example, Duncan proposed a contingency model for designing organizations with environmental variables being the principal determinants of organizational models. Minzberg described a number of guidelines applicable mostly for designing hierarchical organizations that function in a relatively stable environment. However, despite the abundance of organizational design theories no general principles applicable to organizational design in all times and places can be identified (Scott, 1998). Moreover, almost all theoretical findings in organizational design are informal and often vague. In order to provide an organization designer or a manager with operational automated tools for creating, analyzing, and revising organizations, in the first place a formal representation of an organization model as a design object description should be provided. In addition to this, to address the operations performed on such design object descriptions during a design process, a formal representation of design operators underlying possible design steps is needed. Such design operators describe the possible transitions between design object descriptions. Using the design operators, a design process can be described by, at the various points in time, choosing a next operator to be applied to transform the current design object description into the next one. Examples of very simple design operators are adding or deleting an element of a design object description. More sophisticated design operators can involve, for example, the introduction of further refinement of the aggregation levels within a design object description.

In this paper we introduce a formal organizational model format, to be used to represent design object descriptions. On top of this, a set of design operators is formally defined. The formalization is based on the sorted predicate logic (Manzano 1996).

Often in the literature organizational design is recognized as an engineering problem (Child 1973). From this perspective design is considered as a continuous process of a gradual change of an organizational model by applying certain operations (Pfeffer 1978). For example, Minzberg (1993) describes design process as the following sequence of operations: given overall organizational needs, a designer refines the needs into specific tasks, which are further combined into positions. The next step is to build the "superstructure" by performing unit grouping using special guidelines and heuristics (e.g., grouping by knowledge and skill, by work process and function, by time, by place, etc.). Then, the grouping process is repeated recursively, until the organization hierarchy is complete.

For this paper we aimed at identifying the most commonly and generally used set of operators for designing organizations. For this purpose the literature from social sciences, and design principles used in other disciplines were investigated. For example, useful principles for organizational design can be found in the area of

derivative grammars. Thus, graphical changes in organizational designs may be described by shape (Stiny 1991) and graph grammars (Rozenberg 1997). Whereas changes in textual (or symbolic) structural and dynamic descriptions of organizational elements may be specified by string (Chomsky 1965) and graph grammars, which allow representation of relationships between descriptions of different elements. In order to relate graphical organizational designs to designs described in a symbolic form, parallel grammars (or grammars defined in multiple algebras) may be used (Stiny 1991). For designing organization structures with multiple levels of representation (e.g., hierarchical organizations with departments, groups, sections) abstraction grammars (Schmidt and Cagan 1995) and hierarchical graph grammars (Habel and Hoffmann 2004) can be useful. By means of abstraction grammars, design is performed from the top level of the abstraction hierarchy to the bottom (most concrete) level, with each design generation using the prior level design as a pattern. Furthermore, mechanisms for choosing the most appropriate design generated by different transformations defined by grammars have been developed in different areas (e.g. recursive annealing in mechanical design (Schmidt and Cagan 1995)). Although it is widely recognized in social studies that no "best" design of an organization exists, a number of informal guidelines and best practices developed in the area of organizational design can help in identifying the most suitable organizational designs.

Thus, based on the rich literature on design, this paper makes a first attempt to formalize the operators underlying organization design processes. A set of design operators is formally introduced, which provides the means for creating a design of an organization from scratch as well as revising existing designs for organizations. Furthermore, the formalization of operators provides a solid basis for a software tool supporting interactive organization design processes.

In Section 2 a formal framework for the specification of design object descriptions for organizations is described. Sections 3 and 4 introduce a set of classes of operators to create and modify design object descriptions for organizations. Section 5 illustrates the application of a developed prototype by an example. Finally, Section 6 discusses future work and provides general conclusions.

2 Format for an Organizational Model as a Design Object Description

We consider a generic organization model, abstracted from the specific instances of agents (actors), which consists only of structural and behavioral descriptions of organizational roles and relations between them. A top-down ordering of definitions is used, meaning that concepts are referred before they are defined.

Definition 1 (Organization)
A specification of an organization with the name O is described by the relation is_org_described_by(O, Γ, Δ), where Γ is a structural description and Δ is a description of dynamics.

An organizational structure is characterized by the patterns of relationships or activities in an organization, and described by sets of roles, groups, interaction and interaction links, relations between them and an environment.

Definition 2 (Organization Structure)

A structural description Γ of an organizational specification described by the relation is_org_described_by(O, Γ, Δ) is determined by a set of relations, among which[1]:

- a relation has_basic_components(Γ, R, G, IL, ILL, ONT, M, ENV) defined on the subsets R, G, IL, ILL, ONT, M, ENV of the corresponding general sets ROLE (the set of all possible role names), GROUP (the set of all possible group names), INTERACTION_LINK (the set of all possible interaction links names), INTERLEVEL_LINK (the set of all possible interlevel links names), ONTOLOGY (the set of all possible ontology names), ONTO_MAPPING (the set of all possible ontology mappings names), ENVIRONMENT (the set of all possible environment names)[2]
- a relation for specifying a role $r \in R$ in Γ is_role_in(r, Γ)
- a relation for specifying an interaction link $e \in IL$ in Γ is_interaction_link_in (e, Γ)
- a relation for specifying an interlevel link $il \in ILL$ in Γ is_interlevel_link_in(il, Γ)
- a relation for specifying an environment $env \in ENV$ is_environment_in(env, ENV)
- a relation has_input_ontology(r, o) that assigns an input ontology $o \in ONT$ to a role $r \in R$ (similarly the relations for output, internal, and interaction ontologies are introduced: has_output_ontology(r, o), has_interaction_ontology(r, o), has_internal_ontology(r, o))
- a relation has_input_ontology(env, o) that assigns an input ontology $o \in ONT$ to an environment $env \in ENV$ (similarly the relations for output, internal, and interaction ontologies are introduced: has_output_ontology(env, o), has_interaction_ontology(env, o), has_internal_ontology(env, o))
- a relation is_ontology_for(el, o) that assigns an ontology $o \in ONT$ either to a role $el \in R$ or an environment $el \in ENV$
- a relation has_onto_mapping(il, m) that associates an interlevel link $il \in IL$ with an ontology mapping $m \in M$ (an ontology mapping for an interaction link is defined similarly)
- a relation is_interaction_link_of_type(e, type) that specifies an interaction link $e \in IL$ of one of the types: role_interaction_link, env_input_link, env_output_link
- a relation connects_to(e, r, r', Γ) that specifies a connection by an interaction link $e \in IL$ from a source-role $r \in R$ to a destination role $r' \in R$ in Γ
- a relation connects_to(e, env, r, Γ) that specifies a connection by an interaction link $e \in IL$ of type env_output_link from an environment $env \in ENV$ to a role $r \in R$ in Γ (similarly for connects_to(e, r, env, Γ))
- a relation subrole_of_in(r', r, Γ) that specifies a subrole $r' \in R$ of a role $r \in R$ in Γ
- a relation member_of_in(r ,g ,Γ) that specifies a member role $r \in R$ of a group $g \in G$ in Γ
- a relation interlevel_connection(il, r, r', Γ) that specifies a connection by an interlevel link $il \in ILL$ between roles $r, r' \in R$ of adjacent aggregation levels

[1] Notice that all the following relations are defined using the names of organization elements; the specifications for these elements will be provided in the following definitions

[2] The difference between R and ROLE, for example, is that R (subset of ROLE) is the set of all role names that occur in Γ.

Organizational behavior is described by dynamic properties of the organizational structure elements.

Definition 3 (Organization Dynamics)
A description of dynamics Δ of an organizational specification described by the relation is_org_described_by(O, Γ, Δ) is determined by a set of relations, among which:

- a relation has_basic_components(Δ, DP) that specifies a set of dynamic properties names DP defined in an organization model
- a relation has_dynamic_property(r, d) that specifies a dynamic property $d \in DP$ for a role $r \in R$ (the relations for dynamic properties of an interlevel link, a group and an environment are defined in a similar manner: has_dynamic_property(e, d), has_dynamic_property(g, d), has_dynamic_property(env, d))
- a relation has_expression(d, expr) that identifies a dynamic property name $d \in DP$ with a dynamic property expression $expr \in DPEXPR$ (e.g., a formula in sorted first-order predicate logic)

A role is a basic structural element of an organization. It represents a subset of functionalities, performed by an organization, abstracted from specific agents (or actors) who fulfill them. Each role has an input and an output interface, which facilitate the interaction (communication) with other roles. The interfaces are described in terms of interaction (input and output) ontologies: a vocabulary or a signature specified in order-sorted logic. An ontology contains objects that are typed with sorts, relations, and functions. Generally speaking, an input ontology determines what types of information are allowed to be transferred to the input of a role, and an output ontology predefines what kinds of information can be generated at the output of a role.

Each role can be composed of a number of other roles, until the necessary detailed level of aggregation is achieved. Thus, roles can be specified and analyzed at different aggregation levels, which correspond to different levels of an organizational structure. A role that is composed of (interacting) subroles, is called a composite role.

Definition 4 (Role)
A specification of a role r is determined by:
Objects:

- or, oi, o, o', o''$\in$ ONT, or= o $\cup$ o' $\cup$ o'', oi= o' $\cup$ o'', here $\cup$ is a functional symbol that maps names of ontologies to a name of the joint ontology

Relations:

- has_internal_ontology(r, o), has_input_ontology(r, o'), and has_output_ontology(r, o'')
- has_ontology(r, or) and has_interaction_ontology(r, oi)
- $d \in DP$, has_dynamic_property(r, d)

The ontologies, which describe interfaces of interacting roles, can be different. Therefore, if necessary, the specification of a role interaction process includes ontology mapping. An ontology mapping m between ontologies o and o' is characterized by a set of relations is_part_of_onto_map(a, a', m), where a is an atom expressed in ontology o and a' is an atom expressed using ontology o'.

Definition 5 (Ontology mapping)
An ontology mapping m between ontologies o and o' is characterized by:

- is_part_of_onto_map(a, a', m), where $a \in At(o)$ and $a' \in At(o')$
- for $a \in At(o)$ is_in_domain_of(a, m) $\Leftrightarrow \exists a' \in At(o')$ is_part_of_onto_map(a, a', m), where At(o) is the set of all atoms, expressed in ontology o.
- for $a' \in At(o')$ is_in_range_of(a', m) $\Leftrightarrow \exists a \in At(o)$ is_part_of_onto_map(a, a', m)

Roles of the same aggregation level interact with each other by means of interaction links. The interaction between roles is restricted to communication acts.

Definition 6 (Interaction link)
An interaction link e is determined by:
Relations:
- is_interaction_link_in(e, Γ)
- has_onto_mapping(e, m) for some $m \in M$
- has_dynamic_property(e, d) for a number of $d \in DP$

Constraints:
- An interaction link e should connect two roles at the same aggregation level: is_interaction_link_in(e, Γ) $\Rightarrow \exists r, r' \in R$ connects_to(e, r, r', Γ) $\wedge \neg$has_subrole(r, r') $\wedge \neg$has_subrole(r', r)

An interlevel link connects a composite role with one of its subroles. It represents an information transition between two adjacent aggregation levels. For roles connected by an interlevel link, this link is described by an ontology mapping between the corresponding elements of ontologies, part of which may be identity correspondence. Moreover, an ontology mapping associated with an interlevel link may be used for representing mechanisms of information abstraction. These mechanisms can be applied for transmitting (or generating) partial, aggregated or generalized information to the input (or from the output) of a role.

Definition 7 (Interlevel link)
A specification for an interlevel link il is determined by:
Relations:
- is_interlevel_link_in(il, Γ)
- has_onto_mapping(il, m) for some $m \in M$

Constraints:
- An interlevel link il should connect two roles at two adjacent aggregation levels: is_interlevel_link_in(il, Γ) $\Rightarrow \exists r, r' \in R$ subrole_of_in(r', r, Γ) $\wedge$ (interlevel_connection(il, r, r', Γ) $\vee$ interlevel_connection(il, r', r, Γ))

A group is a composite structural element of an organization that consists of a number of roles. In contrast to roles a group does not have well-defined input and output interfaces. Groups can be used for modeling units of organic organizations, which are characterized by loosely defined or sometimes informal frequently changing structures that operate in a dynamic environment. Furthermore, groups can be used at the intermediate design steps for identifying a collection of roles, which may be further transformed into a composite role.

Definition 8 (Group)
A group g is defined by the relations to other concepts:
- membership relation member_of_in: $r \in R$ member_of_in(r, g, Γ)
- has_dynamic_property(g, d) for a number of $d \in DP$

The conceptualized environment represents a special component of an organization model. According to some sociological theories (e.g., contingency theory), an environment represents a key determinant in organizational design, upon which an organizational model is contingent. Similarly to roles, the environment is represented in this proposal by an element having input and output interfaces, which facilitate in interaction with roles of an organization. The interfaces are conceptualized by the environment interaction (input and output) ontologies. Interaction links between roles and the environment are indicated in the organizational model as ones that have a specific type, namely env_input_link or env_output_link by means of the predicate is_interaction_link_of_type.

The internal structure of the environment is not fixed, i.e., the designer has freedom to provide his/her own conceptualization of the environment. For example, the environment can be defined by a set of objects with certain properties and states and by causal relations between objects. On the one hand, roles are capable of observing states and properties of objects in the environment; on the other hand, they can act or react and, thus, affect the environment. We distinguish passive and active observation processes. For example, when some object is observable by a role and the role continuously keeps track of its state, changing its internal representation of the object if necessary, passive observation occurs. For passive observation, no initiative of a role is needed. Active observation is always concerned with the role's initiative and focusing. For particular purposes the internal specification for the environment can be conceptualized using one of the existing world ontologies (e.g., CYC, SUMO, TOVE (Bertino, Zarri and Catania 2001)). However, despite the richness and the extensiveness of these ontological bases, more specific and refined types of concepts and relations are required for modelling particular types of organizations and environments.

The behavior of each element of an organizational structure is described by a set of dynamic properties. With each name of a dynamic property an expression is associated. Dynamic property expressions represent formulae specified over a certain ontology(ies). In particular, a dynamic property for a role is expressed using a role ontology. A dynamic property for an interaction link is constructed using the output ontology of a role-source of a link and the input ontology of a role-destination. A group dynamic property is expressed using ontologies of roles- members of a group.

Definition 9 (Dynamic Property)
A specification of a dynamic property $d \in DP$ is described by:

- has_expression(d, expr) for some $expr \in DPEXPR$
- uses_ont(d, o) for some $o \in ONT$
- if $r \in R$ and has_dynamic_property(r, d), then uses_ont(d, o) $\Rightarrow$ has_ontology(r, o)
- if $e \in IL$ and has_dynamic_property(e, d), then uses_ont(d, o) $\Rightarrow \exists r, r' \in R, \exists o', o'' \in ONT$ such that connects_to(e, r, r', Γ) $\wedge$ has_output_ontology(r, o') $\wedge$ has_input_ontology(r', o'') $\wedge$ $o \subseteq o' \cup o''$
- if $g \in G$ and has_dynamic_property(g, d), then uses_ont(d, o) $\Rightarrow \exists r \in R$ member_of_in(r, G, Γ) $\wedge$ has_ontology(r, o)

Dynamic properties expressions are specified in the Temporal Trace Language (TTL) (Jonker and Treur 2003; Sharpanskykh and Treur 2006), which is a variant of order-sorted predicate logic (Manzano 1996) To enable reasoning about the dynamic

properties the language TTL includes special sorts, such as: TIME (a set of linearly ordered time points), STATE (a set of all state names of a system), and TRACE (a set of all trace names; a trace or a trajectory can be thought of as a timeline with for each time point a state).

Definition 10 (Dynamic Property Expression)
Dynamic Property Expression is constructed as follows:

1. STATOM⊆ONT and has_expression: STATOM x STATOMEXPR where STATOM denotes a static atom in an ontology.
2. Static property expressions (STATPROPEXPR) are generated by applying conjunction, disjunction, implication, and negation operators on STATOMEXPR and STATPROPEXPR.
3. States relate to particular time points in traces (TRACE x TIME → STATE). States are related to state properties via the satisfaction relation |=, formally defined as a binary infix predicate (or by holds as a binary prefix predicate). For example, the expression state(γ: TRACE, t: TIME, output(r: ROLE)) |= p (or holds(state(γ, t, output(r)), p)) denotes that state property p holds in trace γ at time t in the output state of role r.
4. The set of all dynamic properties expressions (DPEXPR) for the corresponding dynamic properties names (DP) is inductively defined by:
   (1) If v1 is a term of sort STATE, and u1 is a term of the sort STATPROPEXPR, then holds(v1, u1) is an atomic dynamic property expression (belongs to the sort DPATOMEXPR, which is a subsort of the sort DPEXPR).
   (2) If τ1, τ2 are terms of any TTL sort, then τ1 = τ2 is an atomic dynamic property expression.
   (3) If t1, t2 are terms of sort TIME, then t1 < t2 is an atomic dynamic property expression.
   (4) The set of dynamic properties expressions (sort DPEXPR) is defined inductively based on atomic dynamic property expressions using boolean propositional connectives and quantifiers (∧, ∨, ⇒, ¬, ∃, ∀).

The application of the basic components of an organizational model is illustrated by means of a running example. Consider the process of organizing a conference. A partial model for the considered conference organization is shown in Figure 1.

At the most abstract level 0 the organization is specified by one role CO (Conference Organization) that interacts with the environment Env. Role CO can act in the environment, for example by posting a call for papers in different media. Note, that the organizational model is depicted in a modular way; i.e., components of every aggregation level can be visualized and analyzed both separately and in relation to each other. Consequently, scalability of graphical representation of an organizational model is achieved. At the first aggregation level the internal structure of the composite role CO is revealed. It consists of subrole Ch (Conference Chair), which interacts with two other subroles: OC (Organizing Committee) and PS (Paper Selection role). At the second aggregation level the internal structure of role PS is represented. It consists of subrole PCh (Program Chair), subrole PCM (Program Committee Member), and subrole R (Reviewer), which interact with each other. The input interface of role PS is connected to the input interface of its subrole PCh by means of an interlevel link. In our example the interlevel link describes the mapping between the input ontology of role PS and the input ontology of its subrole PCh. It

means that information, transmitted to the role PS at the first aggregation level, will immediately appear at the input interface of subrole PCh, expressed in terms of its input ontology at the second aggregation level.

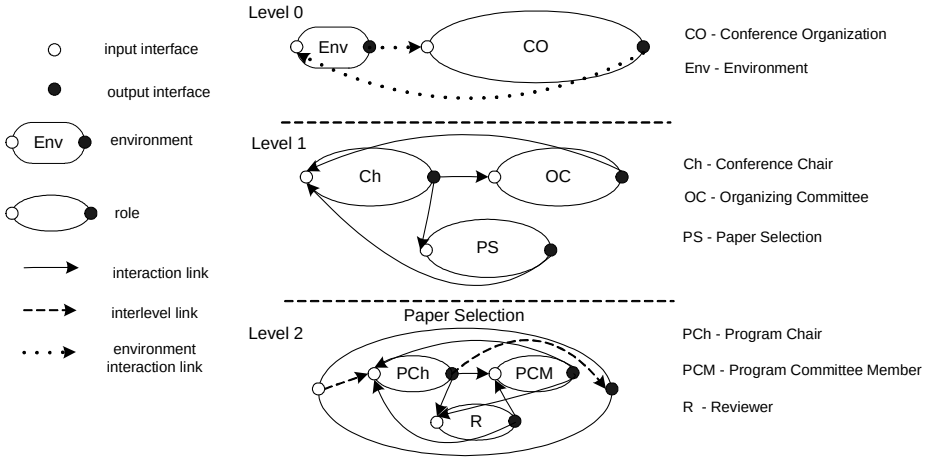


Fig. 1. Model of the conference organizing committee

For example, if Ch requests some information from PS, the request actually arrives at the input of PCh. As a result of the internal communications among PCh, PCM and R, PCh will generate a reply that will appear as a response of PS for Ch.

For each element of the considered organizational model a set of dynamic properties is identified and formally specified in TTL. In fact, these properties define constrains on behavior of elements, thus forming their expected behavioral repertoire in the organization.

For example, for role Reviewer the dynamic property may be specified expressing that a reviewer should send his/her review to the Program Chair before a certain deadline. This property is expressed in TTL as follows:

$\forall t$ state(γ, t, environment) |= deadline_for_conference(d) $\Rightarrow \exists t' < d$ state(γ, t', output(Reviewer)) |= communicate_from_to(Reviewer, Program_Chair, inform, review_report)

The predicate communicate_from_to(r1:ROLE, r2:ROLE, s_act:SPEECH_ACT, message:STRING) is used to specify the speech act s_act (e.g., inform, request, ask) from role-source r1 to role-destination r2 with the content message.

3 Representing Design Operators for Organizational Design

In this section a formal format to represent design operators and based on this format representations are introduces for a number of primitive design operators for designing organizations. Each primitive operator represents a specialized one-step operator to transform a design object description (organizational model) into a next one. Each operator is concerned with a part of the design object description to which

it will be applied and the part of the transformed design object description, resulting from the operator application. The parts of the organization O that are being modified in terms of structure and dynamics (i.e., sets of dynamic properties) are specified using the in-focus relations: structure_in_focus(O, Rf, Gf, ILf, ILLf, ONTf, Mf, ENVf) and dynamics_in_focus(O, DPf), with $Rf \subseteq R$, $Gf \subseteq G$, $ILf \subseteq IL$, $ILLf \subseteq ILL$, $ONTf \subseteq ONT$, $Mf \subseteq M$, $ENVf \subseteq ENV$, $DPf \subseteq DP$. The remaining parts of the organization stay the same.

The following operations all refer to an organization $O \in ORGANIZATION$ described by relations is_org_described_by(O, Γ, Δ), has_basic_components(Γ, R, G, IL, ILL, ONT, M, ENV). This organization is modified by an operator, leading to a second organization $O' \in ORGANIZATION$ described by relations is_org_described_by(O', Γ', Δ'), has_basic_components(Γ', R', G', IL', ILL', ONT', M', ENV').

Our choice of primitive operators is motivated by different design guidelines and theories from social sciences (Galbraith 1978; Blau and Schoenherr 1971; Lorsch and Lawrence 1970), other disciplines, and our own research on formal modeling of organizations (Broek et al 2005). However, the application of the proposed set of operators is not restricted only to these theories. Thus, a designer has freedom to choose any sequence of operators for creating models of organizations. The operators are divided into three classes, which are consecutively described in the following subsections. Thus, in Section 3.1 the operators for creating and modifying roles are specified; in Section 3.2 the operators for introducing and modifying different types of links are described; and in Section 3.3 the operators for composing and modifying groups are introduced.

3.1 OPERATORS FOR ROLES

The classes of primitive operators for creating and modifying roles in a design object description for an organization are shown in Table 1.

Table 1. Operator classes for creating and modifying roles

CLASS	DESCRIPTION
Role Introduction	Introduces a new role
Role Retraction	Deletes all links, connected to a role with their dynamic properties and mappings; deletes a role and all dynamic properties, associated with this role
Role Dynamic Property Addition	Adds a new dynamic property to a role
Role Dynamic Property Revocation	Deletes an existing role dynamic property

A *role introduction operator* adds a new role to the organization. Usually, in organizational design after organizational tasks have been identified, these tasks should be further combined into positions (roles), based on the labor division principles (Kilbridge and Wester 1966). For example, in the conference organization setting if the number of reviewers turns out to be insufficient, a Reviewer Recruiter role can be added to Paper Selection role (see Figure 2). This role, for example, may contact researchers to ask them to review for the conference by means of interaction with the environment.

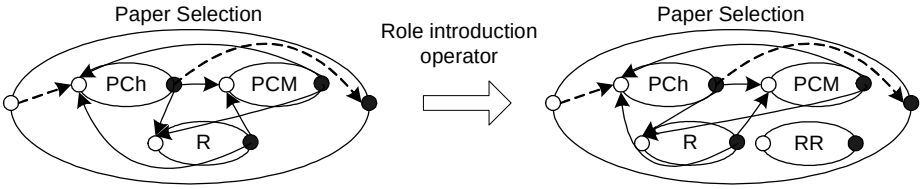


Fig. 2. Application of the role introduction operator for adding Reviewer Recruiter role (RR) into Paper Selection role

Role introduction operator
Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is a role introduction operator iff it satisfies:

1. $\delta \notin R$, $\delta \in R'$ such that is_role_in(δ, Γ')
2. structure_in_focus(O, ∅, ∅, ∅, ∅, ∅, ∅, ∅)
3. structure_in_focus(O', {δ}, ∅, ∅, ∅, ONTf', ∅, ∅), where ONTf'= is_ontology_for(o, δ) and $o \in ONT'$

A *role retraction operator* removes all links, connected to a role with their dynamic properties and mappings; it also deletes dynamic properties, associated with the role and the role itself. In the example of the conference organization, when the Reviewer Recruiter has found enough reviewers, then the role can safely be removed from the organization.

Role retraction operator
Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is a role retraction operator iff it satisfies:

1. $\delta \in R$ such that is_role_in(δ, Γ)
2. $\delta \notin R'$
3. structure_in_focus(O, {δ}, ∅, ILf, ILLf, ONTf, Mf)
   ILf={$e \in$ IL| $\exists r' \in R$ connects_to(e, δ, r', Γ) ∨ $\exists r'' \in R$ connects_to(e, r'', δ, Γ)}
   ILLf={ill$\in$ ILL| $\exists r \in R$ interlevel_connection(ill, δ, r, Γ) ∨ $\exists r' \in R$ interlevel_connection(ill, r', δ, Γ)}
   ONTf= is_ontology_for(δ, o), $o \in$ ONT
   Mf={m$\in$ M| $\exists$ill$\in$ ILLf has_onto_mapping(ill, m) ∨ $\exists e \in$ ILf has_onto_mapping(e, m)}
4. structure_in_focus(O', ∅, ∅, ∅, ∅, ∅, ∅)
5. dynamics_in_focus(O, DPf)
   DPf= {dp$\in$ DP| has_dynamic_property(δ, dp) ∨ $\exists e \in$ ILf has_dynamic_property(e, dp)}
6. dynamics_in_focus(O', ∅)

A *role dynamic property addition operator* creates a new property for the existing role in the organization and a *role dynamic property revocation operator* deletes a property from the dynamic description of a role.

Role dynamic property addition operator
Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is a role dynamic property addition operator iff it satisfies:

1. dynamics_in_focus(O, ∅)

2. dynamics_in_focus(O', DPf')
   DPf'= {δ∈ DP'| ∃r∈ R has_dynamic_property(r, δ)}

Role dynamic property revocation operator

Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is a role dynamic property revocation operator iff it satisfies:

1. dynamics_in_focus(O, DPf)
   DPf= {δ∈ DP| has_dynamic_property(r, δ)}
2. dynamics_in_focus(O', ∅)

3.2 OPERATORS FOR LINKS

In this subsection, we propose a set of classes of primitive operators for creating and modifying links in a design object description for an organization (see Table 2).

Table 2. Operator classes for creating and modifying links

CLASS	DESCRIPTION
Interaction Link Addition	Adds a new interaction link between any two roles
Interaction Link Deletion	Deletes an interaction link and all dynamic properties, associated with this link
Interlevel Link Introduction	Introduces a new interlevel link
Interlevel Link Retraction	Retracts an existing interlevel link
Interaction Dynamic Property Addition	Adds a new dynamic property to an interaction link
Interaction Dynamic Property Revocation	Deletes an existing dynamic property, associated with an interaction link

An *interaction link addition operator* allows the creation of an interaction link (information channel) between two existing roles in the organization. In the organizational design after organizational subtasks are assigned to roles, the problem of coordination of interdependencies among subtasks should be solved.

In the conference management example, the Program Chair (playing in this case a managerial role) may request two reviewers to discuss their reviews. This requirement can be handled by the addition of interaction links between the appropriate reviewer roles in the design object description for an organization (see Figure 3).

Interaction link addition operator

Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is an interaction link addition operator iff it satisfies:

1. δ∉ IL, δ∈ IL' such that is_interaction_link_in(δ, Γ')
2. structure_in_focus(O, ∅, ∅, ∅, ∅, ∅, ∅, ∅)
3. structure_in_focus(O', ∅, ∅, {δ}, ∅, ∅, Mf', ∅)
   Mf'= {m∈ M'| has_onto_mapping(δ, m)}

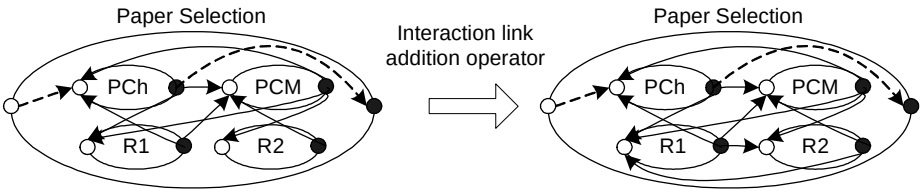


Fig. 3. Application of the interaction link addition operator for adding interaction links between Reviewer 1 role (R1) and Reviewer 2 role (R2) in Paper Selection role

An *interaction link deletion operator* is used to delete an existing interaction link between two roles as well as to revoke all dynamic properties, associated with this link. For example, the Program Chair has taken care of the acceptance proceedings for the conference. He does not need to be in contact with the reviewers any more. This case can be handled by the deletion of the interaction between two roles in the design object description for an organization.

Interaction link deletion operator
Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is an interaction link deletion operator iff it satisfies:

1. $\delta \notin IL'$, $\delta \in IL$ such that is_interaction_link_in(δ, Γ)
2. structure_in_focus(O, ∅, ∅, {δ}, ∅, ∅, Mf)
   Mf= {m∈M| has_onto_mapping(δ, m)}
3. structure_in_focus(O', ∅, ∅, ∅, ∅, ∅, ∅)
4. dynamics_in_focus(O, DPf)
   DPf= {dp∈DP| has_dynamic_property(δ, dp)}
5. dynamics_in_focus(O', ∅)

An *interaction property addition operator* creates a new property for an existing interaction link. An *interaction property revocation operator* deletes a property from the dynamic description of an interaction link.

Interaction property addition operator
Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is an interaction property addition operator iff it satisfies:

1. dynamics_in_focus(O, ∅)
2. dynamics_in_focus(O', DPf')
   DPf'= {δ∈DP'| ∃e∈IL' has_dynamic_property(e, δ)}

Interaction property revocation operator
Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is an interaction property revocation operator iff it satisfies:

1. dynamics_in_focus(O, DPf)
   DPf= {δ∈DP| ∃e∈IL has_dynamic_property(e, δ)}
2. dymanics_in_focus(O', ∅)

An interlevel link creates a relation between a composite role and its subroles. It allows information that is generated outside the role, to be passed into the role through its input interface or it allows information, generated within a role to be transmitted outside through the role output interface. Normally, in hierarchical (mechanical) organizations decisions made at a managerial level are transferred to an

operational level, e.g, to a certain department. Within the department this information is obtained by a certain role(s). For identifying, which roles obtain this information interlevel links are used. In the conference management example, the Conference Chair may have the possibility to send inquiries to Program Committee Members. This can be achieved by introduction of an interlevel link between composite role Paper Selection (with which role Conference Chair has a direct connection by an interaction link) and its subrole Program Committee Member (see Figure 4).

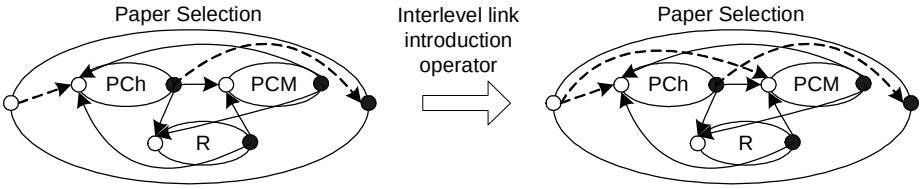


Fig. 4. Application of the interlevel link introduction operator for adding an interlevel link between Paper Selection role and Program Committee Member role (PCM)

Interlevel link introduction operator
Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is an interlevel link introduction operator iff it satisfies:
1. $\delta \notin IL$, $\delta \in IL'$ such that is_interlevel_link_in(δ, Γ)
2. structure_in_focus(O, ∅, ∅, ∅, ∅, ∅, ∅, ∅)
3. structure_in_focus(O', ∅, ∅, ∅, {δ}, ∅, Mf', ∅)
   Mf'= {m∈ M'| has_onto_mapping(δ, m)}

An *interlevel link retraction operator* is used for breaking off interaction between some composite role and one of its subroles. This operation removes an interlevel link from the design object description for an organization. If the Conference Chair does not need to communicate with Program Committee Members any more, the interlevel link between these two roles can be retracted.

Interlevel link retraction operator
Let op(O, O', δ) be an operator that changes O into O' with a focus on δ. Then op is an interlevel link retraction operator iff it satisfies:
1. $\delta \notin IL'$, $\delta \in IL$ such that is_interlevel_link_in(δ, Γ)
2. structure_in_focus(O, ∅, ∅, ∅, {δ}, ∅, Mf)
   Mf= {m∈ M| has_onto_mapping(δ, m)}
3. structure_in_focus(O', ∅, ∅, ∅, ∅, ∅, ∅)

3.3 OPERATORS FOR GROUPS

The classes of primitive operators for creating and modifying groups in a design object description for an organization are shown in Table 3.

Often an organization designer can easily list a number of roles needed in an organization. However, it is not always clear, which roles are related to each other; which roles would most often interact with each other, and so on. Once identified, the organization designer can group roles into sets.

Table 3. Operator classes for creating and modifying groups

CLASS	DESCRIPTION
Grouping	Combines roles into groups
Degrouping	Moves roles outside of a group and deletes the group
Group-to-Role Transformation	Transforms groups into roles
Role-to-Group Transformation	Transforms roles into groups

In the literature on organizational design (Minzberg 1993) different principles of grouping are described. For example, role grouping can be performed based on (1) similarities in role functional descriptions; (2) role participation in the same technological process; (3) identity or similarity of role technical specialties; (4) role orientation on the same market or customer groups. Often roles belonging to the same group interact with each other intensively. However, in the proposed organizational model in contrast to roles, groups do not have interfaces. It means that every role within a group is allowed to interact with roles outside the group by means of direct interaction links. For example, in the conference organization the Program Chair and the Program Committee Members can be joined in one Program Committee group that will be responsible for making final decisions concerning paper acceptance. This can be accomplished by applying the grouping operator (see Figure 5).

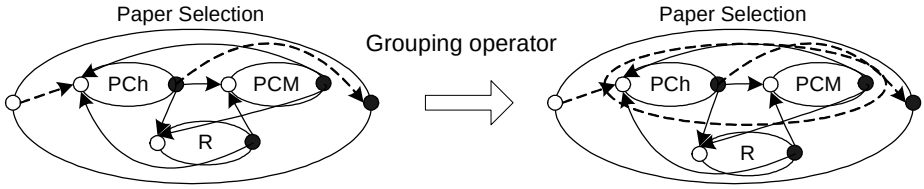


Fig. 5. Application of the grouping operator to create the Program Committee group that consists of roles Program Chair (PCh) and Program Committee Member (PCM) for making final decisions concerning paper acceptance

Grouping operator

Let op(O, Rg, O', Gn) be an operator that changes O into O' wrt. Gn∈G', Rg⊆R. Then op is a grouping operator that creates a new group Gn from the subset of roles Rg iff it satisfies:

Structural aspect:

1. ∀a∈Rg: member_of_in(a, Gn, Γ').
2. structure_in_focus(O, ∅, ∅, ∅, ∅, ∅, ∅, ∅)
3. structure_in_focus(O', ∅, {Gn}, ∅, ∅, ∅, ∅, ∅)

Dynamic aspect:

1. dynamics_in_focus(O, ∅)
2. dynamics_in_focus(O', DPf')
   DPf'={dp∈DP'| has_dynamic_property(Gn, dp) }.
3. Er={e∈IL| ∃r1∈Rg ∃r2∈Rg connects_to(e, r1, r2, Γ)}
   DPr={dp∈DP|∃r∈Rg has_dynamic_property(r, dp) ∨ ∃e∈Er has_dynamic_property(e, dp)}
   DPg={dp∈DP'| has_dynamic_property(Gn, dp)}
4. DPg⊆DCL(DPr), where DCL(DPr) is the deductive closure of DPr

A natural dual to the role grouping is role degrouping. This operator takes a group of roles and moves the roles to outside of the group. Role Degrouping transforms a group into a set of roles.

Degrouping operator

Let op(O, Gd, O', Rdg) be an operator that changes O into O' wrt. Gd∈G, and Rdg⊆R'. Then op is a degrouping operator iff it satisfies:

Structural aspect:

1. Rdg={r∈R| member_of_in(r, Gd, Γ)}
2. Gd∉G'
3. structure_in_focus(O, ∅, {Gd}, ∅, ∅, ∅, ∅)
4. structure_in_focus(O', ∅, ∅, ∅, ∅, ∅, ∅)

Dynamic aspect:

1. dynamics_in_focus(O, DPf)
   DPf={dp∈DP| has_dynamic_property(Gd, dp) }.
2. dynamics_in_focus(O', ∅)

A group can be transformed into a role, a more coherent, integrated and formal organizational unit with proper interfaces (e.g., a department of an organization). For a group to act as a role, it should have well-defined (formalized) input and output interfaces. A Group-To-Role operator takes a group and adds these interfaces. In an organic organization with loosely defined frequently changing structure this would correspond to the formalization of one of the organizational units, i.e., providing a formal (permanent) structural description with the subsequent specifying formal functional procedures. For example, in the conference organization setting Program Committee group from the Paper Selection role can be further transformed into Program Committee role, a formal organizational unit with certain characteristics and functions (e.g., final decision making for the paper acceptance). Such transformation can be achieved by means of Group-to-Role operator (see Figure 6). The next logical step would be to limit interactions of subroles of Program Committee role only to those that exist within Program Committee role, and replace all interactions with the roles outside of Program Committee role by corresponding interactions between outer roles and Program Committee role. This can be done by applying interaction and interlevel link addition and retraction operators. In this case reviewers should follow a formal procedure for interactions with Program Committee role and cannot directly address any arbitrary Program Committee member.

Group-to-Role operator

Let op(O, g, O', r) be an operator that transforms group g∈G in O into role r∈R' in O'. Then op is a group-to-role operator iff it satisfies:

Structural aspect:

1. r∉R, g∉G'.
2. ∀a∈R: member_of_in(a,g, Γ) ⇒ subrole_of_in(a, r, Γ').
3. structure_in_focus(O, ∅, {g}, ∅, ∅, ∅, ∅, ∅)
4. structure_in_focus(O', {r}, ∅, ∅, ∅, ONTf', ∅, ∅)
   ONTf'={o∈ONT'| has_internal_ontology(r, o) ∨ has_input_ontology(r, o) ∨ has_output_ontology(r, o)}

Dynamic aspect:

1. dynamics_in_focus(O, DPf)
   DPf={dp∈DP| has_dynamic_property(g, dp)}.

2. dynamics_in_focus(O', DPf')
   DPf'={dp∈DP'| has_dynamic_property(r, dp)}.
3. DP(g) ⇒ DP(r)

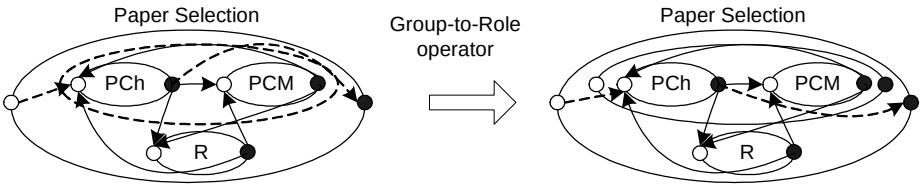


Fig. 6. Application of the Group-to-Role operator to transform Program Committee group into Program Committee role

A role may consist of several other roles that are not exposed to the rest of the world. When a role is converted to a group, it exposes the input and output interfaces of the roles inside it. Transforming a role into a group results in the subroles now residing on the level of the prior composite role. For example, during the reorganization some formal organization units (e.g., groups, sections, and departments) have been eliminated, whereas the roles that constituted these units and relations between them were kept, thus, creating a basis for new organizational formations.

Role-to-Group operator
Let op(O, r, O', Gr) be an operator that changes O into O', with respect to r∈R, and Gr∈G'. Then op is a role-to-group operator that transforms role r into group Gr iff it satisfies:
Structural aspect:
1. Gr∉G, r∉R'.
2. ∀a∈R: subrole_of_in(a,r, Γ) ⇒ member_of_in(a, Gr, Γ').
3. structure_in_focus(O, {r}, ∅, ∅, ∅, ONTf, ∅)
   ONTf={o∈ONT| has_internal_ontology(r, o) OR has_input_ontology(r, o) OR has_output_ontology(r, o)}
4. structure_in_focus(O', ∅, {Gr}, ∅, ∅, ∅, ∅)

Dynamic aspect:
1. dynamics_in_focus(O, DPf)
   DPf={dp∈DP| has_dynamic_property(r, dp)}.
2. dynamics_in_focus(O', DPf')
   DPf'={dp∈DP'| has_dynamic_property(g, dp)}.

4 Composing operators

The described above primitive operators reflect major principles of organizational design. In practice next to the primitive operators more complex operators are used. Complex operators are represented as a combination of a certain number of primitive operators; some of them are given in Table 4.

Table 4. Sample complex operators for creating and manipulating organizations

NAME	PATTERN FOR	DESCRIPTION
Interaction Level Ascent	Interaction link deletion*. Role interaction dynamic property addition*. Interlevel link addition*. Interaction link addition*.	Represents interaction between roles at a higher aggregation level
Role refinement	Role Retraction. Interlevel link deletion*. Interaction link deletion*. Interaction dynamic property addition*. Interlevel link addition*. Interaction link introduction*. Role dynamic property addition*. Role introduction*	Divides a role into several roles such that the role properties of the first role are distributed over the newer roles
Role join	Role Retraction*. Interlevel link deletion*. Interaction link deletion*. Interaction dynamic property addition*. Interlevel link addition*. Interaction link introduction*. Role dynamic property addition*. Role introduction	Joins several roles into a single role
Adding aggregation levels	Interaction Level Ascent. G-t-R. Role grouping. Role refinement*	Aggregates existing roles of the organization in more complex roles

The symbol * denotes that an operator can be applied zero, one or multiple times.

Sometimes an effect produced by application of some composite operator to a design object description for an organization can be achieved by different combinations of primitive operators.

Consider the Role Refinement operator as an example. This operator divides a role into several roles such that the role properties of the first role are distributed over the newer roles. In organizational design role refinement corresponds to the fine-tuned specialization and division of labor for increasing efficiency. It is usually recommended to divide the work so that the portions be differentiated rather than similar, and that each role is responsible for a small portion of the overall task. According to Adam Smith, division of labor is limited by the extent of the market; other general principles of labor division can be found in (Kilbridge and Wester 1966).

Let us illustrate the application of Role Refinement operator in the context of the conference organizing example. In Figure 7 the design object description for an organization is represented at the first aggregation level. Consider the situation when the decision is made to divide the tasks of Organizing Committee (OC) between the Local Organizing Committee (LOC), which is hence responsible for negotiations with publishers for printing proceedings and arranging the conference venue, and the General Organizing Committee (GOC), which is designated for solving financial and other questions. Thus, role OC is refined into two newer roles LOC and GOC. These roles are able to interact with each other and with role Chair.

Alternatively, every composite operator can be considered as an aggregated one-step operator. Such descriptions define formal conditions for a design object description for an organization before and after the application of a complex operator; therefore, they can serve for the purposes of checking integrity and consistency of a design object description.

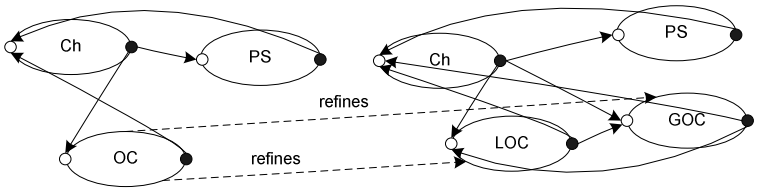


Fig. 7. Example of Role refinement operator application, in which the Organizing Committee role (OC) is refined into the Local Organizing Committee (LOC) and General Organizing Committee roles (GOC)

An example of such a representation for the refinement operator is given below.

Refinement operator (integrity definition)

Let op(O, r, O', Rref) be an operator that refines role $r \in R$ in O into a set of roles $Rref \subseteq R'$ in O'. Then op is a refinement operator iff it satisfies:

Structural aspect:

1. $r \in R$, $r \notin R'$, $Rref \cap R = \varnothing$
2. structure_in_focus(O, {r}, ∅, ILf, ∅, ONTf, Mf, ∅)
   ILf={e∈IL| ∃r'∈R connects_to(e, r', r, Γ) OR ∃r''∈R connects_to(e, r, r'', Γ)},
   Mf= {m∈M| ∃e∈ILf has_onto_mapping(e, m)}
   ONTf={o∈ONT| has_ontology(r, o)}
3. structure_in_focus(O', Rref, ∅, ILf', ∅, ONTf', Mf', ∅)
   ILf'={e∈IL'| ∃r1∈Rref ∃r2∈Rref connects_to(e, r1, r2, Γ') OR ∃r1'∈Rref ∃r2'∈R', r2'∉Rref connects_to(e, r1', r2', Γ') OR ∃r1''∈Rref ∃r2''∈R', r2''∉Rref connects_to(e, r2'', r1'', Γ')}.
   ONTf'={o∈ONT'| ∃r1∈Rref has_ontology(r1, o) }.
4. ∀e∈IL, ∀b∈R, b∈R', b∉Rref connects_to(e, r, b, Γ) ⇒ ∃e'∈IL', ∃r' ∈ Rref connects_to(e', r', b, Γ') and
   ∀e∈IL, ∀a∈R, a∈R', a∉Rref connects_to(e, a, r, Γ) ⇒ ∃e'∈IL', ∃r'∈Rref connects_to(e', a, r', Γ').
5. ∀e' ∈ IL', ∀r'∈Rref ∀b∈R' and b∉Rref connects_to(e, r', b, Γ') ⇒ ∃e∈IL, connects_to(e, r, b, Γ) and
   ∀e' ∈ IL', ∀r'∈Rref ∀a∈R' and a∉Rref connects_to(e, a, r', Γ') ⇒ ∃e∈IL, connects_to(e, a, r, Γ).

Dynamic aspect:

1. dynamics_in_focus(O, DPf)
   DPf={dp∈DP| has_dynamic_property(r, dp) ∨ ∃e∈ILf has_dynamic_property(e, dp)}.
2. dynamics_in_focus(O', DPf')
   DPf'={dp∈DP'| ∃r1∈Rref has_dynamic_property(r1, dp) OR ∃e'∈ ILf' has_dynamic_property(e', dp)}.
3. ONTp={o∈ONT| ∃dp∈DPf uses_ont(dp, o) AND o∉ONTf}
   ∀φ∈DYNPROPEXPR, such as uses_only_ont(φ, $\bigcup_{o \in ONTp}$ o) [DPf ⇒ φ] ⇒[DPf' ⇒ φ]

A natural dual to the role refinement is role joining. This operator takes several roles and joins them into a single role. Consider again the organization arranging a conference. If over time the differences between the tasks of the Program Committee Member and Reviewer roles disappear, then the roles Program Committee Member and Reviewer can be joined in one role.

Let us consider one more often used complex operator *Adding Aggregation Levels*. When certain roles have been joined in one group, this operator allows representing

this group as an integral structural unit of an organization at the more abstract aggregation level. This operator has a counterpart in organizational design studies called *departmentalization*. Based on the departmentalization principles (cf. Galbraith 1978) an organization is partitioned into structural units (called departments) with certain areas of responsibilities, a functional orientation, and a local authority power.

In the conference organization Adding Aggregation Levels operator can be applied for representing the Program Committee as an integral role that consists of the Program Chair and the Program Committee Member roles within Paper Selection role. Such choice, for example, can be motivated by introducing a general formal procedure for paper acceptance. Hence, the Program Committee role is empowered (has a corresponding dynamic property) to make final decisions concerning paper selection. Adding Aggregation Levels operator for this example can be considered as three-step process (see Figure 8 for the representation of the organization model (role Paper Selection) at the second aggregation level).

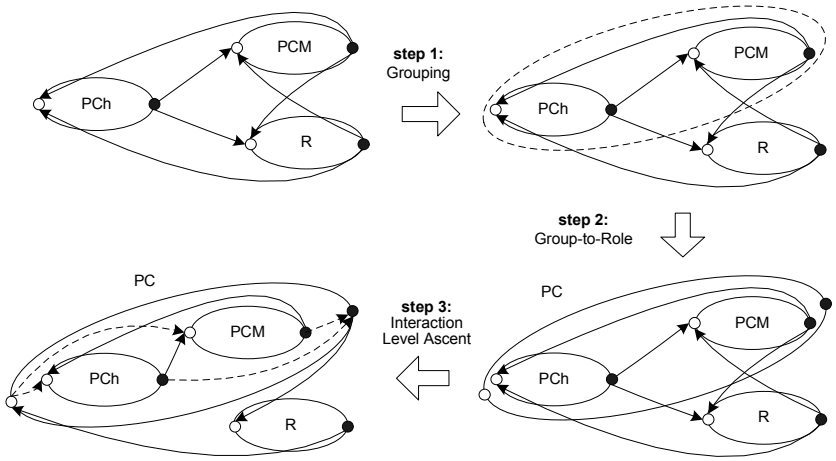


Fig. 8. Example of Adding Aggregation Levels operator application, in which the roles Program Chair (PCh) and Program Committee Member (PCM) are grouped together and transformed into the Paper Selection (PC) role

First, roles Program Chair (PCh) and Program Committee Member (PCM) are joined into one group by application of Grouping operator. After that, at step 2 by means of the Group-to-Role operator the created group is transformed into role Program Committee by adding interaction interfaces. Finally, as the last step using Interaction Level Ascent operator interaction links between roles PC and Reviewer (R) are created, as well as interlevel links within role PC.

5 A Prototype Tool to Support the Design of Organizations

The formal representations of the organization entities and the design operators described in this paper provide a solid basis for the development of a software

environment supporting interactive organization design processes. The proposed formalism accurately distinguishes different types of organization entities with their objects, relations and constraints, which can be naturally represented as classes with members and methods in object-oriented programming (OOP) languages. Furthermore, the identified relationships among organization entities may be fully captured by the fundamental OOP mechanisms (e.g., inheritance, interfaces and inner classes). The design operators can be programmed as transformation functions with explicitly defined arguments, conditions and effects of their application. Moreover, most of the introduced formal concepts are based on the notions from organization theories, which will facilitate use of a tool by organization modelers.

For the purpose of illustration and evaluation a prototype tool was implemented. This tool supports organizational design and allows investigating its dynamics. The application of the design prototype is demonstrated on the example of role refinement as described in the previous Section. The dynamics of the design process is described in Table 5, which is graphically illustrated by a partial trace taken from the tool in Figure 9[1].

Table 5. Dynamics of the design process for the role refinement

ACTIONS OF THE DESIGNER	STATES OF THE TOOL
Chooses to address the role Organizing Committee (OC)	Proposes potentially applicable operators for role OC
Chooses the role refinement operator	According to the specification of the role refinement operator, initiates execution of role introduction operator and requests the designer to specify role names
Specifies GOC (General Organizing Committee) and LOC (Local Organizing Committee) names of the roles, into which role OC is refined	Requests to specify the elements of the ontologies for the newly created roles
Specifies the elements of the ontologies for roles LOC and GOC	Initiates execution of the role dynamic property addition operator. Requests to specify dynamic properties for LOC and GOC roles
(optional) Specifies dynamic properties for the roles	Initiates execution of the interaction link introduction operator. Requests to specify interaction links between roles Chair (Ch), LOC and GOC
Specifies, which interaction links are needed between the roles	Initiates execution of the interaction dynamic property addition operator. Requests to specify dynamic properties for the introduced interaction links
(optional) Specifies dynamic properties for the interaction links	Initiates execution of the interaction link deletion operator, which removes all interaction links connected with role OC. Then, initiates execution of the role retraction operator, which removes role OC from the design object description

In the design process, first, a designer chooses a part of the design object description, on which she intends to put her attention (in the considered example it is the role Organizing Committee). Next, the software proposes to the designer a number of operators, which are potentially applicable to the chosen part of the design object description. The designer chooses one of them, for the example, the role

[1] The complete screen print of a trace illustrating dynamics of the design process for role refinement is given in Appendix A.

refinement operator. Refinement is a composite operator that consists of an ordered sequence of primitive operators. Usually, most of the primitive operators constituting composite ones are imperative (e.g., Role Introduction for Refinement); yet application of some of them may be postponed to the future (e.g., Role dynamic property addition for Refinement) or skipped (e.g., Interlevel link deletion for Refinement). Further, the tool demands specifying roles, into which role OC has to be refined. The designer specifies role names (for this example, Local Organizing Committee (LOC) and General Organizing Committee (GOC)) and their ontologies.

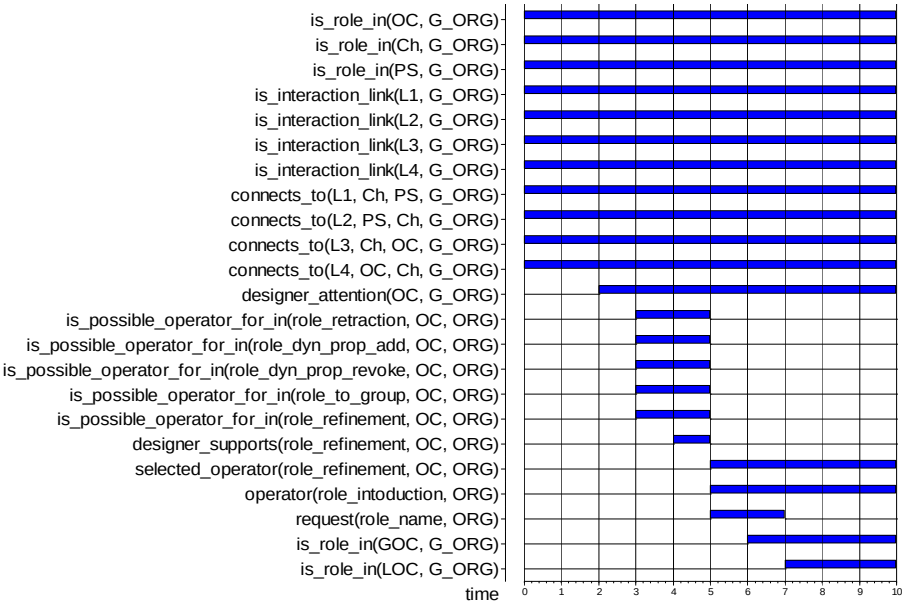


Fig. 9. Screen print of a trace illustrating dynamics of the design process for the role refinement

After that the software tool requests the designer to specify dynamic properties for the created roles. The designer may postpone this operation to a future time point. Thereafter, the tool proposes to add interaction links between roles LOC, GOC and role Chair (Ch), with which the original role OC was connected. After that dynamic properties for the introduced interaction links may be added. As the last step role OC and interaction links connecting it with role Ch, as well as OC role and interaction links dynamic properties are automatically removed from the design object description.

6 Discussion

This paper introduces a representation format and a variety of operators for the design of organizations specified in this representation format. The described operators have several important characteristics. First, they can be combined into composite

operators that can serve as patterns for larger design steps in certain design cases. Second, the identified set of operators is independent of any organization theory or sociological methodology: they can be used for formalizing design principles from different theories. Third, a designer has freedom to choose any sequence of operators for creating designs of organizations of most types (e.g., functional and organic). An example of functional organizational design was discussed in this paper. When designing adaptive organic organizations, dedicated structural elements (e.g., the organization change management role) and dynamic descriptions (e.g., properties that describe the adaptation process) are specified. The operators offer both top-down refinements, as well as bottom-up grouping options. Finally, as has been shown the developed tool provides interactive support in designing organizations. In the future a graphical interface for representing design objects in the tool will be developed.

To a certain extent organizations can be considered as compositional systems (Wijngaards, 1999). However, models and design methods for such systems do not allow representing many organization domain-specific concepts and operators (e.g., a group, a Group-to-Role operator) and, therefore, cannot capture many important organization phenomena.

In the area of component-based software engineering a number of design patterns for building software components (e.g., refinement, chaining, disjoint composition) have been introduced (He, Li, and Liu 2005). These patterns specify general-purpose manipulations with programming constructs (e.g., interface and private methods of components); while in organizational design literature organization transformations are described using domain-specific concepts. The formal representation format proposed in this paper bridges this gap and facilitates the abstraction of organization domain into general-purpose programming design patterns.

Formal specification of design processes enables verification of structural and dynamic consistency of a design object description for an organization. The verification of structural consistency is based on the consistency definitions for operators, such as one given in Section 4 for the role refinement operator. For verifying dynamic consistency (e.g., checking relations between dynamic properties defined at different aggregation levels of a model representation) model checking techniques (McMillan 1993; Sharpanskykh and Treur 2006) may be used, which will be further investigated in the future. Furthermore, verification mechanisms based on certain requirements on organizational functioning and performance (e.g., using organization performance indicators) represent a subject of our future research.

Another way to evaluate an organizational model is by performing simulations. For this purpose, agents with different types of attitudes and internal architectures may be allocated to roles within an organization model on certain conditions. After that, by considering different types and sequences of environmental influences provided within certain simulation scenarios, traces (i.e., temporal sequences of events in the environment and within the organization) corresponding to the execution of scenarios can be generated. These traces may be further used for analysis of the organizational model, more specifically, for evaluating different global properties of the organizational model (e.g., robustness, stability, efficiency, and effectiveness).

In conclusion, this paper introduced a representation format and a set of formally represented design operators dedicated to the design of organizations of most types. Although the choice of operators is motivated by different theories and guidelines

from the area of organizational design, the application of the proposed operators is not restricted to any theories from social studies. The formalization of the operators provides a solid basis for the development of a software tool supporting interactive organization design processes. A prototype implementation for such a tool is demonstrated by an example in this paper.

References

Bertino, E, Zarri, GP, Catania, B: 2001, *Intelligent Database Systems*. Addison-Wesley Professional.

Blau, PM and Schoenherr, RA: 1971, *The structure of organizations*, Basic Books Inc., New York London.

Broek, E.; Jonker, C.; Sharpanskykh, A.; Treur, J. and Yolum, P. 2006. Formal Modeling and Analysis of Organizations. In O. Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowsk, J. Sichman and J. Vazquez Salceda (eds.), *Coordination, Organization, Institutions and Norms in Agent Systems I*, LNAI 3913, Springer, 18-34

Child, J: 1973, *Organization: A Choice for Man, in* J Child (ed), *Man and Organization*, Halsted Press, London, pp. 234-570.

Chomsky, N: 1965, *Aspects of the Theory of Syntax*, The MIT Press.

Duncan, RB: 1979, What Is the Right Organization Structure? *Organizational Dynamics*, Winter, pp.59-79.

Galbraith, JR: 1978: *Organization design*, Addison-Wesley Publishing Company, London Amsterdam Sydney.

Habel, A and Hoffmann, B: 2004, Parallel Independence in Hierarchical Graph Transformation, *in* Proceedings of *International Conference on Graph Transformation*, LNCS, Volume 3256, Springer-Verlag, Heidelberg, pp. 178-193.

He, J, Li, X, and Liu, Z: 2005, Component-Based Software Engineering, in D V Hung, M Wirsing (eds), *Theoretical Aspects of Computing*, LNCS 3722, Springer, pp. 70-95.

Huth, M and Ryan, MD: 2004, *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press.

Jonker, CM., Treur J: 2003, A temporal-interactivist perspective on the dynamics of mental states, *Cognitive Systems Research Journal*, **4**(3): 137-155.

Kilbridge, M and Wester, L: 1966, An economic model for the devision of labor, *Management Science*, February: 255-269.

Lorsch, JW and Lawrence, PR: 1970, *Organization design*, Richard D. Irwin Inc., USA.

Manzano, M.: 1996, *Extensions of First Order Logic*, Cambridge University Press.

McMillan, K: 1993, *Symbolic Model Checking*, Kluwer Academic Publishers.

Mintzberg, H: 1993, *Structure in Fives: Designing Effective organizations*, Prentice-Hall, NJ.

Pfeffer, J: 1978, *Organizational design*, AHM Publishing Corp., Illinois, USA.

Rozenberg, G (ed): 1997, *Handbook of Graph Grammars and Computing by Graph Transformation*, Volume 1: Foundations, World Scientific.

Schmidt, LC and Cagan, J: 1995, Recursive Annealing: A Computational Model for Machine Design, *Research in Engineering Design*, **7**: 102-125.

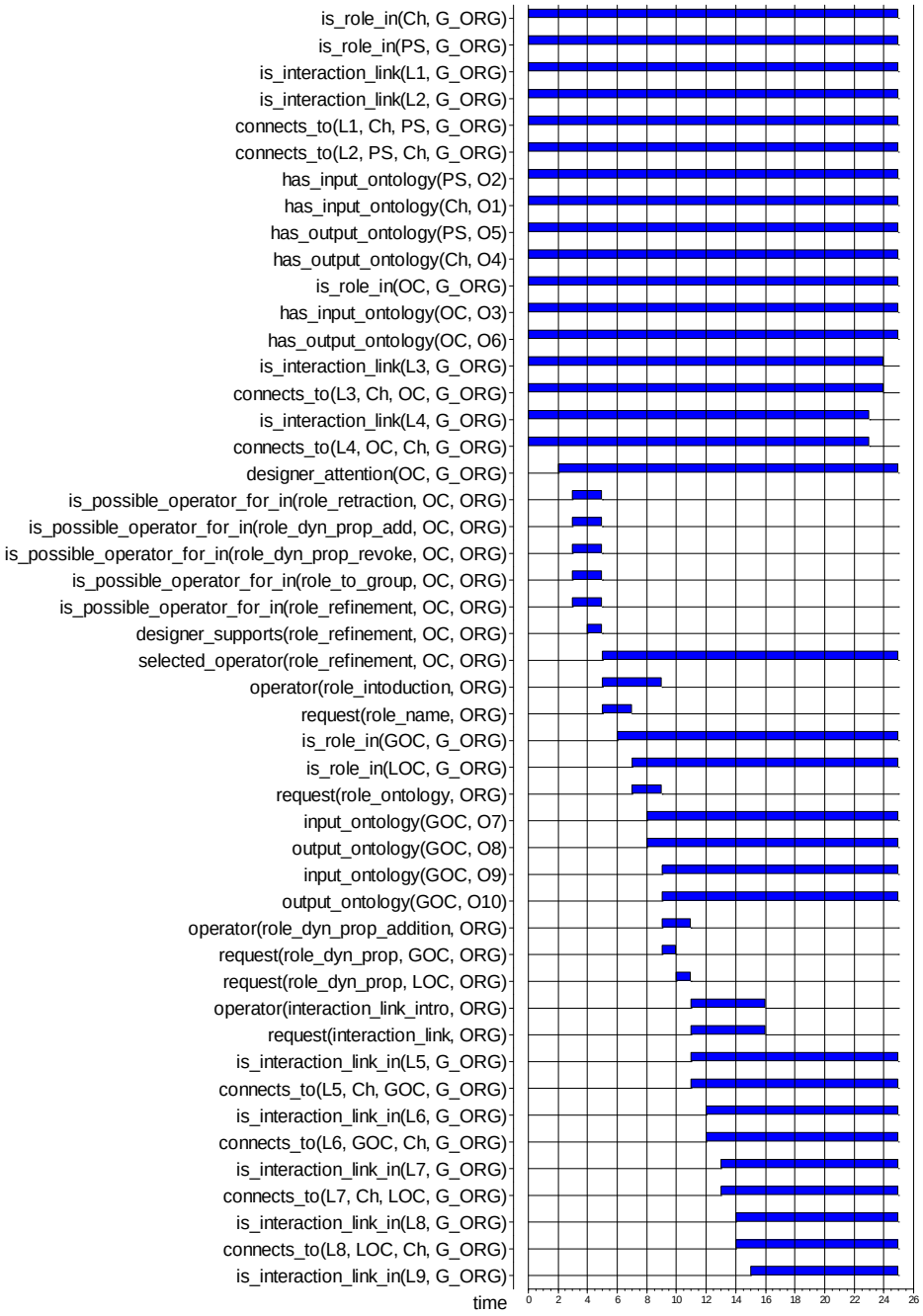
Scott, WR: 1998, *Organizations: rational, natural and open systems*, Prentice Hall, USA.

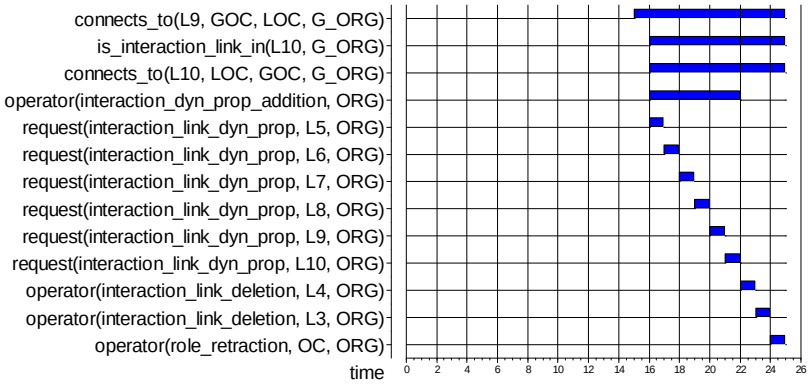
Sharpanskykh, A., and Treur, J.: 2006, Verifying Interlevel Relations within Multi-Agent Systems, *in* Proceedings of the *17th European Conference on Artificial Intelligence, ECAI'06*, IOS Press

Stiny, G: 1991, The Algebras of Design, *Research in Engineering Design*, **2**: 171-181.

Wijngaards, N: 1999, *Re-design of Compositional Systems*, PhD Thesis, SIKS dissertation Series, 99-6, Vrije Universiteit Amsterdam.

Appendix A. Screen print of a trace illustrating dynamics of the design process for role refinement




connects_to(L9, GOC, LOC, G_ORG)
is_interaction_link_in(L10, G_ORG)
connects_to(L10, LOC, GOC, G_ORG)
operator(interaction_dyn_prop_addition, ORG)
request(interaction_link_dyn_prop, L5, ORG)
request(interaction_link_dyn_prop, L6, ORG)
request(interaction_link_dyn_prop, L7, ORG)
request(interaction_link_dyn_prop, L8, ORG)
request(interaction_link_dyn_prop, L9, ORG)
request(interaction_link_dyn_prop, L10, ORG)
operator(interaction_link_deletion, L4, ORG)
operator(interaction_link_deletion, L3, ORG)
operator(role_retraction, OC, ORG)
time
0
2
4
6
8
10
12
14
16
18
20
22
24
26

Part V

Case Study

An Air Traffic Control Organization (ATCO) ensures a safe and efficient flow of aircrafts both at airports and in the air. Nowadays, an ATCO represents a complex organization that involves many parties with diverse goals performing for a wide range of tasks. Among the ATCO's participants are airports, air navigation service providers (ANSP), airlines, regulators, and the government. Constantly increasing air traffic correlates with the growth of intensity and the complexity of interactions among all the parties of an ATCO. Due the structural and behavioral complexity of ATCOs, mistakes, inconsistencies and performance bottlenecks are not rare in such organizations. Some of these organizational flaws may result into performance issues, whereas others can seriously affect safety, causing incidents and even accidents. Therefore, the possibility to perform detailed and reliable automated analysis aiming at detecting safety hazards and performance issues in the structures and dynamics of ATCOs is of primary importance in the air traffic management domain.

To enable such analysis possibilities, in this part the design steps identified in Part IV will be consequently applied to create a specification for an air traffic control organization. This organization combines features of mechanistic and organic organizations. All the required aspects of the organization will be specified using the concepts and relations of the proposed modeling methods. The analysis of the constructed specifications will be performed by applying the general and specific for particular views analysis techniques introduced in this thesis. Furthermore, the consequences of different types of agent behavior that diverges from the formal organization will be investigated by performing simulations.

Chapter 1

Modeling and Analysis of Organizations from the Air Traffic Management Domain [1]

1 Introduction

In this case study we shall focus in particular on three general, related to each other tasks performed by the ATCO:

(1) the development and evaluation of a new operation;
(2) the movement of an aircraft on the ground;
(3) the incident reporting and investigation.

In the following these tasks will be briefly described. Initially, all operations related to the movement of an aircraft on the ground should be developed and evaluated (e.g., the introduction of runways and taxiways). The development of a new operation is often performed by a specially created design team. This team comprises representatives of the operation design units of the airport at which the developed operation will be introduced, of the ANSP, who will be involved into the operation execution and of an airline (usually the largest one that most often uses the airport). The control over the team functioning is exercised by the team manager. Furthermore, during the operation development process the intermediate design concepts of the operation are provided to the operation assessment unit of the ANSP for the evaluation. In such a way, mistakes and inconsistencies in the operation design can be identified at early stages of the operation design. Also, the experience shows that such

[1] We would like to acknowledge the National Aerospace Laboratory for the cooperation and the European Organization for the Safety of Air Navigation for funding the project CARE INO III, in the frames of which this case study has been performed.

a way of work organization results into the operation concepts of a higher quality and into the decrease of the operation development time. When the concept is developed and evaluated, it is provided to the operation management team of the ANSP and to the executive management of the ANSP for the final review.

Based on the evaluated operations related to the movement of an aircraft on the ground the descriptions of tasks are specified, in particular, the taxiing of an aircraft to the designated runway and the subsequent take off from this runway. During the taxiing process an aircraft moves from one sector of the airport to another, until it reaches the runway designated for take off. The monitoring and the control over the traffic in a sector or on a runway are performed by a dedicated controller. During the taxiing the control over an aircraft is handed over from one controller to another, depending on the physical position of the aircraft. Before crossing an active runway the crew of an aircraft should request the controller responsible for the runway for clearance. Only when the clearance is provided, the aircraft is allowed to cross. The same holds for the take off operation.

In case an incident/accident occurs during the taxiing or take off execution, this should be reported and further investigated. There are several incident reporting paths exist in the ATCO, which will be elaborated in this case study later.

In Section 2 first the methodological steps for modeling and analysis of the ATCO structure and processes for the identified tasks is described. Then, the constructed organizational specification for the ATCO is presented in Section 3. Section 4 discusses the analysis results of the constructed organizational specification using both correctness verification and simulation techniques. Finally, the conclusions are presented in Section 5.

2 Methodology

To perform analysis of the organizational structure and behavior the following types of specifications should be developed:

(1) the specification of the formal organization;

(2) the specifications of agents of different types (i.e., their characteristics and behavior) and the principles of their allocation to the roles of the organization.

The required specifications for existing organizations may be created using different sequences of design steps. The chosen sequence given below is one of the alternatives:

Step 1. The identification of the organizational roles

Identify organizational roles, both simple and composite ones and establish subrole-relations between them.

Step 2. The specification of the interactions between the roles

2a) Identify interaction relations between the roles.

2b) Identify additional roles that enable interactions between composite roles.

2c) Identify input and output ontologies for each role.

Step 3. The identification of the requirements for the roles
Identify the requirements for each role at the lowest aggregation level.

Step 4. The identification of the organizational performance indicators and goals
Identify organizational goals and performance indicators, relations between them and organizational roles.

Step 5. The specification of the resources
Identify organizational resource types and resources, and provide characteristics for them.

Step 6. The identification of the organizational tasks, the relations between the tasks, and relations between the tasks, the resources and the goals
6a) Identify the task hierarchy that comprise tasks related by AND- and OR-relations.
6b) For each task identify its characteristics: name, minimum and maximum duration.
6c) Relate each task to organizational resource(s).
6d) Relate each task to organizational goal(s).

Step 7. The specification of the authority relations
Identify authority relations (i.e., formal power relations): superior-subordinate relations on roles with respect to tasks, responsibility relations, control for resources, authorization relations.

Step 8. The specification of the flows of control
Identify flows of control (workflows).

Step 9. The specification of the characteristics and behavior of agents, and the agent allocation principles.

Step 10. The identification of the generic and domain-specific organizational constraints
Identify general and domain-specific rules on the concepts and relations from the particular organizational perspectives and across different perspectives.

Note that for the execution of the steps 1-8 and 10 (formal) organizational documents are required (e.g., organizational charts, job descriptions, procedures, regulations). By execution of these steps the specification of the formal organization is created. On the contrary, the step 9 aims at the identification of possible (realistic) variations and deviations of/from the predefined organizational structure and behavior that may be attributed to agents allocated to the organizational roles.

For the analysis of the developed specifications a number of general and dedicated techniques of the framework will be applied. First, the correctness of the specifications will be checked using the verification techniques of the corresponding views. Then, the behavior of the organization in particular configurations and in particular environmental settings will be investigated by simulation.

3 The organizational specification of the ATCO

The constructed specifications will be described along the methodological steps identified in Section 2.

Step 1. The identification of the organizational roles
In the considered organization roles can be represented at three aggregation levels. For example, at the aggregation level 1 the Airport is considered as one composite role. The subroles of the Airport (the Airport Operation Design Unit and the Airport Management) are described at the aggregation level 2, and so forth. A special role type is the environment (env). In Table 1 all the generic roles of the considered organization with their subroles are listed. Note that based on the introduced generic roles role instances may be defined for particular applications. In particular, in this case study two instances of the Runway Controller role and two instances of the Ground Controller role are introduced. Furthermore, depending on the analysis type different number of instances of the role Airline will be considered.

In the following the abbreviations are used for the role names:

ANSP	Air Navigation Service Provider
ATC	Air Traffic Control
EMATC	Executive Management ATC
OAU	Operation Assessment Unit
OMT	Operation Management Team
ODU	Operation Design Unit
SIU	Safety Investigation Unit
TCU	Tower Control Unit
CST	Controllers Supervision Team
AODU	Airport Operation Design Unit
SMU	Safety Management Unit
NODT	New Operation Design Team

Table 1. Identified organizational roles at three aggregation levels.

Level 1	Level 2	Level 3
Air Navigation Service Provider	ATC Executive Management	-
	Operation Management Team	Team Member (CSU)
		Team Member (ODU)
		Team Member (Systems)
		Team Member (Maintenance)
		OMT Leader
	Operation Assessment Unit	Operation Analyst
		Head OAU
	Operation Design Unit	Operation designer
		Head of ODU
	Safety Investigation Unit	Safety Investigator
		Head of SIU
	Tower Control Unit	Ground Controller
		Runway Controller

		Tower Controllers Supervisor
		Env
	Controllers Supervision Team	Controllers Supervisor
		Head of Controllers
Airport	Airport Operation Design Unit	Airport Operation Designer
		Head AODU
	Airport Management	-
Airline	Airline Management	-
	Safety Management Unit	-
	Crew	Pilot in command
		Second Pilot
		Env
Regulator	-	-
New Operation Design Team	Design Team Manager	-
	ANSP design representative	-
	Airport design representative	-
	Airline representative	-
Env	-	-

Step 2. The specification of the interactions between the roles
Interaction relations between roles can be depicted at different aggregation levels. In particular, the interaction relations between the roles at the aggregation level 1 of the ATCO are depicted in Figure 1.

To enable the interaction between the controllers of the ANSP and the pilots of the Airline the role Controller-Crew Interaction is introduced. This role consists of the subroles the Aircraft's Controller, who performs the constant monitoring and control over the aircraft and the Crew Representative role that interacts with the controllers on behalf of the aircraft's crew, when the contact is required. This interaction role is depicted at the aggregation level 2 in Figure 2.

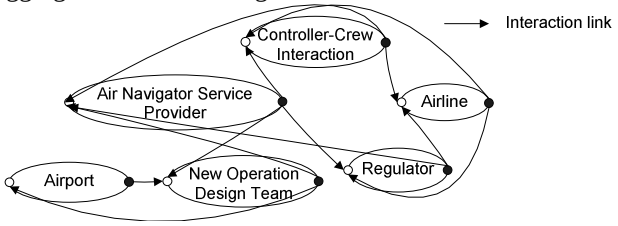


Fig. 1. The interaction relations between the generic roles at the aggregation level 1

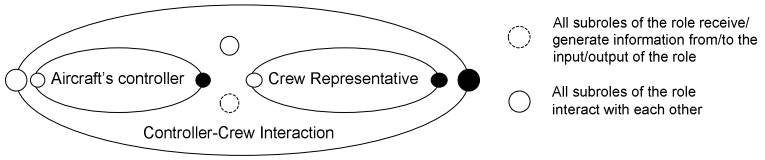


Fig. 2. The additional role for the interaction between the ANSP and the Airline composite depicted at the aggregation level 2

The subroles of the role ANSP are depicted at the aggregation level 2 in Figure 3 below.

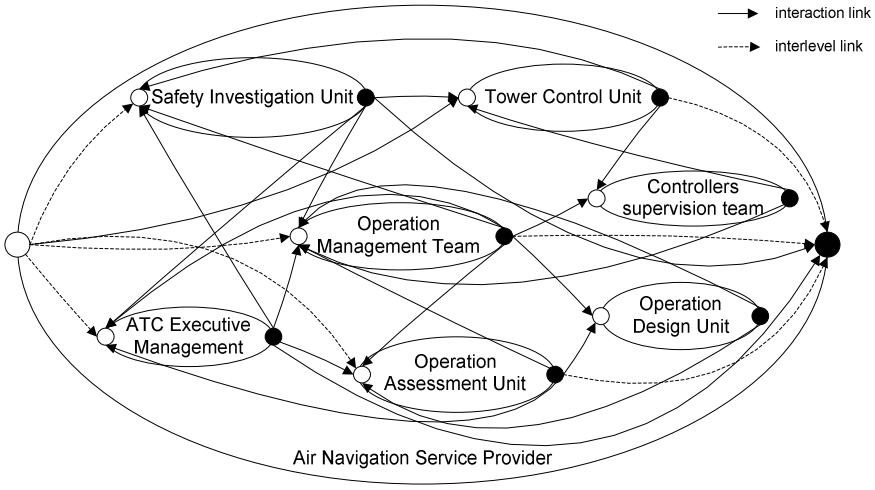
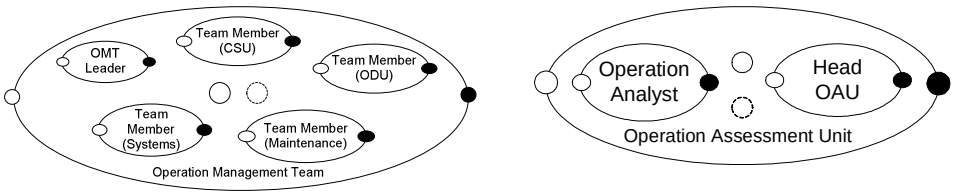


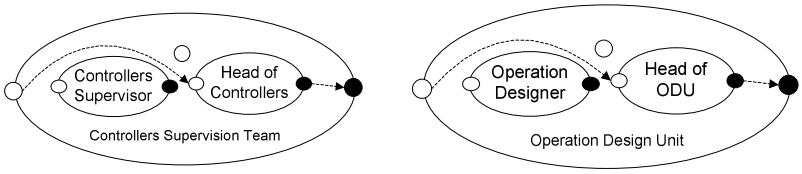
Fig. 3. The interaction relations between the subroles of the role Air Nativation Service Provider at the aggregation level 2

The subroles of each role of the ANSP at the level 2 are depicted at the level 3 below in Figures 4-6.



(a) (b)

Fig. 4. The interaction relations between the subroles of the role Operation Management Team (a) and of the role Operation Assessment Unit (b) at the aggregation level 3



(a) (b)

Fig. 5. The interaction relations between the subroles of the role Operation Design Unit (a) and of the role Controllers Supervision Team (b) at the aggregation level 3

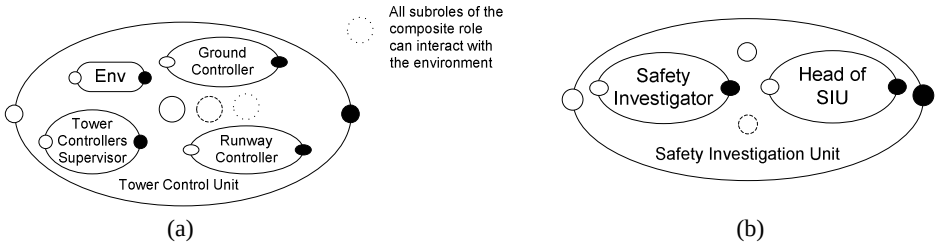


Fig. 6. The interaction relations between the subroles of the role Tower Control Unit (a) and of the role Safety Investigation Unit (b) at the aggregation level 3

The subroles of the New Operation Design Team role are depicted at the level 2 in Figure 7.

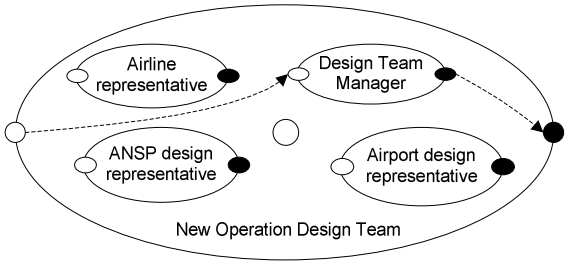


Fig. 7. The interaction relations between the subroles of the role New Operation Design Team at the aggregation level 2.

The subroles of the Airport role are given in Figure 8 (a) and (b) at the aggregation levels 2 and 3 correspondingly.

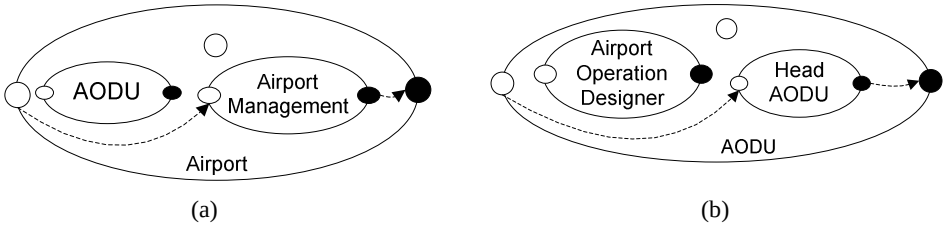
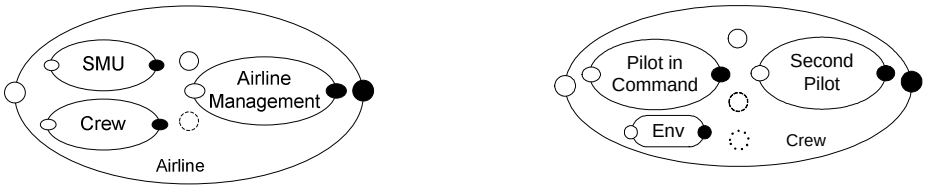


Fig. 8. The interaction relations between the subroles of the role Airport at the aggregation level 2 (a) and of its subrole Airport Operation Design Unit at the aggregation level 3 (b)

The subroles of the Airline role are given in Figure 9 (a) and (b) at the aggregation levels 2 and 3 correspondingly.



(a) (b)

Fig. 9. The interaction relations between the subroles of the role Airline at the aggregation level 2 (a) and of its subrole Crew at the aggregation level 3 (b)

Note that in principle the environment may be included in almost every composite role of the ATCO, as almost all of these roles and their subroles interact with the environment in reality. However, for the purposes of this case study, for the considered tasks only the interactions of the controllers and the pilots with the environment will be modeled explicitly.

Ontologies

For each role three types of ontologies are specified: an input ontology, an output ontology and an internal ontology. Input and output ontologies are often referred to as interface ontologies, which are used to describe interactions with other roles. An internal ontology is used to specify internal states of agents allocated to roles.

For specifying communications the interface ontologies for all roles include the following predicate:

communication_from_to: ROLE x ROLE x MSG_TYPE x CONTENT

Here the first argument denoted the role-source of information, the second – the role-recipient of information, the third argument denoted the types of the communication (which may be one of the following {observe, inform, request, decision, readback}) and the fourth – the content of the communication. The sort ROLE is a composite sort that comprises all subsorts of the roles of particular types (e.g., CONTROLLER, PILOT). The sort CONTENT is also the composite sort that comprises all names of terms that are used as the communication content. Such terms are constructed from sorted constants, variables and functions in the standard predicate logic way. For example, a set of functions that are used to construct the content of communications generated by the controller role of at its output is given in Table 2.

Table 2. A set of functions that belong to the output ontology of the role Aircraft's Controller

Function	Short description
aircraft_approaches_to: AIRCRAFT x REGION → CONTENT	Specifies an aircraft that approaches to a sector or a runway
aircraft_moved_from_to: AIRCRAFT x REGION x REGION → CONTENT	Specifies an aircraft that moved from one region to another
aircraft_at: AIRCRAFT x REGION → CONTENT	Identifies an aircraft situated in a certain sector or on a certain runway
instruction: INSTRUCTION_TYPE x RUNWAY → CONTENT	Specifies an instruction (i.e., a clearance to cross, a clearance to take off, an instruction to position and hold) to a crew

change_frequency_to: FREQUENCY → CONTENT	Identifies a new frequency for a crew
clear_of_the_runway: AIRCRAFT x RUNWAY → CONTENT	Specifies that an aircraft is clear of a runway
notification_report_for: REPORT x INCIDENT → CONTENT	Specifies a notification report for an incident
start_execution: PROCESS → CONTENT	Specifies a process being executed by the role
start_monitoring: PROCESS → CONTENT	Specifies a process being monitored by the role
finish_execution: PROCESS → ACTION_TYPE	Specifies the process execution finished by the role
finish_monitoring: PROCESS → ACTION_TYPE	Specifies the process monitoring finished by the role
action: ACTION_TYPE → CONTENT	Specifies an action of the role

For example, to specify the communication of the decision of the agent allocated to the Aircraft's Controller role on the request for the clearance to cross the runway rw from the agent allocated to the role Crew Representative the following relation is used:

communication_from_to(Aircraft's Controller, Crew Representative, decision, instruction(position_and_hold, rw))

For the correct organizational functioning no distortion should occur to any data transferred between roles. Note that ontologies of roles connected by an interlevel link may not contain common elements. In this case the interlevel link is described by an ontology mapping between the corresponding elements of ontologies. Moreover, an ontology mapping associated with an interlevel link may be used for representing mechanisms of information abstraction. These mechanisms can be applied for transmitting (or generating) partial, aggregated or generalized information to the input (or from the output) of a role. For example, a version of the incident notification report 1 provided by the ANSP role to the Regulator may differ (can be more abstracted) from the original received by the role OMT within the ANSP. To specify the corresponding ontology mapping consider the following part of the specification for the organization-oriented view:

is_org_described_by(ATCO, Γ, Δ)
is_interaction_link_in(e1, Γ)
connects_to(e1, OMT, ANSP, Γ)
has_onto_mapping(e1, m1)
is_part_of_onto_map(notification_report1_original, notification_report1_abstracted_version, m1)

To specify beliefs of agents allocated to roles the following predicate is used:

belief: BEL_TYPE x ROLE x CONTENT

Here the first argument specifies the belief type from the set {observed, requested, requested_by, informed, informed_by, decision_provided, decision_provided_by}, the second argument specifies the role that initiated the belief creation, and the third argument specifies the content of the belief. For example, the following relation represents the

belief of the agent allocated to the Crew Representative role about the decision position and hold received from the Aircraft's Controller role:

belief(decision_provided_by, Aircraft's Controller, instruction(position_and_hold, rw))

Ontologies of all other roles are constructed in a similar way.

Step 3. The identification of the requirements for the roles

At this step an example of the requirements for the air traffic controller role (of any type) all of which are capabilities (i.e., knowledge or skills) is given:

(1). Passed a rigid medical examination.
(2) 2 or 4 year college degree before initiation of ATC training.
(3) Thorough knowledge of the air traffic management system and the flight regulations.
(4) Computer training.
(5) Air traffic control training.
(6) Excellent listening and communication skills.
(7) Quick decision-making skills.
(8) Ability to stand stress.
(9) Good short-term memory capabilities.

Note that the measure of the level of development may be associated with (some) capabilities. For example, controllers may differ in the amount of hours that they spent for air traffic control training. This may be expressed by assigning corresponding development levels of this capability to the controllers.

Step 4. The identification of the organizational performance indicators and goals

Organizational goals are specified as expressions built based on performance indicators (PIs). In Table 3 the goals considered in the case study together with the PIs on which they are based, are specified.

Table 3. The goals and PIs of the ATCO

#	Goal	Based on the PI
1	It is required to achieve a high level of completeness and accuracy of the identification of high level requirements for a new operation from all parties involved into the operation	the completeness and accuracy of the identification of high level requirements for a new operation from all parties involved into the operation
1.1	It is required to achieve a high level of completeness and accuracy of the identification of high level safety-related requirements for a new operation from all parties involved into the operation	the completeness and accuracy of the identification of high level safety-related requirements for a new operation from all parties involved into the operation
1.2	It is required to achieve a high level of completeness	the completeness and

	and accuracy of the identification of high level capacity- and volume-related requirements for a new operation from all parties involved into the operation	accuracy of the identification of high level capacity- and volume-related requirements for a new operation from all parties involved into the operation
1.3	It is required to achieve a high level of completeness and accuracy of the identification of all other high level requirements for a new operation from all parties involved in the operation	The completeness and accuracy of the identification of all other high level requirements for a new operation from all parties involved in the operation
2	It is required to achieve a high level of safety of a new implemented operation	the level safety of a new implemented operation
2.1	It is required to achieve a high level of safety of the implementation of a new operation	the level of safety of the implementation of a new operation
3	It is required to achieve a high level of quality of the internal investigation of a new operation	the level of quality of the internal investigation of a new operation
3.1	It is required to achieve a high level of thoroughness of the internal investigation of a new operation	the level of thoroughness of the internal investigation of a new operation
3.2	It is required to maintain a high professional level of operation analysts	the professional level of operation analysts
3.3	It is required to maintain up-to-date knowledge of norms, standards and statistics used for the evaluation of a new operation	knowledge of norms, standards and statistics used for the evaluation of a new operation
4	It is required to achieve a satisfactory realization of the high level requirements and their refinements in the concept of a new operation	the realization of the high level requirements and their refinements in the concept of a new operation
4.1	It is required to achieve a satisfactory realization of the safety-related requirements in the concept of a new operation	the realization of the safety-related requirements in the concept of a new operation
4.2	It is required to achieve a satisfactory realization of the capacity- and volume-related requirements in the concept of a new operation	the realization of the capacity- and volume-related requirements in the concept of a new operation
4.3	It is required to achieve a satisfactory realization of other types of requirements in the concept of a new operation	the realization of other types of requirements in the concept of a new operation
4.4	It is required to achieve great involvement of the experts (e.g., controllers) possessing knowledge in the domain of the operation in the process of operation design	the involvement of the experts possessing knowledge in the domain of the operation in the process of operation design
5	It is required to achieve a high level of quality of the external investigation of a new operation	the level of quality of the external investigation of a

		new operation
6	It is required to achieve a high level of effectiveness and efficiency of a new introduced operation	the level of effectiveness and efficiency of a new introduced operation
6.1	It is required to achieve a high level of accuracy of the implementation of the concept of a new operation	the level of accuracy of the implementation of the concept of a new operation
7	It is required to achieve a high level of elaboration of the high level requirements for a new operation	the level of elaboration of the high level requirements for a new operation
7.1	It is required to achieve a high level of elaboration of the identified high level safety-related requirements for a new operation	the level of elaboration of the identified high level safety-related requirements for a new operation
7.2	It is required to achieve a high level of elaboration of the identified high level capacity-related requirements for a new operation	the level of elaboration of the identified high level safety-related requirements for a new operation
7.3	It is required to achieve a high level of elaboration of the other identified high level requirements for a new operation	the level of elaboration of the other identified high level requirements for a new operation
8	It is required to achieve a high level of productivity of the collaboration within the New Operation Design Team during the development of the concept for a new operation	the level of productivity of the collaboration within the New Operation Design Team during the development of the concept for a new operation
8.1	It is required to maintain a high professional level of the members of the team	the professional level of the members of the team
8.2	It is required to maintain constructive discussions during the development of a new operation	discussions during the development of a new operation
8.3	It is required to maintain a high level of consideration of the opinions of different members of the team	the level of consideration of the opinions of different members of the team
9	It is required to minimize the development and assessment time of a new operation	the development and assessment time of a new operation
9.1	It is required to maintain a frequent collaboration between the New Operation Design Team and the Operation Assessment Unit of the Air Navigation Service Provider during the development of the concept for a new operation	collaboration between the NODT and the OAU of the ANSP during the development of the concept for a new operation
9.2	It is required to minimize the development time of the concept of a new operation	the development time of the concept of a new operation
9.3	It is required to minimize time for an external assessment of a new operation	the time for an external assessment of a new operation
9.4	It is required to minimize time for an internal assessment of a new operation	the time for an internal assessment of a new operation

10	It is required to maintain a high level of safety of execution of tasks related to the air traffic management	the level of safety of execution of tasks related to the air traffic management
10.1	It is required to maintain a high level of conformance of all roles involved into the air traffic management to the formal norms and regulations defined for their tasks	the level of conformance of all roles involved into the air traffic management to the formal norms and regulations defined for their tasks.
10.2	It is required to maintain a high (sufficient) level of proficiency of pilots	the level of proficiency of pilots
10.2.1	It is required to maintain a high (sufficient) level of proficiency of pilots operating in regular conditions	the level of proficiency of pilots operating in regular conditions
10.2.2	It is required to maintain a high (sufficient) level of proficiency of pilots operating in non-stationary (hazardous) conditions	the level of proficiency of pilots operating in non-stationary (hazardous) conditions
10.3	It is required to maintain a high (sufficient) level of proficiency of controllers	the level of proficiency of controllers
10.3.1	It is required to maintain a high (sufficient) level of proficiency of controllers operating in regular conditions	the level of proficiency of controllers operating in regular conditions
10.3.2	It is required to maintain a high (sufficient) level of proficiency of controllers operating in non-stationary (hazardous) conditions	the level of proficiency of controllers operating in non-stationary (hazardous) conditions
10.4	It is required to maintain the high quality and reliability of communication lines between roles that are supposed to communicate during the execution of the tasks related to the air traffic management	the quality and reliability of communication lines between roles that are supposed to communicate during the execution of the tasks related to the air traffic management
10.5	It is required to maintain the high quality and reliability of communication lines between the roles involved into the air traffic management and the environment	the quality and reliability of communication lines between the roles involved into the air traffic management and the environment
10.6	It is required to maintain the high quality and reliability of the hardware used in the air traffic control management	the quality and reliability of the hardware used in the air traffic control management
11	It is required to maintain an up-to-date set norms and regulations that ensure the safe execution of the air traffic management tasks	the set norms and regulations that ensure the safe execution of the air traffic management tasks
11.1	It is required to maintain a sufficient proficiency level of regulators and other norm- and regulation-makers	the proficiency level of regulators and other norm- and regulation-makers
11.2	It is required to maintain the regular monitoring of flight data to identify potential hazards and to improve	the regularity of the monitoring of flight data to

	the safety	identify potential hazards and to improve the safety
11.3	It is required to maintain the regular investigation of potential safety hazards	the investigation of potential safety hazards
11.4	It is required to maintain the regular performance of risk assessment of operations.	the performance of risk assessment of operations
11.5	It is required to maintain a timely update of norms and regulations based on investigation reports	the timeliness of update of norms and regulations based on investigation reports
12	It is required to maintain a consistent set of norms and regulations for the execution of the air traffic control management tasks	the consistency of a set of norms and regulations for the execution of the air traffic control management tasks
13	It is required to maintain a high level of effectiveness and efficiency of the work organization within the Tower Control Unit	the level of effectiveness and efficiency of the work organization within the Tower Control Unit
13.1	It is required to maintain effective coordination of the task execution within the Tower Control Unit	the coordination of the task execution within the Tower Control Unit
13.2	It is required to maintain high flexibility of the task allocation to the controllers within the Tower Control Unit	the flexibility of the task allocation to the controllers within the Tower Control Unit
13.3	It is required to maintain a high level of collaboration within the Tower Control Unit	The level of collaboration within the Tower Control Unit
14	It is required to maintain a high level of effectiveness and efficiency of the work organization of the crew of an aircraft	the level of effectiveness and efficiency of the work organization of the crew of an aircraft
14.1	It is required to maintain a high flexibility of the task distribution between the pilots of a crew	the flexibility of the task distribution between the pilots of a crew
14.2	It is required to maintain a high level of collaboration between the pilots of the crew	the level of collaboration between the pilots of the crew
14.3	It is required to maintain a high level of collaboration during decision making in the crew	the level of collaboration during decision making in the crew
15	It is required to maintain the timely execution of the processes of the air traffic management	the timeliness of the execution of the processes of the air traffic management
16	It is required to maintain a high level of robustness and unambiguousness of the control (coordination) structure for the execution of tasks	the level of robustness and unambiguousness of the control (coordination) structure for the execution of tasks
16.1	It is required to maintain a high level of robustness and unambiguousness of the control (coordination)	the level of robustness and unambiguousness of the

	structure for the execution of tasks in standard conditions	control (coordination) structure for the execution of tasks in standard conditions
16.2	It is required to maintain a high level of robustness and unambiguousness of the control (coordination) structure for the execution of tasks in non-stationary (exceptional) conditions	the level of robustness and unambiguousness of the control (coordination) structure for the execution of tasks in non-stationary (exceptional) conditions
17	It is required to maintain timely reporting of incidents/hazards	the timeliness of reporting of incidents/hazards
18	It is required to maintain timeliness and a high quality of the incident investigation	the timeliness and quality of the incident investigation
18.1	It is required to maintain a high proficiency level of incident investigators	the proficiency level of incident investigators
18.2	It is required to maintain a sufficient level of details of (incident/hazard) notification reports	the level of details of (incident/hazard) notification reports
18.3	It is required to maintain the timely investigation of an incident/hazard	the timeliness of the investigation of an incident/hazard
18.4	It is required to maintain a high level of thoroughness of the incident investigation	the level of thoroughness of the incident investigation
19	It is required to maintain a high level of recognition of actual incidents/hazards from the potential ones	the level of recognition of actual incidents/hazards from the potential ones
20	It is required to maintain a sufficient level of autonomy of decision making and the operation execution for the roles involved into the air traffic management	The level of autonomy of decision making and the operation execution for the roles involved into the air traffic management
21	It is required to maintain unambiguousness, consistency, correctness and timeliness of information exchanged between agents	the unambiguousness, consistency, correctness and timeliness of information exchanged between agents
21.1	It is required to maintain a high level of unambiguousness and consistency of information exchanged between agents	unambiguousness and consistency of information exchanged between agents
21.2	It is required to maintain the timely provision of information to all agents that require this information	the timeliness of the provision of information to all agents that require this information
21.3	It is required to maintain the high correctness of information exchanged between agents	the correctness of information exchanged between agents
22	It is required to achieve a highly expeditious flow of air traffic at an airport	the flow of air traffic at an airport
22.1	It is required to maintain a high level of efficiency of scheduling of the aircrafts (for taxiing, departures, arrivals) at an airport	the level of efficiency of scheduling of the aircrafts (for taxiing, departures, arrivals) at an airport

23	It is desired to increase the volume of passengers, departing/arriving from/to an airport	the volume of passengers, departing/arriving from/to an airport
23.1	It is required to minimize the execution time of the air traffic management tasks	the execution time of the air traffic management tasks
23.2	It is desired to maximize the territory of an airport	the territory of an airport
23.3	It is desired to maintain low prices for services	the prices for services
24	It is desired to maintain a high level of job satisfaction of agents fulfilling the roles in all organizations	the level of job satisfaction of agents fulfilling the roles in all organizations
24.1	It is desired to maintain a sufficient level of motivation of every employee	the level of motivation of every employee
24.1.1	It is desired to maintain a sufficient level of autonomy for every employee	the level of autonomy for every employee
24.1.2	It is desired to maintain a sufficient amount of feedback for every employee	the amount of feedback for every employee
24.2	It is desired to achieve that an effective reward system is developed	the effectiveness of the organization reward system

Other characteristics of the identified goals are given in Table 4. A part of these characteristics are obtained from the formal documents of the considered ATCO. Other characteristics (such as the priority and the negotiability) are assigned based on the informal evidences (such as case studies, interviews, organization investigation documents).

Table 4. The characteristics of the goals considered in the case study

#	Priority	Horizon	Ownership	Perspective	Hardness	Negotiability
1	2	short-term	ATCEM, OMT	management	soft	neg.
1.1						
1.2						
1.3						
2	3		NODT, Regulator, Airport, ANSP	management, customer		non-neg.
2.1			Airport, ANSP			
3	2		OAU, NODT	management		neg.
3.1			OAU		hard	
3.2						
3.3						
4			NODT		soft	
4.1						
4.2						
4.3						
4.4	1					
5	2		Regulator, NODT			
6	3		NODT, OAU, Regulator, Airport, ANSP	management, customer		
6.1			Airport, ANSP	management		non-neg.
7	2		NODT			neg.
7.1						
7.2						
7.3						

8			NODT, ATCEM, OMT			
8.1		long-term	NODT		hard	non-neg.
8.2		short-term			soft	neg.
8.3					hard	
9			NODT, OAU, Regulator		soft	
9.1			NODT, OAU		hard	
9.2			NODT			
9.3			Regulator			
9.4			OAU			
10	3	long-term	TCU, SCU, SIU, Airline	management, customer	soft	non-neg.
10.1					hard	
10.2			Pilot in Command, Second Pilot			
10.3			TCU, CSU	management		
10.4			TCU, CSU, SIU, Airline			
10.5						
10.6						
11			SIU, Regulator		soft	neg.
11.1					hard	
11.2	2		SIU			
11.3			SIU, Regulator			
11.4						
11.5						
12	3					non-neg.
13	2		TCU, CSU		soft	neg.
13.1						
13.2			TCU			
13.3						
14			Pilot in Command, Second Pilot			
14.1						
14.2						
15	3		Airline, TCU, CSU, SIU	management, customer	hard	
16	2		All roles	management		
16.1						
16.2						
17			TCU, Airline			
18			SIU, Regulator		soft	
18.1	3				hard	non-neg.
18.2	2				soft	neg.
18.3	3				hard	
18.4	2				soft	
19			TCU, SIU, Regulator		hard	
20			TCU, SIU, Crew, SIU		soft	
21	3		All roles		hard	non-neg.
21.1						
21.2						
21.3						
22			Airline, TCU, CSU		soft	neg.
22.1			TCU, CSU			
23			Airline, TCU, CSU, Airport, OMT, ATCEM			
23.1			TCU, CSU, OMT,		hard	

			ATCEM			
23.2	2		Airport, ATCEM, Airline			
23.3	3		Airport, ATCEM, Airline	management, customer		
24	2		All roles	management	soft	

The relations between the PIs on which the considered goals are based are depicted in Fig. 10. The relations between the PIs have been identified based on expert knowledge from the air traffic control domain.

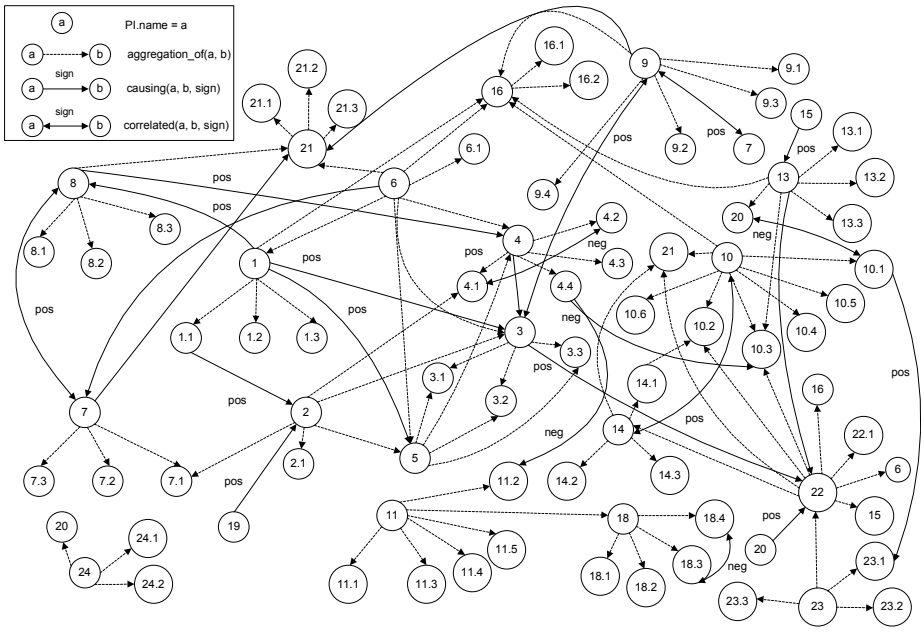


Fig. 10. Relations between the PIs considered in the case study

Based on the relations between PIs the relations between corresponding goals are specified (see Fig. 11). Note that the types of refinement relations of the soft goals not explicitly identified in Fig. 11 are the satisfices relations.

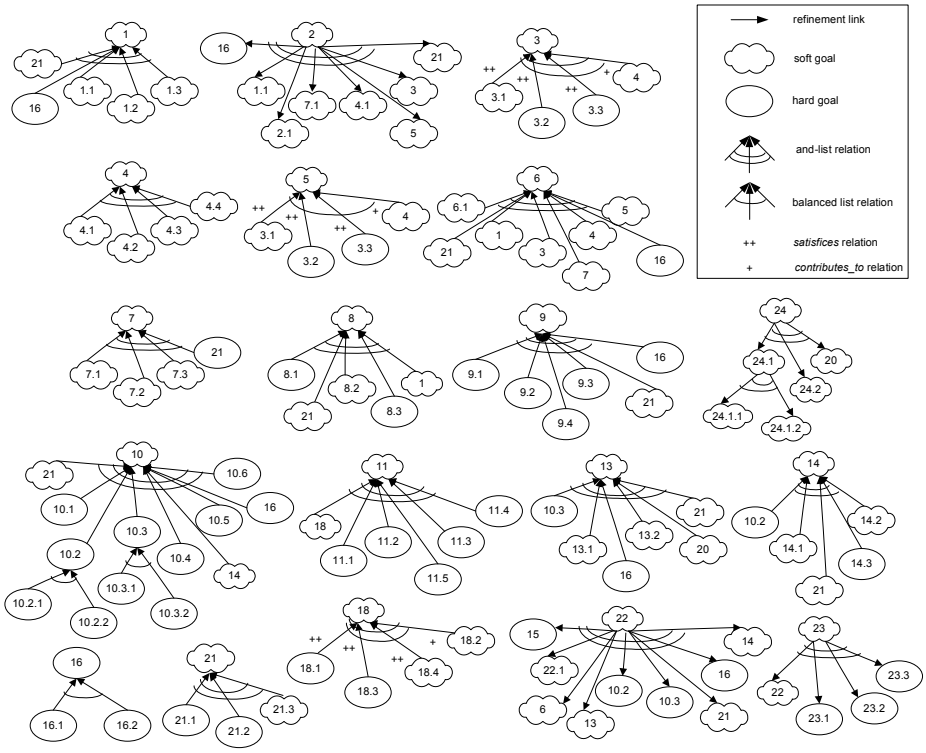


Fig. 11. The refinements of the goals considered in the case study

Step 5. The specification of the resources

Some of the resource types used in the case study are given in Table 5.

Table 5. Some of the resource types used in the case study

Name	Category	Measurement unit	Expiration duration
Airport's diagram	Discrete	Item	Conditional: until any changes in the airport's layout are performed
Aircraft	Discrete	Item	Depending on the aircraft's type
Runway Incursion Alert System (RIAS)	Discrete	Item	10 years
Radar	Discrete	Item	10 years
Communication R/T system	Discrete	Item	10 years

Different types of data represent special resource types, some of which are listed in Table 6. Note that a conditional expiration duration is specified by an executable rule in the

Table 6. Some of the resource types used in the case study

Name	Expiration duration
Taxiing instructions	For one flight
Incident classification database	Conditional: until any changes are introduced
Data about the new frequency of a controller	Conditional: for the time during which an aircraft is situated in the controller's region
Clearance to cross a runway	Conditional: until the runway is crossed by the aircraft that received the clearance
Clearance to takeoff from a runway	Conditional: until the aircraft that received the clearance took off
The readback of a controller's instruction by the Crew Representative	Conditional: until the readback is received and checked by the controller guiding the aircraft
A notification report	1 year
The results of the incident investigation	25 years
An incident investigation report	50 years
A list of identified hazards	1 year
Operation requirements	50 years
An intermediate concept for a new operation	Conditional: until the final concept of a new operation is created
The final concept for a new operation	50 years

Step 6. The identification of the organizational tasks, the relations between the tasks, and relations between the tasks, the resources and the goals

In Table 7 the names, the short descriptions and the durations of the tasks considered in the case study are described. Furthermore, this table identifies relations between the lowest level tasks and goals. Sets of goals that correspond to higher level tasks are formed by combination of all goals that correspond to the subtasks of these tasks.

Table 7. The tasks considered in the case study, their characteristics and the relations to the goals

#	Task name	Short description	Durations
1	Taxiing the aircraft to the designated runway	Taxiing the aircraft on the taxiways and through the runways to the designated runway according to the taxiing instructions provided to the crew	Depends on the durations of subtasks
1.1	Taxiing the aircraft on a taxiway	-	Depends on a particular taxiway
Goal: 13.2, 14.1, 14.3, 24.1.1, 14.2, 23.1, 20			
1.2	Switching to the frequency of another controller	The action of switching to the frequency of the controller, who will continue the guidance of an aircraft	Min: 1 sec Max: 5 sec
Goal: 14.1, 14.3, 14.2			
1.3	Inquiry for the clearance for crossing an active runway	An inquire to the controller currently guiding the aircraft	Min: 2 sec Max: 5 sec
Goal: 14.1, 14.3, 14.2			

1.4	Making and communicating the decision on a request for crossing a runway	The decision on a request from a crew for crossing an active runway is made by the controller currently guiding the aircraft of the crew	Min: 3 sec Max: 11 sec
Goal: 13.1, 22.1, 24.1.1, 23.1, 20			
1.5	Crossing a runway	-	Min: 30 sec Max: 60 sec
Goal: 14.1, 14.3, 23.1, 14.2, 20			
1.6	Provision of data about a new frequency to a crew	The data are provided to the aircraft's crew by the controller currently guiding the aircraft before the aircraft is handed over to another controller	Min: 2 sec Max: 6 sec
Goal: 10.1			
1.7	Readback of a pilot of the controller's instructions	All instructions provided by controllers to a crew should be read back by one of the pilots of the crew and corrected by the controller if necessary	Min: 2 sec Max: 6 sec
Goal: 14.1, 14.3, 21.1, 21.3, 14.2			
1.8	Transfer of control over an aircraft between controllers	The transfer of control is performed by means of strips that are handed over between the controllers	Min: 2 sec Max: 5 sec
Goal: 13.1, 13.3, 23.1			
2	Acquiring a takeoff allowance	When an aircraft is close to the designated runway the crew initiates the acquiring of the allowance for takeoff from the Runway Controller responsible for the runway	Depends on the durations of subtasks
2.1	Request for clearance to take off	The request is communicated by the aircraft's crew to the Runway Controller of the designated runway	Min: 2 sec Max: 5 sec
Goal: 14.1, 14.3, 14.2			
2.2	Making and communicating the decision on a request for takeoff	The decision is made by the controller responsible for the runway	Min: 3 sec Max: 11 sec
Goal: 13.1, 22.1, 24.1.1, 23.1, 20			
3	Take off	-	Min: 30 sec Max: 60 sec
Goal: 10.1, 14.1, 14.3, 15, 20, 23.1, 14.2, 13.2			
4	Incident reporting management based on the data provided by a controller	The incident reporting loop initiated by a (runway or ground) controller	Min: 1 day Max: 160 days
4.1	Create a notification report	When a controller observes an occurrence that may be classified as an incident/accident, s/he is obliged to create a notification report	Min: 1 min Max: 2 hours
Goal: 17, 18.2, 19, 20			
4.2	Preliminary processing of a notification report	A notification report created by a controller is examined and improved by his/her supervisor. The occurrence described in the report is classified.	Min: 1 min Max: 2 days

Goal: 19, 13.3, 20			
4.3	Making decision about the investigation necessity based on the provided notification report	If the occurrence is of a high severity, the incident/accident investigation will be initiated. The lower the level of severity of the occurrence, the less the chance that the occurrence will be immediately investigated	Min: 1 day Max: 30 days
Goal: 19, 20			
4.4	Investigation of the occurrence based on the notification report	During the investigation the (possible) causes of the incident/accident are identified, and based on the investigation results recommendations are provided	Min: 3 days Max: 90 days
Goal: 18.3, 18.4, 20			
4.5	Discussion of the intermediate occurrence investigation results	The intermediate results of the investigation are provided to the OMT of the ANSP	Min: 1 day Max: 15 days
Goal: 11.5, 12			
4.6	Reviewing of the occurrence investigation results	The Executive Board of the ATC reviews the incident/accident investigation results	Min: 5 min Max: 4 hours
Goal: 11.5, 12, 20			
4.7	Distribute the occurrence investigation report among all concerned roles	-	Min: 1 hour Max: 15 days
Goal: 21.2, 21.3, 21.1			
5	Incident reporting management based on the data provided by a crew	The incident reporting loop initiated by a crew	Depends on the durations of subtasks
5.1	Create a notification report and provide it to the SMU	When a pilot (a crew) observes an occurrence that may be classified as an incident/accident, s/he is obliged to create a notification report, which may be further provided to the SMU of the airline by which the pilot is employed	Min: 2 hours Max: 2 days
Goal: 14.1, 14.3, 17, 18.2, 19, 14.2, 20			
5.2	Create a notification report and provide it to the regulator	A notification report created by a pilot may be provided directly to the regulator	Min: 1 day Max: 14 days
Goal: 14.1, 14.3, 17, 18.2, 19, 14.2, 20			
5.3	Process a notification report and provide it to the regulator	A notification report created by a pilot (a crew) is examined and improved by the SMU and provided further to the regulator for further investigation	Min: 1 day Max: 90 days
Goal: 19, 20			
5.4	Making decision about the investigation necessity and about the role-investigator	The decision is based on the notification report and the choice of the role-investigator is based on the severity of the incident/accident, the availability of roles and the competences of the available roles	Min: 1 day Max: 30 days

Goal: 18.3, 18.4, 20			
6	Identification of hazards, safety problems and trends	Once in three months the SIU of the ANSP performs an investigation on the collected notification reports and the occurrence investigation results. The aim of this investigation is to identify safety hazards, problems and trends	Min: 20 days Max: 30 days
Goal: 11.3, 11.4, 15, 11.2, 20			
7	Design and evaluate a new operation	-	Depends on the durations of subtasks
7.1	Produce an intermediate design for a new operation	During the development of the concept of a new operation a number of intermediate design concepts are produced which are provided for the further evaluation	Depends on the particular operation. For the operation "runway introduction": Min: 2 weeks Max: 1 month
Goal: 4.1, 4.2, 4.3, 7.1, 7.2, 7.3, 8.1, 8.2, 8.3, 9.1, 9.2, 24.1.1, 20			
7.2	Assess an intermediate design for a new operation internally	The internal evaluation of an intermediate design within the ANSP	Min: 2 days Max: 1 week
Goal: 3.1, 9.1, 9.4, 20			
7.3	Produce the final concept of a new operation	Based on the previous intermediate design concepts and the results of the internal evaluation, the final concept of operation is produced	Min: 4 days Max: 2 week
Goal: 4.1, 4.2, 4.3, 7.1, 7.2, 7.3, 8.1, 8.2, 8.3, 9.2, 20			
7.4	Assess the final concept of a new operation externally	The final concept of a new operation is assessed externally	Min: 2 week Max: 1 month
Goal: 3.1, 3.2, 3.3, 9.3, 5, 20			
7.5	Review the final concept of a new operation	-	Min: 3 days Max: 1 week
Goal: 4.1, 4.2, 4.3, 6.1, 20			
8	Implement a new operation	-	Depends on the particular operation. For the operation "runway introduction": Min: 3 month Max: 1 year
Goal: 2.1, 20			
9	Create a list of high level requirements for a new operation	This task precedes and produces an input for the final concept development task. Both safety and performance-related goals should be reflected in the requirements.	Min: 1 month Max: 3 month
Goals: 1, 20			
10	Schedule training for a	-	Min: 1 min

	pilot		Max: 5 min
Goal: 10.2, 14, 20			
11	Schedule training for a controller	-	Min: 1 min Max: 5 min
Goals:10.3, 20			
12	Schedule training for an operation analyst	-	Min: 1 min Max: 5 min
Goals: 3.2, 20			
13	Schedule training for a safety investigator	-	Min: 1 min Max: 5 min
Goal: 18.1, 11.1, 20			
14	Get training	-	Depends on the particular training type
Goal: 10.2, 14, 10.3, 3.2, 18.1, 11.1			
15	Allocation of agents to controllers roles	-	Constantly
Goal: 13.2, 16, 20			

In Fig. 12 the decompositions of the identified composite tasks are depicted.

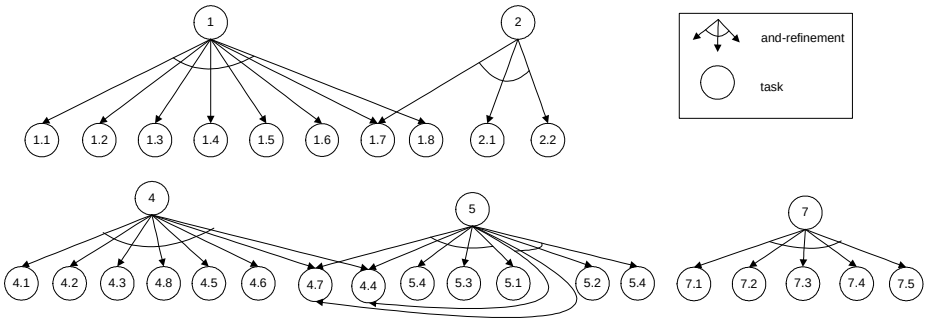


Fig. 12. The refinement of composite tasks into subtasks

The relations between the identified tasks and resource types are given in Table 8. Note that only the relations between the simple tasks and resource types are shown. The resource types that are used/consumed/produced by a composite task comprise all the resource types that are used/consumed/produced by all the subtasks of this task.

Table 8. The relations between the identified tasks and the resource types that these tasks use/ consume/ produce

Task	Task uses	Task produces
1.1	the airport's diagram, the taxi instructions for the flight, compass, radar, visual observations, aircraft	-
1.2	data about the new frequency	-
1.3	the observation of the close proximity of the active runway to be traversed,	a request to the Runway Controller responsible for the runway for the clearance for crossing the

	the taxi instructions for the flight, communication R/T system	runway
1.4	data about the current state of the runway, a request from the aircraft's crew to the Runway Controller responsible for the runway for the clearance for crossing the runway, communication R/T system	the instruction to the crew (which may be 'position and hold' (wait) or 'the clearance to cross is provided')
1.5	the clearance from the Runway Controller for crossing the runway, the airport's diagram, the taxiing instructions, compass, radar, visual observations, aircraft	the pilot's report 'clear of the runway' to the Runway Controller controlling the runway
1.6	the observation that the aircraft is approaching to the margins the current sector, the data about the controller of the adjoining region (i.e., a sector or a runway), communication R/T systems	data about the frequency of the controller of the adjoining sector
1.7	the controller's instruction, communication R/T systems	the correct readback of the controller's instruction
1.8	the observation that the aircraft is approaching to the margins the current sector, the data about the controller of the adjoining region (i.e., a sector or a runway)	the strip with the aircraft's details provided to the controller, who will be guiding the aircraft
2.1	the observation of the close proximity of the designated runway, communication R/T systems	a request to the Runway Controller of the designated runway for the clearance to take off
2.2	data about the current state of the runway, a request to the Runway Controller of the designated runway for the clearance to take off, communication R/T systems	the instruction to the crew (which may be 'position and hold' (wait) or 'the clearance to takeoff is provided')
3	the clearance from the Runway Controller to takeoff, compass, radar, visual observations, aircraft, RIAS	the pilot's report 'clear of the runway' to the Runway Controller controlling the runway
4.1	the observation from the environment of an occurrence that may be classified as an incident/accident, the incident classification database	a notification report
4.2	a notification report	a processed notification report
4.3	a processed notification report	a decision on the initiation of the occurrence investigation
4.4	a processed notification report, additional data about the occurrence (optional)	an incident investigation report
4.5	a processed notification report, an incident investigation report, occurrence statistics	recommendations based on the incident investigation report
4.6	an incident investigation report	directions to redesign some operation(s) (optional)
4.7	an incident investigation report, the list	-

	of all concerned roles	
5.1	the observation from the environment of an occurrence that may be classified as an incident/accident, the incident classification database	a notification report
5.2	the observation from the environment of an occurrence that may be classified as an incident/accident, the incident classification database	a notification report
5.3	a notification report	a processed notification report
5.4	a (processed) notification report	a decision on the initiation of the occurrence investigation, a decision concerning the role-investigator
6	Collected notification reports, incident/accident investigation reports, occurrence statistics	a report on the identified safety hazards, problems and trends
7.1	a list of high level requirements for a new operation, (optional) results of the internal assessment of the previous intermediate design concepts for a new operation, (optional) the previous intermediate design concept for a new operation.	an intermediate design concept for a new operation
7.2	an intermediate design for a new operation, the evaluation framework	the results of the internal assessment of an intermediate design for a new operation
7.3	operation requirements, the current intermediate design for a new operation	the final concept of a new operation
7.4	the final concept of a new operation, the evaluation framework	the results of the external assessment of the final concept of a new operation
7.5	the final concept of a new operation, the results of the internal assessment of the final concept of a new operation, the results of the external assessment of the final concept of a new operation	a permission/prohibition for the implementation of a new operation
8	the permission for the implementation of a new operation, the final concept of a new operation	the new operation implemented
9	-	a list of high level requirements for a new operation
10	data about a pilot	training scheduled for a pilot
11	data about a controller	training scheduled for a controller
12	data about an operation analyst	training scheduled for an operation analyst
13	data about a performance investigator	training scheduled for a performance investigator
14	data about a scheduled training	-
15	Data about the available agents and their workload, data about the roles that have to be allocated	-

Step 7. The specification of the authority relations

The *responsibility relations* of the roles on different aspects of the identified tasks are given in Table 9. The connective "or" between two role names denotes that both roles are *authorized* for some aspect of a task.

Table 9. The responsibility relations of the roles on different aspects of the identified tasks

Task	Execution	Monitoring	Consulting	Technological decisions	Managerial decisions
1.1	Crew	Runway Controller, Tower Controllers Supervisor	Runway Controller	Crew	Runway Controller, Tower Controllers Supervisor
1.2	Crew		Runway Controller	Crew	
1.3	Crew				
1.4	Runway Controller	Tower Controllers Supervisor	Tower Controllers Supervisor, other Runway and Ground Controllers	Runway Controller	
1.5	Crew	Runway Controller	Runway Controller, Tower Controllers Supervisor	Crew	Crew, Runway Controller, Tower Controllers Supervisor
1.6	Runway or Ground Controller	Tower Controllers Supervisor		Runway Controller	
1.7	Pilot	Runway or Ground Controller			
1.8	Aircraft's controller (i.e., controller currently responsible for the aircraft)	Tower Controllers Supervisor		Aircraft's controller	
2.1	Crew				
2.2	Runway Controller	Tower Controllers Supervisor		Runway Controller	
3	Crew	Runway Controller	Crew		Crew, Runway Controller
4.1	Runway or Ground Controller				
4.2	Tower Controllers Supervisor				
4.3	SIU				
4.4	Safety Investigator	Head SIU		Safety Investigator	Safety Investigator, Head SIU
4.5	OMT				

4.6	ATCEM			
4.7	OMT			
4.8	Safety Investigator or Regulator	OMT, ATCEM	Safety Investigator, Regulator	OMT, ATCEM
5.1	Crew			
5.2	Crew			
5.3	SMU			
5.4	Regulator			
6	Regulator			
7.1	New Operation Design Team			
7.2	Operation Analyst	Head OAU	Operation Analyst	Head OAU
7.3	New Operation Design Team			
7.4	Regulator			
7.5	OMT, ATCEM, Airport Management			
8	Airport			
9	OMT, ATCEM, Airport Management, Airline Management			
10	Airline Management			
11	Tower Controllers Supervisor			
12	Head OAU			
13	Head SIU			
14	Pilot, Controller, Operation Analyst, Safety. Investigator	Airline Management, Tower Controllers Supervisor, Head OAU, Head SIU		
15	Tower Controllers Supervisor			

In Table 10 the superior-subordinate relations on the identified roles with respect to the tasks are specified.

Table 10. The superior-subordinate relations on the identified roles with respect to the tasks

Subordinate role	Superior role	Task
ANSP design representative	Design Team Manager	7.1, 7.3
Airport design representative	Design Team Manager	7.1, 7.3
Airline representative	Design Team Manager	7.1, 7.3
OMT	ATCEM	7.2, 7.4
OAU	OMT	7.2, 7.4
Operation Analyst	Head OAU	7.2, 7.4
Runway Controller	Tower Controllers Supervisor	1.4
Ground Controller	Tower Controllers Supervisor	1.1
Runway Controller	Tower Controllers Supervisor	1.5
Runway Controller	Tower Controllers Supervisor	2.2
Runway Controller	Tower Controllers Supervisor	4.1
Safety Investigator	Head SIU	4.4, 6
OMT	ATCEM	4.4, 6
SIU	OMT	4.4, 6

Step 8. The specification of the flows of control

At this step the workflows for the execution of the general tasks identified at the beginning are designed. The workflow for the taxiing of an aircraft to the designated runway and for the subsequent take off is given in Fig. 13.

A formal specification for this workflow is given below. In particular, this specification identifies the temporal relations and the delays between the processes.

is_instance_of(Allocation of agents to controllers roles, Allocation of agents to controllers roles)
is_instance_of(Transfer of control over the aircraft, Transfer of control over an aircraft between controllers)
is_instance_of(Provision data on the new frequency, Provision of data about a new frequency to a crew)
is_instance_of(Switching to the new frequency, Switching to the frequency of another controller)
is_instance_of(Taxiing the aircraft on the taxiway, Taxiing the aircraft on a taxiway)
is_instance_of(Inquire for clearance to cross, Inquiry for the clearance for crossing an active runway)
is_instance_of(Repeated inquiry for clearance to cross, Inquiry for the clearance for crossing an active runway)
is_instance_of(Decision making on the request for the clearance, Making and communicating the decision on a request for crossing a runway)
is_instance_of(Readback of the crossing instructions, Readback of a pilot of the controller's instructions)
is_instance_of(Readback of the take off instructions, Readback of a pilot of the controller's instructions)
is_instance_of(Crossing the runway, Crossing a runway)
is_instance_of(Request for the clearance to take off, Request for clearance to take off)
is_instance_of(Decision making concerning the request for the clearance to take off, Making and communicating the decision on a request for takeoff)
is_instance_of(Take off, Take off)
starts_after(begin_and(and4), begin, 0)
starts_after(begin_loop(c5), begin_and(and4), 0)
loop_cond(c5, "workflow is not finished")
loop_max(c5, 50)
starts_after(begin_or(or3), begin_loop(c5), 0)
or_cond(or3, "A new allocation of agents controllers to roles?")
or_branch(or3, yes, Allocation of agents to controllers roles)
or_branch(or3, no, end_or(or3))
starts_after(end_or(or3), Allocation of agents to controllers roles, 0)
starts_after(end_loop(c5), end_or(or3), 0)
starts_after(end_and(and4), end_loop(c5), 0)
and_cond(and4, all)

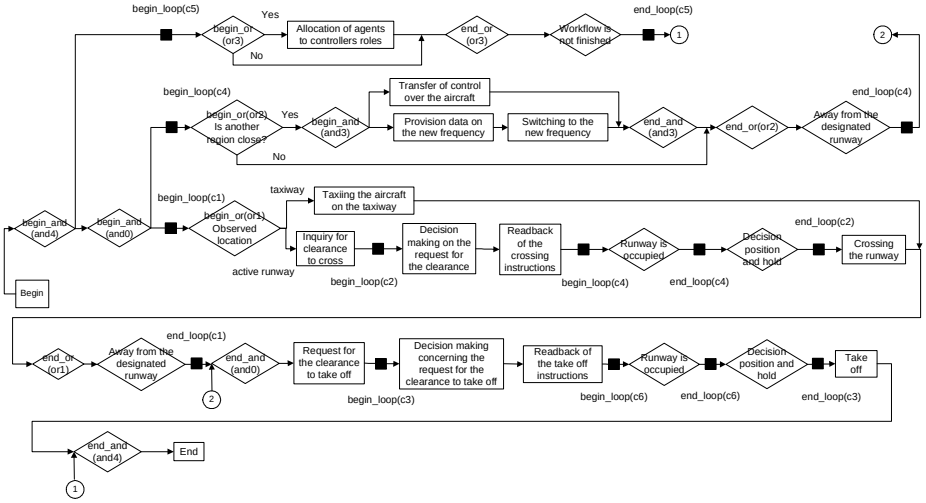


Fig. 13. The workflow for the for the taxiing of an aircraft to the designated runway and for the subsequent take off

starts_after(begin_and(and0), begin_and(and4), 0)
starts_after(begin_loop(c4), begin_and(and0), 0)
starts_after(begin_or(or2) ,begin_loop(c4), 0)
or_cond(or2, “is another region close?”)
or_branch(or2, yes, begin_and(and3))
or_branch(or2, no, end_or(or2))
starts_after(Transfer of control over the aircraft, begin_and(and3), 5)
starts_after(end_and(and3), Transfer of control over the aircraft, 0)
starts_after(Provision data on the new frequency, begin_and(and3), 1)
starts_after(Switching to the new frequency, Provision data on the new frequency, 1)
starts_after(end_and(and3), Switching to the new frequency, 0)
and_cond(and3, all)
starts_after(end_or(or2), end_and(and3), 0)
starts_after(end_loop(c4), end_or(or2), 0)
loop_cond(c4, “away from the designated runway”)
loop_max(c4, 5)
starts_after(end_and(and0), end_loop(c4), 0)
and_cond(and0, all)
starts_after(begin_loop(c1), begin_and(and0), 0)
loop_cond(c1, “away from the designated runway”)
loop_max(c1, 7)
starts_after(begin_or(or1), begin_loop(c1), 0)
or_cond(or1, “observed location”)
or_branch(or1, taxiway, Taxiing the aircraft on the taxiway)
or_branch(or1, active runway, Inquire for clearance to cross)
starts_after(end_or(or1), Taxiing the aircraft on the taxiway, 0)
starts_after(begin_loop(c2), Inquire for clearance to cross, 0)
loop_cond(c2, “decision position and hold”)
loop_max(c2, 4)
starts_after(Decision making on the request for the clearance, begin_loop(c2), 0)
starts_after(Readback of the crossing instructions, Decision making on the request for the clearance, 1)
starts_after(begin_loop(c4), Readback of the crossing instructions, 0)
loop_cond(c4, “runway is occupied”)

loop_max(c4, 1000)
starts_after(end_loop(c4), begin_loop(c4), 1)
starts_after(end_loop(c2), end_loop(c4), 0)
starts_after(Crossing the runway, end_loop(c2), 8)
starts_after(end_or(or1), Crossing the runway, 0)
starts_after(end_loop(c1), end_or(or1), 0)
starts_after(end_and(and0), end_loop(c1), 0)
starts_after(Request for the clearance to take off, end_and(and0), 1)
starts_after(begin_loop(c3), Request for the clearance to take off, 0)
loop_cond(c3, “decision position and hold”)
loop_max(c3, 4)
starts_after(Decision making concerning the request for the clearance to take off, begin_loop(c3), 0)
starts_after(Readback of the take off instructions, Decision making concerning the request for the clearance to take off, 1)
starts_after(begin_loop(c6), Readback of the take off instructions, 0)
loop_cond(c6, “runway is occupied”)
loop_max(c6, 1000)
starts_after(end_loop(c6), begin_loop(c6), 1)
starts_after(end_loop(c3), end_loop(c6), 0)
starts_after(Take off, end_loop(c3), 8)
starts_after(End, Take off, 0)

Two another workflow examples that describe the execution of the incident reporting and investigation task are depicted in Figures 14 and 15.

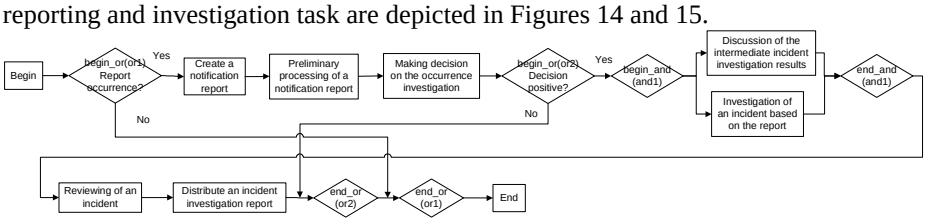


Fig. 14. The workflow that defines the execution of the incident reporting task initiated by a controller

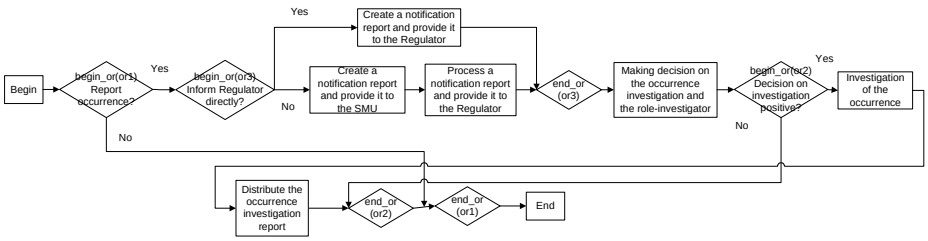


Fig. 15. The workflow that defines the execution of the incident reporting task initiated by a crew

One more flow of control describes the process of the development and the implementation of a new operation.

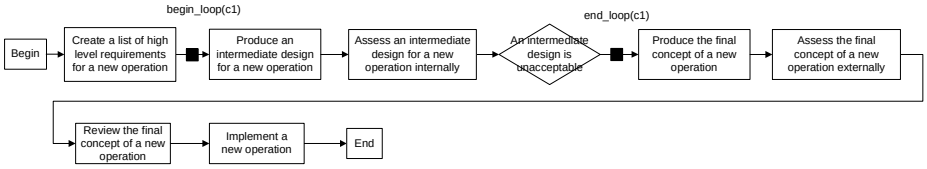


Fig. 16. The flow of control that describes the process of the development and the implementation of a new operation

Step 9. The specification of the characteristics and behavior of agents, and the agent allocation principles.

The performance variability in an organization is represented by specifications of agents that are allocated to organizational roles. Agents are characterized by capabilities and traits. A prerequisite for the allocation of an agent to a role is the existence of a mapping between the capabilities and traits of the agent and the role requirements.

For the case study a number of agent types have been identified: Controller, Pilot, Safety Investigator, Operation Designer, Operation Analyst, and Manager. Then, particular agents of these types with varying capabilities and traits have been defined. For example, the agent allocated to the Tower Controllers Supervisor role in comparison to the agent allocated to some Runway Controller role has a higher level of development of the technical and interpersonal skills and additionally has the developed managerial skills.

The behavior of an agent is considered as goal-driven. In this case study only the goals of agents that are in line with the organizational goals are taken into account. Furthermore, the internal states of agents allocated to organizational roles are represented as beliefs. A belief of an agent is created based on:

(a) an observation from the environment;
(b) a communication provided to/obtained from another agent;
(c) an action performed by the agent in the environment.

Consider an example of the generation of belief state of the agent allocated to the role Crew Representative based on the communication from the agent allocated to the Aircraft's Controller role.

$\forall\gamma$: TRACE $\forall$content_var:CONTENT $\forall$t1:TIME holds(state(γ, t1, input(Crew Representative)), communication_from_to(Aircraft's Controller, Crew Representative, inform, content_var))) $\Rightarrow$ holds(state(γ, t1+1, internal(Crew Representative)), belief(informed_by, Aircraft's Controller, content_var))

For each belief state a belief persistency rule is specified. The following rule specifies the persistency of the belief of the agent allocated to the Crew Representative role about the clearance to cross the runway12:

$\forall$ct_var: MSG_TYPE $\forall$content_var:CONTENT $\forall$t1:TIME
holds(state(γ, t1, internal(Crew_Representative)), belief(decision_provided_by, Aircraft's Controller, instruction(clearance_to_cross, runway12)) & $\exists$t3: TIME t1 < t3 +119 &

holds(state(γ, t3, input(Crew_Representative)), communication_from_to(Aircraft's Controller, Crew Representative , decision, instruction(clearance_to_cross, runway12))) &
& ¬∃t4 > t3 holds(state(γ, t4, internal(Crew Representative)), belief(performed, Crew, action(cross_runway, runway12)))
⇒ holds(state(γ, t1+1, internal(Crew_Representative)), belief(decision_provided_by, Aircraft's Controller, instruction(clearance_to_cross, runway12))

Informally this rule specifies that the belief will persist for two minutes at most upon the receipt of the clearance to cross the runway and will be ceased immediately after the aircraft's crossing action.

Step 10. The identification of the generic and domain-specific organizational constraints

At this step a set of the identified constraints is specified. First, some of the constraints obtained from the regulations related to the functioning of a crew are given:

Constraint 1.
Each instruction of the controller guiding an aircraft provided to the crew of the aircraft should be read back by one of the pilots within 5 minutes.
Formally:
∀γ: TRACE ∀ct_var: MSG_TYPE ∀content_var:CONTENT ∀t1:TIME holds(state(γ, t1, input(Crew Representative)), communication_from_to(Aircraft's Controller, Crew_Representative, ct_var, content_var)) ⇒ ∃t2 ≤ t1+5 holds(state(γ, t2, output(Crew Representative)), communication_from_to(Crew Representative, Aircraft's Controller, readback, content_var))

Constraint 2.
Before performing the crossing or taking off pilots must visually check for conflicting traffic regardless of clearance.
Formally:
∀γ: TRACE ∀content_var:CONTENT ∀t1:TIME ∀clear_var:CLEARANCE ∀runway_var: RUNWAY
holds(state(γ, t1, internal(Crew)), belief(decision_provided_by, Aircraft's Controller, instruction(clear_var, runway_var)) ∧ belief(observed, env, runway(runway_var, clear)) ⇒ holds(state(γ, t1+1, output(Crew)), begin_execution(cross_runway(runway_var)))

Constraint 3.
Both pilots should monitor the frequency when a clearance is called for to ensure that both pilots hear the taxi clearance.
Formally:
∀γ: TRACE ∀t1, t2:TIME t2 > t1 ∀clear_var:CLEARANCE ∀runway_var: RUNWAY ∀instr_var: INSTRUCTION_TYPE ∀ag1, ag2: AGENT
holds(state(γ, t1, output(Crew Representative)), communication_from_to(Crew Representative, Aircraft's Controller, request, instruction(clear_var, runway_var))) &
holds(state(γ, t2, input(Crew Representative)), communication_from_to(Aircraft's Controller, Crew Representative, decision, instruction(instr_var, runway_var))) &
holds(state(γ, t1, env, agent_plays_role(ag1, Pilot in Command) ∧ agent_plays_role(ag2, Second Pilot)) &
⇒ ∀t3 t2 > t3 > t1 holds(state(γ, t3, env), agent_plays_role(ag1, Crew Representative) ∧ agent_plays_role(ag2, Crew Representative))

Constraint 4.
When an aircraft is approaching to an active runway, the pilots should cease all processes not related to the taxiing.
Formally:
∀γ: TRACE ∀t1, t2: TIME t2 > t1 ∀acraft_var:AIRCRAFT ∀clear_var:CLEARANCE ∀runway_var: RUNWAY ∀ag1, ag2: AGENT ∀p1: TAXIING_AIRCRAFT ∀r1: ROLE
holds(state(γ, t1, env), aircraft_approaches_to(acraft_var, runway_var) ∧ agent_plays_role(ag1, Pilot in Command) ∧ agent_plays_role(ag2, Second Pilot)) ⇒
holds(state(γ, t1, env), process_execution(p1, Pilot in Command) ∧ process_execution(p1, Second Pilot) ∧ ¬∃p2: PROCESS p2≠p1 process_execution(p2, r1) ∧ (agent_plays_role(ag1, r1) ∨ agent_plays_role(ag2, r1)))

Constrain 5.
In case the pilots have different beliefs about certain objects or events related to the taxiing or taking off tasks, one of the pilots should contact the controller currently guiding the aircraft for the clarification.

This rule should be specified by several constraints, one of which is the following:
If pilots have different beliefs about the received clearance, they should contact the Aircraft's Controller again.
This and the following constraints are formalized in a similar way as the previous constraints.

Constraint 6.
Each observed incident/accident should be reported by a crew.

Constraint 7.
The pilots of the crew should verbally share relevant information with each other.

Constraint 8.
Any information received or transmitted during the absence of one of the crew members should be provided to him/her upon his/her return.

Constraint 9.
If a pilot is revealed/reveals himself/herself from the allocation to the role Crew Representative that constantly monitors the frequency of the Aircraft's Controller role, then this pilot should inform all other crew members about this.

In the following consider some of the constraints obtained from the regulations related to the functioning of a controller:

Constraint 10.
A shift of a controller consists of three sessions. The duration of each session is 1 hour. After each session the obligatory break follows, which lasts for 1 hour.
Formally:
∀r1:CONTROLLER ∀a1:AGENT ∀t1:TIME ∀proc1:PROCESS holds(state(γ, t1, env), is_responsible_for(r1, execution, proc1) ∧ is_instance_of(proc1, control_region) ∧ agent_allocated(ag1, r1)) ⇒
[holds(state(γ, t1+3600, env), agent_released_from(ag1, r1)) ∧

[sum([t2: TIME], case(holds(state(γ, t2, env), agent_allocated(ag1, r1)), 1, 0) < 10800 ⇒ holds(state(γ, t1+7200, env), agent_allocated(ag1, r1))]]

Constraint 11.
A controller may guide maximum two aircrafts at the same time.
Formally:
∀r1, r2: CONTROLLER ∀r3, r4: CREW ∀a1:AGENT ∀t1:TIME ∀p1, p2:PROCESS
holds(state(γ, t1, env), [process_monitoring(p1, r1) ∧ process_monitoring(p2, r2) ∧ process_execution(p1, r3) ∧ process_execution(p2, r4) ∧ p1≠p2 ∧ agent_allocated(ag1, r1) ∧ agent_allocated(ag1, r2)] ⇒
[[∀p3:PROCESS ∀r5:CONTROLLER ∀r6:CREW process_monitoring(p3, r5) ∧ agent_allocated(ag1, r5) ∧ process_execution(p3, r6)] ⇒ [p3 = p1 ∨ p3 = p2]])

Constraint 12.
A controller is not allowed to issue any new clearances for some runway until this runway is vacated by the aircraft that had received the last clearance from the controller.

Constraint 13.
As soon as a runway is vacated and some aircraft(s) is (are) waiting for the clearance for this runway, the controller responsible for the runway should provide a clearance to one of the waiting aircrafts.

Constraint 14.
If a controller cannot reach an aircraft taxiing in the sector for which this controller is responsible, s/he should contact the controller of the sector from which the aircraft came.

Constraint 15.
Each observed incident/accident should be reported by a controller.

One of the constraints for the tower controllers supervisor is the following:

Constraint 16.
Perform the allocation of agents-controllers to the aircraft monitoring processes in such a way that the number of processes executed at the same time by each controller is less than two.

Several constraints describe the required common allocations of agents:

Constraint 17.
At any time point one of the roles of type CONTROLLERS_SUPERVISOR should have the common allocation with the role Tower Controllers Supervisor.
Formally:
∀γ:TRACE ∀t1:TIME ∃ag1: AGENT ∃r1: CONTROLLERS_SUPERVISOR holds(state(γ, t1, env), agent_plays_role(ag1, r1) ∧ agent_plays_role(ag1, Tower Controllers Supervisor))

Constraint 18.

At any time point the role Aircraft's Controller should have the common allocation with one of the roles of the type CONTROLLER.

Constraint 19.
At any time point the role Crew Representative should have the common allocation with one of the roles of the type Pilot.

Constraint 20.
At any time point the role Head ODU should have the common allocation with the role Team Member (ODU).

Constraint 21.
At any time point the role Head of Controllers should have the common allocation with the role Team Member (CSU).

Constraint 22.
At any time point the role Airport design representative of the role NODT should have the common allocation with one of the roles of type Airport Operation Designer.

Constraint 23.
At any time point the role ANSP design representative of the role NODT should have the common allocation with one of the roles of type Operation Designer.

Furthermore, one more general constraint for the allocation of agents to any role is defined:

Constraint 24.
A prerequisite for the allocation of an agent to a role is the existence of a mapping between the capabilities and traits of the agent and the role requirements.

4 Analysis results

In this Section first the results of the correctness verification of the specifications of the particular views and across different views are presented (Section 4.1). Then, the analysis by simulation of a combined specification with agents allocated to the roles is described (Section 4.2).

4.1 Correctness verification

First, the correctness of the goal and PI specifications for the performance-oriented view has been checked. More specifically, the generic structural integrity and consistency constraints defined in the performance-oriented view have been verified on these specifications. No inconsistencies have been identified in the considered PI structure. At the same time the automated checking of the consistency of the considered goal structure has identified a number of (potential) conflicts:

(a) between the goals 9 and 3;
(b) between the goals 9 and 7;

(c) between the goals 10.3 and 4.4;
(d) between the goals 11.2 and 4.4;
(e) between the goals 20 and 10.1;
(f) between the goals 4.1 and 4.2;
(g) between the goals 18.3 and 18.4;
(h) between the goals 23.1 and 10.1.

The goals that are in conflict cannot be satisfied (satisficed) at the same time. To resolve a goal conflict one of the solutions proposed in the performance-oriented view in Part III may be used. To this end the characteristics of goals such as the priority, ownership and the negotiability may be used.

Then, the correctness of the specifications for the process-oriented view has been checked. The identified structures of resources and tasks are correct with respect to the generic constraints identified in the process-oriented view. However, the following generic constraint defined over both the process-oriented and the performance-oriented views is not satisfied by the task specification: each organizational goal should be related to at least one organizational task and each task should contribute to the satisfaction of at least one organizational goal. The automated analysis showed that the goal 4.4 "It is required to achieve great involvement of the experts (e.g., controllers) possessing knowledge in the domain of the operation in the process of operation design" does not have the corresponding task. In reality it appeared that this goal is satisfied in an ad-hoc way, without following any particular procedures or regulations. Since this goal is in conflict with two other organizational goals 10.3 and 11.2, the satisfaction means of this goal directly influence the frequency of the occurrence of inconsistencies and performance drawbacks in the organizational operation. This problem type is called a latent flaw.

The tasks that contribute to the satisfaction of the goals 10.4, 10.5, 10.6 and 23.3 are not considered in this case study and it is assumed that these goals are always satisfied.

The automated analysis showed that the constructed specifications of the flows of control depicted in Fig. 13 are correct with respect to the generic constraints defined in the process-oriented view. Furthermore, the defined process-oriented specifications satisfy all the relevant domain-specific constraints defined on the step 10: more specifically, the constraints 1 and 6 defined on the crew-related processes and the constraints 12, 13 and 15 defined for the controller-related processes. Establishing the satisfaction of all other identified constraints is performed on actual executions of organizational scenarios (traces).

The structural integrity of the specifications for the organization-oriented view that describe interactions between the roles is established using the constraints defined in the organization-oriented view. Furthermore, the interaction relations between the roles defined by the constraints identified at the step 10 should be also represented in the specification. The automatic verification showed that the constructed specifications that describe the relations between the roles identified in the case study are correct.

The responsibility relations of the roles on different aspects of the identified tasks are also defined in an unambiguous way: For each aspect of each identified task the responsible role(s) is (are) assigned. It is assumed that the requirements for roles

reflect reliably the capabilities of agents required for the effective and efficient accomplishment of the corresponding tasks.

The formal authority system of the considered organization combines features of mechanistic and organic organizations. In particular, the division of tasks and decision making within the crew is performed by common agreement of all members. This also contributes to the satisfaction of the goal 14. Furthermore, although a controller guides an aircraft, s/he is not given the formal power on its crew. Also, the way of working of the NODT is highly informal. Nevertheless, the superior-subordinate relations are defined between the Design Team Manager role and other members of the NODT. Note that because of the common allocations defined by the constraints at the step 10 almost all members of the NODT have at least two superior roles, one of which is the Design Team Manager. To ensure the effective and efficient work of the agents allocated to the roles-members of the NODT, their superiors should jointly coordinate the allocation of tasks to these agents. For the roles ODU, SIU, TCU, OAU and AODU the authority relations are defined that are inherent in mechanistic organizations. No inconsistencies or ambiguities have been identified in the authority structures of these roles.

Note that besides the execution of tasks the formal authority relations of the organization influence the satisfaction of organizational goals. For example, to achieve the satisfaction of the goal 20 the crews and the controllers should be provided sufficient decision making power with respect to their tasks. In particular, according to Table 9 a crew is responsible for making both technological and managerial decisions with respect to the tasks such as “the inquiry for the clearance for crossing an active runway”, “crossing a runway”, “take off”, whereas a controller has the formal power to decide about the provision of clearances to crews.

To ensure that the constraint 24 is always satisfied the organization should have a sufficient number of properly trained agents to be allocated to all the roles of the organization. The analysis based on the amount of agents currently employed by the organization that satisfy the requirements for the Ground and Runway Controllers roles identified many situations in which the same agent-controller should be allocated to more than two aircraft’s guiding processes (which violates the constraint 16). In reality such situations are not rare when the amount of traffic increases. An example of such a situation is when more than two aircrafts approached to a runway in the same time frame and the Tower Controllers Supervisor could not assign the guidance of any of these aircrafts to some other agent-controller (e.g., because of their high workload). The interviews showed that in reality such situations are resolved by violating the constraint 16, thus, sacrificing the satisfaction of the organizational goals 10, and 24. Also, it is reported that sometimes the management of the organization to keep the satisfaction of the constraint 16, violates the constraint 24, by allocating not (completely) qualified agents to the controller roles, thus, causing the dissatisfaction of the goal 10. Obviously, the satisfaction of the important organizational goal 10 related to safety is sacrificed in both solutions. The lack of the consideration for the safety-related goals may cause incidents or even accidents. Therefore, problems that create obstacles to the satisfaction of such goals should be closely investigated by the organization. A possible solution for the described problem could be the provision of training for agents that are not (completely) qualified for the controllers roles or the

employment of the agents that already conform to the requirements for the controllers roles.

4.2 Analysis by simulation

In reality the behavior of agents allocated to the organizational roles may diverge (to a substantial extent) from the predefined description of the formal organization (i.e., the specifications of the views and the related constraints). An agent may intentionally or unintentionally deviate from the prescribed behavior. Such deviations may affect the organizational performance both in a positive and in a negative way.

In this Section the simulation results of two cases are described: in the first case the movement of aircrafts on the ground is simulated (Section 4.2.1), whereas the simulation for the second case concerns the formal and informal incident reporting and investigation paths (Section 4.2.2). In Section 4.2.1 it is demonstrated how agent deviations may affect the satisfaction of the organizational goals in a negative way. Then, Section 4.2.2 introduces an informal way of incident reporting that, as follows from the simulation results, may allows identifying organizational safety-related problems faster than through the formal incident reporting and investigation ways, thus positively contributing to the satisfaction of the organizational safety-related goals.

4.2.1 Simulation of the movement of aircrafts on the ground

Empirical studies in the area of air control have shown that among all causes of the agent's unintentional deviations during the execution of aircraft taxiing operations, the agent's incorrect situation awareness is the most frequent one. Among other common causes are: technical errors of the execution of processes, unexpected environmental circumstances not foreseen by the specification of the formal organization.

Situation awareness is the agent's mental representation of objects and events that exist/occur in his/her environment, modeled by a set of beliefs. The incorrect situation awareness of an agent may result from:

(a) some communication problem;
(b) misunderstanding and/or misinterpretation of information provided;
(c) forgetting of information.

Among the common communication problems are the following:

(a) required information is not provided;
(b) required information is sent, but not received;
(c) incorrect information is provided;
(d) partial information is provided;
(e) correct information is provided to a wrong recipient;
(f) required information provided untimely.

A number of studies have been performed in the area of air control [2, 3, 6, 7], which by applying statistics to a large corpus of empirical data identified the probabilities of different types of errors and deviations of agents involved into the task of movement of an aircraft on the ground. In the following a simulation case of the execution of taxiing and take off processes in the ATCO will be described that

uses some probability values from these studies. The configuration of the organization used in the case is taken from the ATCO reports. In the following the background of the case is described.

Based on the joint decision of the Airport's Management, of the ANSP and of the largest airlines a new operation has been developed, evaluated and implemented – a new runway has been introduced. Due to the physical position of the new runway, most of the aircrafts taxiing to other runways designated for take off need to cross this new runway. The new runway can be crossed at one place only, whereas may be approached using two taxiways situated in two different sectors of the airport. Thus, the focus of this study is on the processes performed on the new runway (runway1) and in two adjoining sectors (sector 1 and sector2). In normal conditions one controller is responsible for each region (i.e., a sector or a runway).

The purpose of this study is to investigate the safety issues that may be caused by the introduction of the runway1. We shall investigate the organizational behavior in the normal configuration, when the number of the aircrafts guided by each controller is less than 3 and in a critical configuration, when the number of aircrafts increases significantly and the constraint 16 cannot be satisfied for some controllers.

The number of agents-controllers is limited to four. One of these agents is always allocated to the Tower Controllers Supervisor role. This agent sees to the satisfaction of the constraint 16, reallocating three other agents-controllers to the ground and runway controllers roles depending on their workload. It is assumed that the agents are properly qualified for their roles.

Generally, the agents allocated to the organizational roles behave as it is specified by the formal organizational specification. The differences with the predefined organizational scenarios are caused by agent deviations, the probabilities for some of which and for other events related to the functioning of the agents are given in Table 11.

Table 11. The probability values of the events involving controllers and pilots that deviate from the formal organizational specification

Event	Probability value
(1) A crew recognizes a runway as a taxiway (wrong situation awareness)	normal visibility conditions: 3.5e-5 low visibility conditions: 2e-3
(2) A controller forgets about an aircraft scheduled to wait for a clearance	$0.001 * 2^n$, if $0.001 * 2^n < 1$ 1, otherwise
(3) A controller forgets to inform the crew to change the frequency	$0.001 * 2^n$, if $0.0005 * 2^n < 1$ 1, otherwise
(4) A controller makes a mistake in the calculation of the separation distance between aircrafts	$1.4e\text{-}5 * 2^{2n}$, if $1.4e\text{-}5 * 2^{2n} < 1$ 1, otherwise
(5) Crew reacts to the clearance of another aircraft	1e-4
(6) A controller correctly recognizes an incident, when it occurred	for serious occurrences: 1 for less serious: 0.8

(7) A controller reports an incident, when it occurred	for serious occurrences: 1 for less serious: 0.7
(8) A pilot correctly recognizes an incident, when it occurred	for serious occurrences: 1 for less serious: 0.9
(9) A pilot reports an incident, when it occurred	for serious occurrences: 1 for less serious: 0.5

In the table n=0, when the number of the aircrafts simultaneously guided by the controller is less than 3, otherwise, n = the number of the aircrafts simultaneously guided by the controller -2.

The deviations of the agents may cause different types of incidents/accidents. The examples of the serious incidents are the following: an incursion on a runway, an occurrence when an aircraft(s) cross(es) a runway, whereas another aircraft takes off. The examples of less serious occurrences are the following: a taxiing aircraft makes a wrong turn and progresses towards the runway crossing, a taxiing aircraft switches to a wrong frequency, a taxiing aircraft initiates crossing due to misunderstanding in communication.

In this case study we shall focus on serious occurrences, namely on situations when two aircrafts are situated on a runway at the same time. Based on such type of incidents incident reports will be produced with a high probability. In Table 11 the events (1), (3), (4) and (5) (or their combinations) have direct effect on the incursion probability. As it follows from the table, an incursion on a runway is most probable in the conditions of the low visibility and of a high amount of traffic that causes the overload of controllers managing this traffic. For example, the probability that an aircraft in low visibility conditions enters a runway without receiving the clearance due to the coincidence of two events (1) the crew mistakenly recognized the runway as a taxiway, and (3) the responsible ground controller forgot to inform the crew about the frequency change because of the high workload (4 aircrafts simultaneously guided) is 0.002 * 0.004 = 0.000008. Notice that the occurrence of the event (1) without (3) will unlikely lead to an incident. In such a situation, a communication channel between the crew and the runway controller is established before the aircraft enters the runway. During the first contact with the crew, the runway controller will correct the wrong situation awareness of the crew. Also, the probability that the event (3) without the event (1) will cause an incursion is very low. However, although such and other events (e.g. (2)) may not lead to an incident, still they influence the organizational performance. A decrease in performance is caused by unnecessary delays and time required for the problem solving. However, in this study only safety issues are considered.

The probability of an incursion occurrence in the normal configuration under low visibility conditions can be estimated analytically: 0.002* 0.001 + 0.000014+ 0.0001 = 0.000116. In the normal visibility conditions is it even lower.

A more serious threat for safety represent critical configurations. From Table 11 it is obvious that the higher is the workload of the controllers (of the runway controller in particular) the higher the probability of an incursion. Due to the physical location of the new runway, the growth of the workload on the controller of the runway1 is

proportional to the traffic growth. For example, for the configuration, in which each ground controller guides four aircrafts, and the runway controller manages five aircrafts in low visibility conditions, the probability of an incursion is approximately 0.001.

For the simulation a critical configuration with six aircrafts is used. Four aircrafts approach to the runway1 from the sector 1, and two aircrafts approach to the runway1 from the sector 2. The time points at which the aircrafts appear in the sectors 1 and 2 are uniformly distributed within 7 minutes. For two of the aircrafts taxiing in the sector 1and for one aircraft taxiing in the sector 2 the runway1 was designated for take off. For the rest of the aircrafts the runway1 was situated on the way to their designated runway.

Since the actual probabilities of the events given in Table 11 are very low and would require many (thousands of) simulations to draw some plausible conclusions, the probability values used in this case study are obtained from the original ones by the multiplication by the factor 100.

The structural and behavioral formal organizational specification extended with the probability values of the events and the environmental conditions defined above has been specified in the simulation environment Leadsto [4]. Then, one hundred simulations have been performed and the obtained simulation traces have been analyzed using the checking environment [5].

In particular, by the analysis the following results have been obtained:

(1) The agent allocated to the Controller Runway1 role in all traces most of the time was guiding at least four aircrafts.

Formally it means that the following property holds for any simulation trace γ:

$\forall$n:INTEGER n$\neq$4 & n$\geq$0 sum([t:TIME], case(holds(state(γ, t, env), agent_workload(ag_controllerB, n), true), 1, 0)) < sum([t:TIME], case(holds(state(γ, t, env), agent_workload(ag_controllerB, 4), true), 1, 0))

Here ag_controllerB is the agent-controller allocated to the Controller Runway1 role.

(2) Also, the ground controller of the sector 1 (agent ag_controllerA) was overloaded, guiding in average three aircrafts at the same time.

(3) The incursion event on the runway1 occurred in 36 traces from 100.

Formally:

sum([γ:TRACE], case($\exists$t:TIME $\exists$crew1_var:CREW $\exists$crew2_var:CREW holds(state(γ, t, env), incursion_at_between(runway1, crew1_var, crew2_var), true), 1, 0)) = 36

Therefore, the relative frequency of the incursion on the runway1 in the described configuration is 0.36.

(4) The number of incursions caused by the combination of the events (1) and (3) from Table 11 is 30.

Formally:

prop1(γ:TRACE) is defined as

$\exists$t1, t3: TIME t3<t1 $\exists$crew1_var, crew2_var, crew_rep1_var, crew_rep2_var: CREW $\exists$freq1:FREQUENCY $\exists$taxiway_var:TAXIWAY $\exists$acraft1_var, acraft2_var: AIRCRAFT

holds(state(γ, t1, env), incursion_at_between(runway1, crew1_var, crew2_var) $\wedge$ controller_frequency(controller runway1, freq1) $\wedge$ crew_repr(crew1_var, crew_rep1_var) $\wedge$ crew_repr(crew2_var, crew_rep2_var) $\wedge$ crew_of_aircraft(crew1_var, acraft1_var) $\wedge$ crew_of_aircraft(crew2_var, acraft2_var)) &

$\forall$t2:TIME $\forall$controller1_var:CONTROLLER t2 < t1

((holds(state(γ, t2, input(crew_rep1_var)), ¬communication_from_to(controller1_var, crew_rep1_var, inform, change_frequency(freq1))) & holds(state(γ, t3, internal(crew_rep1_var)), belief(observed, env, aircraft_close_to(acraft1_var, taxiway_var))) & holds(state(γ, t3, env), aircraft_approaches(acraft1_var, runway1)))) ∨
((holds(state(γ, t2, input(crew_rep2_var)), ¬communication_from_to(controller1_var, crew_rep2_var, inform, change_frequency(freq1))) & holds(state(γ, t3, internal(crew_rep2_var)), belief(observed, env, aircraft_close_to(acraft2_var, taxiway_var))) & holds(state(γ, t3, env), aircraft_approaches(acraft2_var, runway1)))))

Then, the following property holds:
sum([γ:TRACE], case(prop1(γ)), true), 1, 0)) = 30

(5) The number of incursions caused by mistakes of the runway controller in the calculation of the separation distance between aircrafts is 5.
Formally:
sum([γ:TRACE], case(∃t:TIME ∃crew_rep1_var, crew_rep2_var:CREW ∃clear1_var, clear2_var: CLEARANCE ∃controller1_var, controller2_var: CONTROLLER ∃ag1_var:AGENT crew_rep2_var ≠ crew_rep1_var holds(state(γ, t, env), clearance_provided(clear1_var, runway1, controller1_var, crew_rep1_var)) ∧ clearance_provided(clear2_var, runway1, controller2_var, crew_rep2_var)) ∧ agent_plays_role(ag1_var, controller_runway1) ∧ agent_plays_role(ag1_var, controller1_var) ∧ agent_plays_role(ag1_var, controller2_var)), true), 1, 0)) = 5

(6) There is only one incursion caused by a crew mistakenly reacting to the clearance for some other crew.

Although the increased probability values have been used this case study for simulations, still the analysis results point at a significant risk that exist, when an active runway is situated on the way to other runways. In this case the safety reevaluation of the new operation (i.e., the new runway introduction) is required.

4.2.2 Simulation of formal and informal incident reporting paths

Serious incidents (e.g., runway incursions) investigated in the simulation case study in the previous Section usually lead to an immediate investigation using the formal incident reporting paths identified in Section 3. However, more often less important events related to the operation of aircrafts on the ground occur that are less likely to lead to an immediate investigation, however, are often registered for further investigation. Examples of such events are the following: taxiing aircraft stops progressing on the runway crossing only after the stopbar and due to a call by the runway controller; taxiing aircraft makes wrong turn and progresses towards the runway crossing; taxiing aircraft makes wrong turn and progresses on a wrong taxiing route that is not a runway crossing; taxiing aircraft has switched to a wrong frequency. When a significant number of such events have been accumulated, an investigation may begin. Often accumulation of data about such occurrences takes long time. Also, the process of incident reporting and investigation is time consuming. One of the main reasons for this is that several roles from different organizations are involved in the formal incident reporting processes. Quite often interaction between organizations operating based on different norms and regulations creates time delays. Furthermore, incident investigation is a complex process that requires good analysis

skills, experience and ingenuity. Time delays, interaction inefficiencies between organizations and a low quality of the incident investigation may cause a late identification of safety-related problems in the operations of the ATCO that may result into incidents.

From the practical experience it is known that next to the formal incident reporting paths also informal incident reporting paths exist in the ATCO. In particular, examples are known when based on discussions among tower controllers, potential safety problems were informed to the management of the ANSP. This sometimes resulted into a much faster identification of safety problems related to aircraft movement operations. Such an incident communication path is not specified by any organizational documents and is initiated by agents-controllers themselves. More specifically, this path consists of the following steps:

1. Tower Controllers (including their supervisor) discuss among themselves during the breaks occurrences that they observed during their shifts.
2. If a potential (important) safety issue is identified during such discussions, the information about this issue will be further provided to the Head of Controllers, who is also the member of the Operation Management Team (OMT).
3. The provided information is discussed in the OMT and may support the ATC Executive Management in their understanding of the vent and resulting decision-making.

The aim of this simulation case is to investigate the path of informal incident reporting, its influence on the general organizational performance and to compare the consequences of both formal and informal incident reporting.

This case is performed in the context of the previous case described in Section 4.2.1 with the difference that 7 agent controllers are used in this simulation instead of 4. In Table 12 the aggregated skills and their development levels are presented for the considered agents controllers. All the agents-controllers possess the aggregated air traffic control skill (atc), which allows them to be assigned either to runway or ground controllers roles. The agent ag_controllerG also possesses the skill “employee management”, which allows allocating this agent to the role Tower Controllers Supervisor.

Table 12. The agents-controllers considered in the simulation case study.

Agent Controller	Skill (the level of development)	Influence level
ag_controllerA	atc (2)	0.3
ag_controllerB	atc (3)	0.6
ag_controllerC	atc (2)	0.3
ag_controllerD	atc (4)	1
ag_controllerE	atc (3)	0.6
ag_controllerF	atc (4)	1
ag_controllerG	employee management(4) atc (4)	1

Furthermore, the development level of the skills related to air traffic control forms the basis for influence (i.e., informal power) in the ANSP. The influence levels of the

considered agents are given in Table 12. The higher the skill development level of an agent-controller, the more influence this agent has in the ANSP organization. The level of influence of an agent-controller plays an important role in the propagation of information about a potential safety problem to the management level of the organization.

In the simulation the work management of controllers is specified according to the constraints identified in Section 3. A traffic flow in the surrounding of the runway1 is assumed to be 30 aircraft per hour, 12 hours per day. Table 13 presents the probabilities values for some events that may occur during the execution of taxiing and taking off processes based on the results of the empirical studies.

Table 13. The probabilities values for some events that may occur during the execution of taxiing and taking off processes.

Event	Event probability (per taxi operation)
(a) Aircraft rejects take-off as result of a runway incursion	5e-6
(b) Taxiing aircraft stops progressing on the runway crossing only after the stopbar and due to a call by the runway controller	2e-5
(c) Taxiing aircraft makes wrong turn and progresses towards the runway crossing	1e-4
(d) Taxiing aircraft makes wrong turn and progresses on a wrong taxiing route that is not a runway crossing	2e-4
(e) Taxiing aircraft has switched to a wrong frequency	1e-3
(f) Taxiing aircraft initiates to cross due to misunderstanding in communication	1e-4

Note that some types of the identified events can be observed only by the agents allocated to particular roles in the organization. Table 14 specifies which types of events may be identified by which roles.

Table 14. The observation possibilities of the identified events by the organizational roles.

Event	Identification by			
	runway controller	ground controller	crew of a taxiing aircraft	crew of a taking-off aircraft
(a)	yes	no	yes	yes
(b)	yes	maybe	yes	maybe
(c)	yes	maybe	maybe	no
(d)	no	maybe	maybe	no
(e)	maybe	maybe	maybe	no
(f)	yes	no	maybe	maybe

However, events that occur during operations of aircrafts on the ground may not always be noticed/ correctly recognized both by crews and controllers. Furthermore, both pilots and controllers do not always register events that occur. To represent this in the simulation model, the probability values have been assigned both to the observation and registration events by both controllers and crews (see Table 15).

Table 15. The probability values for the observation and the registration of the identified types of events by controllers and crews.

Event	The probability of the correct event recognition, when it occurred		The probability of the event registration, when it observed	
	by a controller	by a crew	by a controller	by a crew
a	1	1	1	1
b	1	1	0.99	0.99
c	0.99	0.98	0.9	0.9
d	0.95	0.8	0.5	0.5
e	0.7	0.9	0.5	0.5
f	0.99	0.9	0.99	0.99

Furthermore, for this simulation case study we made assumptions regarding the number of events that are needed to formally and informally initiate a detailed investigation: N(a)=1, N(b)=1 to 5, N(c)=1 to 10, N(d)=10 to 100, N(e)=10 to 100, N(f)=1 to 10.

Also, the combined influence level of the controllers involved into the discussion (the identification of safety-related problems) contributes to the incident reporting initiation through the informal path. To represent the prerequisites for an action of reporting of an identified safety-related problem based on the occurrence type occur_type by a representative of a group of controllers, the motivation model introduced in Chapter 7 of part III of this thesis is used (see Fig. 17).

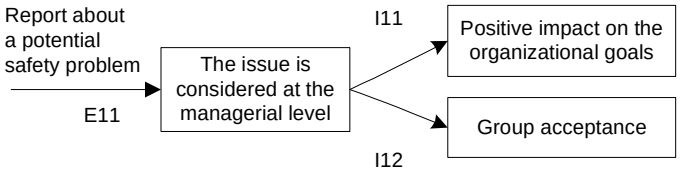


Fig. 17. The motivation model of a representative of a group of controllers for reporting about a potential safety problem

Here, both instrumentalities I11 and I12 are equal to 0.9 as the controllers involved into the discussion believe that the identified safety related issue will contribute to the satisfaction of some organizational safety-related goals and that they will be acknowledged for that. Both second-order outcomes (positive impact on the organizational goals and group acceptance) have a high level of priority for the controllers (i.e., valence value = 1).

The expectancy E11 is calculated as the product of the sum of the influence levels of the controllers involved into the discussion and the coefficient ac(occur_type), which value is dependant on the number of occurrences of the type occur_type required for the investigation (N(occur_type)) and the number of occurrences of the type occur_type observed by the controllers involved into the discussion so far ($N(occur\_type)_{curr}$):

$$ac(occur\_type) = \begin{cases} 1, & N(occur\_type) \le N(occur\_type)_{curr} \\ \dfrac{N(occur\_type)_{curr}}{N(occur\_type)}, & N(occur\_type) > N(occur\_type)_{curr} \end{cases}$$

Then, the motivation force to report about a possible problem based on the observations of events of type occur_type is calculated as it is described in Chapter 7 of Part III: F(occur_type, CD) = 1.8* ac(occur_type) * overall_influence(CD), where CD is the set of the agents-controllers involved into the discussion. If F(occur_type, CD) > 1.8, then the problem will be reported to the Head of Controllers by a controller representative (e.g., the controller with the highest influence level) of the group of the controllers involved into the discussion. After that, the problem will be discussed at the nearest OMT meeting. In this simulation it is assumed that this discussion will always result in a detailed investigation of the problem.

Based on the simulation model described above, 200 simulations have been performed with the simulation time 1 year (12 working hours per day) each. If the problem has been identified, the simulation halts. The simulation results for both formal and informal reporting cases are presented in Table 16. The mean time value of the investigation beginning with respect to some event type in the third column is calculated over all traces, in which the occurrences of events of this type caused the incident investigation. The corresponding standard deviation in the fourth column is calculated over the same set of traces.

Table 16. The results of the simulation experiments.

Event	Number of traces, in which based on the event type the investigation began		Mean time value before the recognition of a safety problem (in hours)		Standard deviation of time before the recognition of a safety problem	
	Formal	Informal	Formal	Informal	Formal	Informal
a	31	40	2093.37	1304	836.05	633
b	13	2	2523.32	1055	908.24	323.8
c	59	135	2466.69	1558	705.28	544
d	0	0	-	-	-	-
e	0	2	-	2756	-	113.1
f	86	8	2502.47	2191	638.3	612

The table shows that in all cases the path of the informal incident reporting allows a faster identification of safety-related problems than through the formal incident reporting paths.

A difference between the numbers of traces for the event type b for formal and informal incident reporting can be explained by a lower generation probability of events of this type in comparison for example to the events of the type c. Thus, since both b and c require comparable numbers of occurrences before the investigation initiation, an investigation based on events of the type c begins often sooner than the investigation based on the events of the type b. A large number of traces, in which based on the events of the type c the investigation began, can be explained by the simultaneous observation possibility of events of this type by both a ground controller and the runway controller, and by the high observation and registration probabilities of events of the type c by the controllers. On the contrary, only a small number of events of the type e is observed and registered by controllers, thus, the problem identification based on this event type is rare. A large difference between the numbers

of traces for the event type f for the formal and informal cases can be explained by the fact that only one controller is able to observe the events of this type.

5 Conclusions

The modeling and analysis of the structure and behavior of the Air Traffic Control Organization using the proposed framework proved to be practicable and useful for the understanding of the organizational functioning, for the identification of organizational errors and inconsistencies, and for the investigation of the organizational dynamics in different environmental settings. In the following this general conclusion will be substantiated.

The modeling framework allowed the identification of diverse aspects of the considered organization at a detailed level. The expressivity of the languages of the views of the framework allowed specifying all static structures and the dynamic rules of behavior defined in the ATCO.

Most of the tasks of the ATCO allow a significant degree of freedom of the execution by agents allocated to these tasks. Such tasks were specified at a high level of abstraction, whereas the important norms and regulations on these tasks were formalized as constraints. Some of these constraints can be verified on the organizational specifications, whereas others can only be checked on actual executions of the tasks by agents.

In general, the mechanism of constraints proposed by the framework allows specifying diverse aspects of both mechanistic and organic types of organizations. Constraints may be defined using the concepts and relations from multiple views. For example, the representation of the constraint expressing that the Tower Controllers Supervisor performs (re)allocation of tower controllers roles to some tasks depending on the workload situation uses the concepts and relations both from the process-oriented view (such as tasks, resources) and the organization-oriented view (such as roles, the authority relations between the Tower Controllers Supervisor and the tower controllers). The satisfaction of the constraints both of particular views and across multiple views can be determined using the efficient algorithms defined in Part III of this thesis. Using these analysis techniques in the frame of the case study the missing and the conflicting parts of the organizational description have been identified and for some problems possible solutions have been proposed. In particular, many of the conflicts identified at the level of organizational goals stem from the principal difference between performance and safety objectives of the organization. To survive and to make profit in the highly competitive environment ATCOs often strive for (a high degree of) the satisfaction (satisficing) of the performance-related goals. This often results into a decrease of the satisfaction of some safety-related goals, which may bring to an incident or even to an accident. To ensure a sufficient degree of safety, the ATCO should always ensure sufficient degrees of satisfaction of its safety-related goals.

Another analysis type described in this case study is performed on the simulation results of actual executions of organizational scenarios. In general, actual executions may diverge from the organizational scenarios defined by the formal organizational

specification. The diverging behavior of agents may influence the organizational performance and the satisfaction of the organizational goals both in a positive and in a negative way. In this case study the examples of both negative and positive influence have been demonstrated. In particular, in Section 4.2.1 it has been shown how different types of divergent agent behavior may result into delays in executions of processes and even into incidents. On the other hand, an example of the positive influence on the organizational performance of the informal incident reporting path established by agents-controllers has been described in Section 4.2.2.

Another example of the positive influence is the following. From the interviews with air traffic controllers it appeared that in low traffic conditions controllers often do not exactly follow the imposed on them prescriptions and regulations. From the controllers's experience, these deviations often lead to the increase in the organizational performance. At the same time, many of the safety-related goals are also satisfied, since the incident/accident probabilities are low in the low traffic conditions. In the future more investigations into relations between the formal and informal organizational structures and behaviors will be performed. One of the ways to perform such analysis is by simulation as it is described in this study.

Furthermore, in this case study beliefs were used for representing the internal states of agents. This choice was motivated by the assumption that the goals of agents are in line with the organizational goals and, therefore the analysis of intentional and motivational aspects of agents was not necessary. However, under different assumptions also other internal states and attitudes of agents (e.g. desires, intentions and motivation) would be useful to consider.

Some scalability considerations on the used framework can be given with respect to this case study. Many parts of the presented organizational specification are specified and can be considered at different aggregation levels: e.g., the goals, the tasks, the interaction relations on roles, the workflows. The specifications of different aggregation levels can be developed and represented separately, which decreases the complexity of modeling. At the same time, to ensure the integrity of the whole organizational specification, the relations between different aggregation levels should be identified.

Another point concerns the specification of role instances. When a generic role and its behavior are defined, the specification of any number of instances of this role requires only the definition of characteristics and behaviors that are different from those of the generic role, without providing the complete structural and behavioral specifications.

Also the analysis techniques of the proposed framework are scalable. In particular, the analysis by simulation part of this case study focused on the task of the aircraft movement on the ground, for which a detailed specification was required. At the same time, other tasks (e.g., incident reporting) were considered at a high aggregation level, without unnecessary details in the analyzed specification. Further, since the relations between the tasks, the roles and the goals are clearly defined in the complete organizational specification, only the relevant subsets of the goals and the roles were automatically chosen for the analysis of the considered task. Furthermore, a set of the relevant constraints was also chosen based on the concepts and relations specified in the analyzed specification. In such a way, the analyzed specification was automatically abstracted from all irrelevant for the considered task information. Also,

the outcomes of the analysis of the task of the aircraft movement on the ground (e.g., relative frequencies of different types of incidents) can be further used as input for the analysis of the incident reporting task. Thus, the proposed framework allows achieving the scalability of analysis.

References

1. Air navigation system safety assessment methodology. Eurocontrol. SAF.ET1.ST03.1000-MAN-01, edition 2.0 (2004)
2. Blom, H.A.P., Bakker, G.J.: Conflict probability and Incrossing probability in air traffic management. In Proceedings of the IEEE Conference on Decision and Control, Las Vegas, Nevada, December, 2421-2426 (2002)
3. Blom, H.A.P., Bakker, G.J., Everdij, M.H.C., van der Park, M.N.J.: Collision risk modelling of air traffic. In Proceedings of the European Control Conference (ECC03), Cambridge, UK, September (2003)
4. Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J.: A Language and Environment for Analysis of Dynamics by Simulation. International Journal of Artificial Intelligence Tools, 16: 435-464 (2007)
5. Bosse, T., Jonker, C.M., Meij, L. van der, Sharpanskykh, A., and Treur, J., Specification and Verification of Dynamics in Cognitive Agent Models. In: Proceedings of the Sixth International Conference on Intelligent Agent Technology, IAT'06. IEEE Computer Society Press, 2006, pp.247-255.
6. Kardes, E, Luxhoj, J.T.: A hierarchical probabilistic approach for risk assessments of an aviation safety product folio. Air Traffic Control Quarterly 13(3):279-308 (2005)
7. Stroeve, S.H., Blom, H.A.P., Bakker, G.J.: Safety risk impact analysis of an ATC runway incursion alert system, Eurocontrol Safety R&D Seminar, Barcelona, Spain, 25-27 October (2006)

Part VI

Conclusions

Chapter 1

Discussion of results

In this dissertation a number of organization modeling and analysis methods have been investigated and introduced. These methods have been structured along several dedicated modeling views on organizations and have been integrated into a general organization modeling and analysis framework. The developed framework aims at addressing current needs both of academic organization theory and of organizational practitioners for automated techniques and means that enable reliable computational analysis of complex organizations of different types. The investigation of these needs performed in the context of this work resulted into the set of high-level requirements related to different aspects of modeling and analysis of organizations that have been identified in Part I. Among them:

(1) e*xpressivity*: the modeling methods should be expressive enough to represent different structural and behavioral aspects of organizations of different types; furthermore, relations between different aspects of the organizational reality should be specified explicitly;

(2) *a strong connection to Social Science*: the meaning attached to the introduced modeling concepts and the rules of correct use of these concepts in organizational specifications should be specified based on the literature from Social Science;

(3) *automated formal analysis:* the language used for the formalization of organizational specifications should also allow manifold and rigorous automated formal analysis of these specifications (e.g., by simulation, verification and validation);

(4) *complexity*: since specifications for real organizations may be very complex, means to handle a high complexity and to increase scalability of modeling and analysis should be identified;

(5) *support for the execution of organizational scenarios*: the developed methods should allow designing organizational specifications that form a basis for enterprise information systems, which support and control the execution of organizational scenarios;
(6) *usability*: the framework based on the developed techniques should be usable and convenient for organizational practitioners: modelers, designers, analysts, managers, etc.

In the following it will be elaborated how these requirements have been realized.

Expressivity
The investigated modeling methods are positioned along a number of dedicated views (or perspectives) on organizations: the performance-oriented view, the process-oriented view, the organization-oriented view and the agent-oriented view. The concepts and relations of each view have been identified based on the literature from Social Science, on the analysis of existing dedicated organization modeling architectures and frameworks, and on the needs of practitioners. Furthermore, the concepts and relations have been chosen in such a way that the structures and dynamics of different paradigmatic types of modern organizations identified in organization theory can be represented. The formal languages of the dedicated views have both qualitative and quantitative expressivity and allow representing both structural and dynamic aspects, which is an indispensable property for modeling organizations of different types. Moreover, for each modeling concept a clear definition has been provided and the relations to other concepts have been identified.

In particular, in the performance-oriented view described in Chapters 1 and 2 of Part III organizational performance indicators and goals of different types are identified. The relations between these two concepts, which remain implicit in other frameworks, are clearly defined. Goals may be refined into more specific goals, thus forming hierarchies of goals. Performance indicators may be related by a number of relation types (causality, aggregation and correlation). Since structures of goals and performance indicators are interrelated, in most cases changes in one structure require also changes in the other structure.

Furthermore, the process-oriented view described in Chapters 3 and 4 of Part III introduces a formal language for specifying organizational flows of control and actual executions of organizational scenarios. In particular, this language enables detailed modeling of resources, including expiration durations and different modes of sharing that are distinct from other existing modeling frameworks. The language of the view allows to represent all common templates that describe the execution of processes in workflows. Moreover, since the process-oriented view is related to other organizational views, process-oriented specifications may include relations between tasks, processes, resources and other organizational concepts (e.g. roles, goals, agents).

The organization-oriented view described in Chapters 5 and 6 of Part III provides modeling means for representing both flat structures of organic organizations and hierarchical structures with any arbitrary number of aggregation levels of mechanistic organizations. Two types of structures are defined within the organization-oriented view: interaction and authority structures. In particular, for representing the authority

structure of an organization the view introduces a set of concepts and relations based on the findings from the managerial literature. It distinguishes different aspects of responsibility for the execution of processes. Using the language of the view, formal power relations that exist in organizations of different types can be represented. Furthermore, the view establishes the relations between the formal power structure and the interaction structure of an organization.

The agent-oriented view described in Chapters 7 and 8 of Part III introduces the characteristics and attitudes of agents. This view addresses both intentional (goals) and motivational attitudes of agents. Note that the framework for agent modeling described in the view is not fixed and can be easily extended by other mental attitudes of agents (e.g., desires, intentions) when needed. Furthermore, using the expressive language TTL any particular type of agent behavior can be specified.

In the developed methods special attention is given to the identification of relations between different views (i.e., between different aspects of organizational reality). In contrast to more specialized modeling frameworks (e.g., goal-oriented, process-oriented frameworks) that elaborate on particular organizational perspectives only, the proposed modeling methods allow to investigate relations between multiple perspectives on organizations and using these relations for analysis of organizations. In such a way, a better understanding of the organizational structure and behavior can be gained. In particular, the goals of an organization are achieved by the execution of organizational processes. Also, performance indicators are associated with processes, measuring some aspects of their execution. Goals are attributed to roles and agents. The formal authority relations on roles of an organization are defined with respect to tasks. Also, the relations between the role interaction structure and processes of an organization are identified. Some organizational rules are formulated over structures from all organizational views. For example, the following rule from a reward policy is formulated using concepts and relations from all organizational views: ‘when an agent allocated to some role has been keeping the values of some PI(s) related to its process above a certain threshold for some time period, then s/he will be granted some reward’.

Specifications of the views often contain the environment description. In general, no particular restrictions on the representation of the environment are imposed. Many of the environmental conditions in which an organization operates can be represented by both generic and domain-specific constraints. Furthermore, for particular purposes (e.g., for simulation of particular scenarios) the environment can be represented by a separate modeling component. In this case either an aggregated view on the environment may be taken (e.g., to represent the global behavior of markets) or a more elaborated specification of the environment may be created. In the latter case, the internal specification for the environment can be specified using one of the existing world ontologies (e.g., CYC, SUMO). It can be also defined by a set of objects with certain properties and states and with causal relations between objects.

A strong connection to Social Science

One of the central notions of the performance-oriented view - the performance indicator – plays an important role in the modern Business Performance Management [1, 5, 6]. The definitions of relations between performance indicators introduced in the performance-oriented view were inspired by the literature from this area.

Furthermore, informal goal-oriented modeling of an organization has been always one of the important topics in Social Science [10]. The existing findings in this area influenced our choice of the characteristics of goals.

Organization theory describes many ways of execution of tasks (i.e., work management) in organizations of different types [8, 10]. Using the language of the process-oriented view detailed prescriptive flows of control, of resources and of information of mechanistic organizations can be specified. An example of such a specification is given in Chapter 3 of Part III. At the same time, more vaguely and generally defined flows of organic organizations can be described by the mechanism of constraints, as it is illustrated in the case study in Part V.

Further, based on the theoretical findings from Social Science (managerial literature in particular) that describe power, duties and responsibilities of organizational positions [2, 11], a number of relations for the specification of formal authority have been identified in the organization-oriented view.

The choice of the characteristics of agents described in the agent-oriented view is based on social psychology theories from [7, 9]. In general, the specification of internal attitudes and dynamics of agents is not limited to any particular theory. Instead, a modeler is given freedom to choose a suitable theoretical basis depending on the purpose of modeling. Thus, the motivation modeling of agents presented in Chapter 7 of Part III is based on the expectancy theory (the version of Vroom) [12] that has received good empirical support. At the same time, the investigation of the relations between the environmental complexity and the behavioral complexity of agents situated in this environment is inspired by the Environmental Complexity Thesis formulated by Godfrey-Smith [4].

Automated formal analysis

The formal basis of the developed techniques provides the possibility to give precise definitions for the concepts and relations and to define rules of the correct use of these concepts and relations. Using the proposed languages the consistency and integrity constraints for the structures of each view and over multiple views have been defined. Furthermore, using the formal basis different sophisticated and rigorous types of analysis can be performed. Since the behavior of an organization can be complex, the specifications that describe dynamic aspects of an organization may contain complex temporal relations. These relations are expressed using Temporal Trace Language (TTL) in this thesis. Complex temporal specifications in TTL cannot be used for automated analysis directly, since complex temporal formulae cannot be processed by existing analysis tools. To address this issue, the thesis introduces executable sublanguages of TTL. To translate a behavioral specification of an organization in TTL into executable form, the thesis proposes an automated procedure, described in Chapter 3 of Part II. The result of such a translation – an executable specification – can be used to perform different types of analysis. The specification of organizational behavior in executable form directly can be cumbersome for complex organizations, since it would require a large quantity (including auxiliary) of executable rules. TTL proposes a much easier way of creating behavioral specifications, which still can be automatically translated into executable form.

The analysis techniques developed in this thesis can be divided into the general ones and the ones specific to a particular view.

Three general automated analysis techniques have been introduced in Part II. In particular, two of these techniques use the results of the procedure of transformation of a behavioral specification of a system in TTL into the executable format. The application of the first analysis techniques – by simulation – is demonstrated in Chapter 4 in the context of an example from the area of multi-agent systems. Chapter 5 of Part III illustrates how the second technique can be applied in organizational context, to verify consistency of an organizational specification by checking relations between dynamic properties of different aggregation levels using model checking. The same chapter illustrates the application of the third general technique – checking properties on a limited set of traces – that is described in Chapter 5 of Part II. In this example dynamic properties are checked on a formalized empirical trace obtained by executing a particular scenario with roles of the organizational specification allocated to (human) agents. One more application of the third analysis technique is described in Part V, in the context of a case study from the area of air traffic control. In this part, the checking of safety-related properties is performed on a large number of simulation traces.

As noted in Part II the introduced general techniques can be applied to dynamic systems in a great variety of domains. Several examples of the application of these techniques in the areas of multi-agent systems and cognitive science are given in Parts II and III.

However, the generality has also a reserve side. Since the general verification algorithms are not tuned to particular characteristics of a system being analyzed, the computational properties of such algorithms are in most cases worse than the ones that can be achieved by dedicated domain-specific verification.

To enable efficient analysis of specifications of different views, a number of dedicated formal analysis techniques have been developed and implemented. Using these techniques the correctness of organizational specifications with respect to different sets of constraints can be verified effectively. In particular, using one of the techniques described in the performance-oriented view (Chapter 1 of Part III), the consistency of a performance indicators structure can be established. Using the techniques described in Chapter 2 of Part III, the satisfaction of organizational goals can be automatically established, and the correspondence between goal and performance indicators structures can be determined in a semi-automated way. The process-oriented view described in Chapter 3 of Part III introduces a set of constraints of different types and provides the automated algorithms for their verification. Furthermore, this section discusses the complexity issues of the introduced algorithms. Note that because of the interconnections between the views, some of the constraints are expressed over multiple views. For checking of such constraints across views the automated algorithms are provided in Chapter 3 of part III.

Complexity

To reduce the complexity of modeling, the contributed modeling approach distinguishes four views, the specifications for each of which can be developed separately. However, in the end, the relations between the views should be identified.

Furthermore, as it is shown in Part III, to reduce the complexity of modeling and analysis, the structure and dynamics of an organization can be specified, depicted and analyzed at different aggregation levels (e.g., at the level of departments, units, teams,

groups). The specifications of each view can be created for each level separately. Then, properties related to the specifications of a particular level only can be checked without considering specifications of other levels. However, some properties may be formulated over structures of two or more aggregation levels (e.g., properties expressing relations between the performance of organizational units and the overall organizational performance; properties that ensure the consistency and integrity of the whole organizational specification). For checking such interlevel properties the general verification approach is applied, as it is described in Chapter 4 of Part II and in Chapter 5 of Part III.

Moreover, organizational processes can be considered at different levels of abstraction, as it is discussed in Chapter 3 of Part III. More specifically, using task hierarchies, specifications can be built at different levels of abstraction. General constraints defined for high level processes are refined into more specific ones that should be satisfied by processes of lower levels. In such a way, to decrease complexity, specifications of different abstraction levels can be analyzed separately keeping relations with each other through task hierarchies and the constraint refinement.

Also, the developed modeling methods allow reuse of specifications in a number of ways, which reduces the complexity of modeling. Libraries of commonly occurring parts of structures (goals and tasks hierarchies, PI-structures, workflow graphs, etc.) can be stored and reused for organizations in the same domain. The research in identifying and classifying important PIs for different domains [e.g. 1] can easily be applied here. Reuse can also be supported by predefined templates for various aspects of different types of organizations (mechanistic, organic, etc.). For example, templates for domain-specific constraints can be provided for each view to be customized by the designer. The developed tools allow defining parameterized templates (macros) for TTL formulae that can be instantiated in different ways (Chapter 3 of Part III).

Support for the execution of organizational scenarios

The developed modeling techniques provide a sufficient level of details to describe actual executions of organizational scenarios. As has been discussed in Chapters 4 and 6 of Part III organizational specifications developed using the proposed techniques may serve as a basis for automated enterprise information systems. In particular, in Chapter 4 an approach is proposed for both online and offline checking of executions of organizational scenarios (traces) with respect to formal process-oriented specifications. In such a way, the correspondence between predefined (or planned) scenarios and the actual executions of these scenarios by agents can be ensured. To guarantee an appropriate distribution of rights and responsibilities for roles and agents in an organization, and to ensure the legitimacy of the agents' actions during the execution of organizational scenarios, Chapter 6 of Part III introduces a set of constraints (i.e., axioms, execution rules) to be implemented in an enterprise information system. However, the implementation and the exploitation of an EIS built based on a complete organization model description for the automated enterprise management is out of scope of this research.

Usability

The predicate-based languages used for the formalization of the views are intuitive, close to the natural language. This facilitates the development of specifications of each view for designers, who are not familiar with formal logics. Furthermore, the graphical notations are provided for the representation of structures of each view. Currently, the graphical interface is implemented for the performance-oriented view, whereas other views are specified textually using the developed modeling tools. Using these tools parameterized templates (macros) for complex constraints can be defined that can be instantiated in different ways. Such constraints cannot be represented graphically, thus, the form of templates would be the most appropriate for designers not skilled in logics.

The developed analysis tools perform verification in a completely automated way and do not require from analysts any knowledge of the underlying verification algorithms. An exception is the process of the correspondence checking between goal and performance indicators structures described in Chapter 2 of Part III. During this process, due to the high expressivity of the language of the performance-oriented view, additional background information and the confirmation of the verification results may be requested from the analyst.

Another objective of the thesis was the identification of the methodological guidelines for the development of organizational specifications. The design steps for the development of specifications for both new and existing organizations have been introduced in Part IV. A sequence of these steps may vary depending on a particular design process. For example, the sequence of the steps that have been chosen for the case study described in Part V was one of the possible alternatives available to the designer. Further, Chapter 2 of Part IV elaborates on the design process of specifications from the organization-oriented view. Methodological guidelines for the development of specifications of other views are given in the corresponding descriptions of the views in Part III.

Although, the described methodological guidelines contain many useful modeling advices, still the designer has much freedom and choices to make during the design process. Although it is always possible to check automatically the consistency of an intermediate organizational specification at some design phase, many of the issues related to the representation of a real organization and its environment in a specification are to be solved by the designer alone. The quality of an organization design depends not only on the modeling skills of the designer, but also on the amount of knowledge available to the designer about the (formal and informal) organizational structure and processes, and of the environment in which this organization is situated. Furthermore, the analysis results of an organizational specification depend to a great extent on how plausible and complete the representation of the analyzed issue in this specification.

Also, a preliminary evaluation of the proposed organization modeling and analysis methods was one of the objectives of this dissertation. To this end, the proposed methods have been integrated into a general framework. This framework has been applied in a number of case studies from the logistics, incident management and air traffic management domains.

In particular, using the proposed methodological guidelines the specifications of the organization-oriented (Chapter 5 of Part III) and of the process-oriented views (Chapter 3 of Part III) have been created for a transport organization from the domain of logistics. Most of the processes of this organization were organized according to the principles of mechanistic organizations. The expressive power of the framework was sufficient to represent the structural and dynamic aspects of both views for this organization. Since the analyzed part of the organization was relatively small, no complexity issues emerged. The simulation-based analysis techniques proposed by the framework allowed investigating the organizational behavior in different scenarios. Furthermore, some inconsistencies in the constraint set of the process-oriented view have been identified using the analysis techniques of the framework.

One more organization that has been analyzed using the proposed framework was from the area of incident management. The analyzed part of the organization (the planning department) had many features of adhocracy (e.g., job specialization based on formal training, work organization rested on specialized teams, culture based on democratic and non-bureaucratic work). No expressivity or complexity problems have been encountered during the modeling of this organization. The dedicated analysis techniques of the performance-oriented view allowed establishing the consistency of the specifications of this view (e.g., the consistency of the goal structure and the performance indicators structure) for the considered organization (Chapter 1 and 2 of Part III). Furthermore, the mechanisms for the satisfaction of some goals of this organization have been described in Chapter 4 of Part III. Also, Chapter 4 describes how the constructed process-oriented specification of the considered organization can be used to guide and control the real time operation of the organization.

The most extensive example of the application of the proposed framework in this dissertation is provided in Part V. In this part the design steps have been consequently applied to create a specification for an air traffic control organization. This organization combines features of mechanistic and organic organizations. All the required aspects of the organization were specified using the concepts and relations of the proposed framework. The developed specification has been represented at different aggregation levels. This allowed reducing the complexity of analysis, in which a more aggregated view has been taken on roles and processes that were not directly relevant for the analysis. By applying the general and specific for particular views analysis techniques, a number of organization problems and inconsistencies have been identified. The validity of the identified faults has been confirmed by the experts and the expert knowledge existing in the air traffic management domain. Furthermore, the consequences of different types of agent behavior that diverges from the formal organization have been investigated. It is demonstrated in the case study, the identified behavioral differences may influence the organizational performance (i.e., the satisfaction of organizational goals) both in a positive and in a negative way.

From our modeling experience of organizations of different types two interesting (however, expected) observations have been made: Whereas mechanistic organizations require detailed modeling of both specifications of different views and of constraints imposed on these specifications, organic organizations are modeled to a great extent by sets of constraints that define general norms or rules imposed on the organizational structure and dynamics. Furthermore, often the modeling of organic organizations requires more extensive and sophisticated agent modeling (e.g., to

represent adaptation, reasoning processes of agents) in comparison to the modeling of agents in mechanistic organizations. These observations are also in line with the known informal assessments of the autonomy levels that agents have in organizations of both types [8, 10].

To summarize, the modeling and analysis of the structures and dynamics of different types of organizations considered in the case studies discussed above using the developed methods, proved to be practicable and useful for the understanding of the organizational functioning, for the identification of organizational errors and inconsistencies, and for the investigation of the organizational dynamics in different environmental settings.

Chapter 2

Future work

The major part of this dissertation is dedicated to the modeling and analysis of formal organizations. Behavior of agents allocated to organizational roles was considered in the context of the structures of formal organizations. However, agents themselves may form informal structures within formal organizations (e.g., social networks [3]). Such structures influence the operation of an organization and may even determine alternative ways of work management in an organization that could be formalized in the future. An agent-based simulation example of such an alternative way of work management is provided in Part V. The further investigation of informal structures of agents and their relations to formal organizations is one of the future research topics.

Also, macro level processes are of interest for the future research, in particular, different types of interactions between organizations. This research direction gains a special importance nowadays, when the interdependency of the world increases with every passing year. A preliminary study in Part V showed that many modeling principles and analysis techniques introduced in this thesis can be also applied for the investigation of processes at the macro level. However, more precise and detailed investigations should be still performed. Furthermore, it is interesting to consider reciprocal relations between different types of inter-organizational interactions at the macro level on the one side, and internal structures and processes of organizations described at the micro and meso levels on the other side. This is also left for the future research.

In Chapter 1 different ways of reuse of organizational specifications have been described. Based on these results, a set of templates for paradigmatic types of modern organizations can be defined. A template comprises partial specifications of different views and sets of constraints peculiar to organizations of a particular type. Such templates may facilitate modeling choices for designers-novices and may speed up a

design process performed by experienced designers. The development of such templates is also a part of the future work.

From the usability viewpoint, although the modeling views considered in the thesis are formalized based on intuitive, close to the natural, predicate languages, still a graphical interface would be of help. Currently, the graphical interface is provided for the performance-oriented view, whereas other views are specified textually using the dedicated tools. In the future, modeling related to other views will be also supported graphically. To enable the development of such graphical modeling tools, formal definitions for the design operators for each organizational view should be given, similarly to the ones provided for the organization-oriented view in Chapter 2 of Part IV.

Finally, further evaluation of the proposed methods has to be performed. To this end, further case studies will be performed, in which different types of organizations will be modeled and analyzed. Also, after the graphical interface is developed, the framework based on the proposed modeling and analysis methods will be made available for organization practitioners (e.g., managers, analysts) for the evaluation.

References

1. Chan, F.T.S.: Performance measurement in a supply chain. International Journal of Advanced Manufacturing Technology: 21(7): 534-548 (2003)
2. Clegg, S.R.: Frameworks of Power. London: Sage (1989)
3. Freeman, L.C.: The Development of Social Network Analysis: A Study in the Sociology of Science. Vancouver: Empirical Press (2004)
4. Godfrey-Smith, P.: Complexity and the Function of Mind in Nature. Cambridge University Press (1996)
5. Ittner, C.D., Larcker, D.F.: Coming Up Short on Nonfinancial Performance Measurement. Harvard Business Review, 81(11): 88-96 (2003)
6. Kaplan, R.S., Norton, D.P.: The balanced scorecard – measures that drive performance. Harvard Business Review, January-February: 71-79 (1992)
7. Katz, D., Kahn, R.: The social psychology of organizations. Wiley, New York (1966)
8. Mintzberg, H.: The Structuring of Organizations, Prentice Hall, Englewood Cliffs (1979)
9. Pinder, C. C.: Work motivation in organizational behavior. Upper Saddle River, NJ: Prentice-Hall (1998)
10. Scott, W.G., Mitchell, T.R., Birnbarum, P.H.: Organization theory: a structural and behavioural analysis, Richard D. Irwin inc., Illinois, USA (1981)
11. Simon, H.A.: Administrative Behavior. 2nd edn. Macmillan Co., New York (1957)
12. Vroom, V.H.: Work and motivation. Wiley, New York (1964)

Samenvatting: Over Computergesteunde Methoden voor Modelleren en Analyse van Organisaties

Organisaties spelen een sleutelrol in de moderne wereld. Snelle wetenschappelijke, sociale en technologische ontwikkelingen van de laatste eeuwen zorgden ervoor, dat er veel verschillende vormen van organisaties en typen van interacties tussen hen ontstonden. De complexiteit van de structuur en het gedrag van een organisatie hangt af van de condities van de omgeving, waar de organisatie zich in bevindt. Om leefbaar en welvarend te zijn, moeten de interne structuren en processen van een organisatie effectief en efficiënt beheerst worden, zodat de externe aanpassing van de organisatie met de omgeving in stand wordt gehouden.

Veel moderne organisaties zijn gekenmerkt door een groot aantal actoren, die diverse rollen spelen, verschillende (soms tegenstrijdige) doelen hebben en bij de uitvoering van diverse taken betrokken zijn. Vaak functioneren zulke organisaties in omgevingscondities, die steeds veranderen, waar slechts een beperkt aantal hulpmiddelen beschikbaar is. Door een hoge complexiteit van de structuur en gedrag van een moderne organisatie kunnen erin diverse fouten, inconsistenties en knelpunten ontstaan, die ernstige gevolgen voor de productiviteit van de organisatie kunnen hebben. Slechts een klein aantal van zulke problemen kan snel en simpel geïdentificeerd en opgelost worden. Veel latente problemen vergen een gedetailleerde en diepe analyse.

Methoden, die in de Sociale Wetenschappen ontwikkeld zijn (i.h.b. in Organisatietheorie), zijn in hoge mate informeel en fragmentarisch, hoewel nuttig voor het begrip van functioneren van organisaties. Daarom kunnen zulke methoden geen betrouwbare basis vormen voor een gedetailleerde analyse van organisaties. Voor een nauwkeuriger inspectie van structuren en processen van organisaties, een gedetailleerde evaluatie van de productiviteit van een organisatie, bestudering van invloeden van verschillende omgevingsfactoren op het gedrag van een organisatie zijn analysemethoden, die op een formeel organisatiemodel zijn gebaseerd, nodig. De eerste formele methoden voor analyse van organisaties zijn op basis van de Systeemdynamica en Operations Research ontwikkeld. Deze methoden abstraheren van aparte gebeurtenissen, objecten en actoren, en beschouwen organisaties op een hoog aggregatieniveau. Door de abstractie ging belangrijke informatie over locale gebeurtenissen en interacties tussen actoren verloren. Tegenwoordig onderscheidt men drie niveaus, waarop structuren en gedrag van organisaties kunnen bestudeerd worden: macro (het niveau van interacties tussen een organisatie en haar omgeving), meso (het niveau van interacties tussen actoren en/of groepen van actoren in de context van een organisatie) en micro (het niveau van een actor, haar eigenschappen en het gedrag in een organisatie). Alle drie de niveaus zijn met elkaar verbonden en beïnvloeden elkaar. Daarom is het begrip van het organisatiegedrag op ieder van de niveaus belangrijk voor een betrouwbare analyse. Tegenwoordig gebruikt men vaak de conceptie van een *multiagent systeem* voor het modelleren van het gedrag van actoren en interacties tussen hen. *Agent* is een autonoom object, dat zelfstandig beslissingen kan nemen en met haar omgeving (bijvoorbeeld, met andere agenten) interacteert door observeren en reageren. In deze context beschrijft een

organisatiemodel een toewijzing van agenten aan de *rollen* van de organisatie (verzamelingen van functies van een organisatie), die met elkaar bepaalde relaties hebben. Aan de ene kant kunnen acties van agenten invloed op het gedrag van een gehele organisatie hebben. Aan de andere kant, hebben de normen en regels van een organisatie een bepaalde invloed op het gedrag van agenten. Afhankelijk van een organisatietype, worden agenten van een bepaalde mate van autonomie voorzien. Daarom moet een formele taal voor het modelleren van organisaties mogelijkheden bieden zowel voor het beschrijven van formele voorschriften (normen, regels) en structuren van een organisatie als voor het specificeren van autonoom gedrag van agenten. De huidige aanpakken in het gebied van multiagent systemen zijn meer op het laatste aspect gefocust. Deze methoden bieden brede mogelijkheden voor het specificeren van intern en extern geobserveerd gedrag van agenten, terwijl het beschrijven van formele structuren en gedrag van organisaties niet veel aandacht krijgt. Een van de redenen daarvoor is dat het organisatieparadigma in deze aanpakken alleen gebruikt wordt om distributieve algoritmen, die op multiagent systemen gebaseerd zijn, te verbeteren.

Voor het ontwerp van plausibele modellen van reële (mens) organisaties moeten verschillende aspecten van de structuur en het gedrag van een organisatie expliciet geïdentificeerd worden. Om de complexiteit van het modelleren te verminderen, zijn diverse aspecten van organisaties in dit proefschrift vanuit vier perspectieven bestudeerd: het prestatiegerichte perspectief, het procesgerichte perspectief, het organisatiegerichte perspectief en het agentgerichte perspectief.

In het kader van het prestatiegerichte perspectief worden met elkaar verbonden structuren van doelen en prestatie-indicatoren van een organisatie en agenten onderzocht. Het procesgerichte perspectief beschrijft zowel structuren van processen en hulpmiddelen van een organisatie, als dynamische stromen van processen. In het organisatiegerichte perspectief worden de rollen van een organisatie, hun interactie- en machtsrelaties gespecificeerd. Het agentgerichte perspectief beschrijft eigenschappen en gedrag van agenten in de context van een organisatie. De perspectieven zijn door bepaalde relaties met elkaar verbonden. Bijvoorbeeld, de doelen van een organisatie kunnen door de uitvoering van processen bereikt worden; de processen zijn met de rollen verbonden, die eventueel aan agenten toegewezen worden.

De ontwikkelde formele talen voor de beschrijving van de concepten en relaties van elk perspectief zijn op de expressieve meersoortige predikaatlogica gebaseerd. Om dynamische eigenschappen te specificeren en erover te redeneren wordt de taal Temporal Trace Language (TTL) gebruikt. De formele grondslag voor deze taal (syntaxis en semantiek) is in het kader van dit proefschrift ontwikkeld. De talen van de perspectieven en TTL zijn geschikt voor het specificeren van zowel kwalitatieve, als kwantitatieve eigenschappen van een systeem (bij voorbeeld van een organisatie).

Met behulp van de voorgestelde talen worden de exacte definities voor de concepten en relaties van de perspectieven in dit proefschrift gegeven. Daarnaast zijn ook de axioma's gedefinieerd, die de regels van het correcte gebruik van de elementen van de talen en de beperkingen aan de integriteit van organisatiemodellen beschrijven. Voor de definities van deze regels en beperkingen wordt een theoretische basis vanuit de Sociale Wetenschappen gebruikt (voornamelijk vanuit de Organisatietheorie), waarmee een connectie tussen het formele logische fundament en

de resultaten van empirisch onderzoek is vastgelegd. Het proefschrift beschrijft methodologische principes voor het ontwerp van een organisatiemodel met behulp van de ontwikkelde talen.

Voor een geautomatiseerde analyse van organisatiemodellen moet de modelleringstaal uitvoerbaar (executeerbaar) zijn, d.w.z. dat specificaties van organisatiemodellen moeten door een computer kunnen worden gebruikt voor simulatie. Om dit te bereiken, beschrijft het proefschrift enige executeerbare subtalen van de TTL. Bovendien wordt in het kader van het proefschrift een geautomatiseerde procedure ontwikkeld voor de vertaling van complexe, niet uitvoerbare specificaties in TTL naar een executeerbare vorm.

Het proefschrift beschrijft een aantal geautomatiseerde analysemethoden, die tot doel hebben het identificeren van diverse fouten, inconsistenties en knelpunten van organisaties. Een deel daarvan is op algemene analysemethoden voor logische modellen gebaseerd (bijvoorbeeld, model checking). De andere hebben als basis gespecialiseerde algoritmen voor het verifiëren van bepaalde eigenschappen van organisaties (bijvoorbeeld, de consistentie van een doelenstructuur, eisen voor de executie van processen, de integriteit van interactie- en machtsstructuren). Bovendien beschrijft het proefschrift methoden voor het doen van simulaties en geautomatiseerde verificatie van systeemeigenschappen op de basis van de simulatieresultaten. Tevens wordt een soortgelijke methode ontwikkeld voor een geautomatiseerde analyse van empirische data van een organisatie.

De ontwikkelde methoden zijn in de praktijk binnen drie projecten toegepast, op het gebied van logistiek, incident management en luchtverkeersleiding. Dankzij de hoge expressiviteit van de modelleertalen konden alle belangrijke aspecten van de bestudeerde organisaties in de organisatiemodellen opgenomen worden. De ontwikkelde analysemethoden maakten het mogelijk, om problemen van diverse typen in deze organisaties te identificeren, die eerder niet bekend waren. De realiteitswaarde van de gevonden problemen wordt door de experts bevestigd.

De formele methoden voor het modelleren en analyseren van structuren en gedrag van organisaties van verschillende typen, die in dit proefschrift worden voorgesteld, bleken toepasbaar in de praktijk en bruikbaar te zijn voor het begrip van de principes van functioneren van een organisatie, voor de identificatie van diverse typen van fouten en inconsistenties, en voor de bestudering van het gedrag van een organisatie in verschillende omgevingscondities.

Резюме: Об автоматизированных методах моделирования и анализа организаций

Организации играют ключевую роль в современном мире. Стремительный научно-технический и социально-экономический прогресс последних столетий способствовал появлению большого разнообразия организационных форм и типов межорганизационного взаимодействия. Сложность структуры и поведения организации во многом зависит от условий окружающей среды, в которых организация функционирует. Жизнеспособность и процветание организации находятся в зависимости от того, насколько эффективно организация способна управлять своими внутренними структурами и процессами в соответствии с условиях окружающей среды.

Многие современные организации включают в себя большое число акторов (ролей), имеющих различные (иногда несовместимые) цели, вовлеченных в исполнение разнообразных организационных процессов. Часто такие организации находятся в постоянно изменяющихся условиях окружающей среды с ограниченным количеством ресурсов. Иногда изменения в окружающей среде являются также предпосылками к изменению структуры и/или поведения организации. Из-за высокой сложности структур и поведения современных организаций, в них часто возникают ошибки, несогласованности и проблемы узких мест, влекущие серьезные последствия для производительности организации. Только незначительная часть таких проблем может быть легко обнаружена и устранена. Многие же скрытые проблемы требует более детального и глубокого анализа.

Методы организационного анализа, разработанные в области социальных наук (в частности, в теории организаций), в значительной степени фрагментарны и неточны, хотя и полезны для понимания принципов функционирования организаций. Поэтому такие методы не могут служить надежным базисом для детального анализа организаций. Для более точного исследования организационных структур и процессов, подробной оценки организационной производительности, исследования влияния различных факторов окружающей среды на поведение организации необходимы методы анализа, основанные на формальной организационной модели. Первые формальные методы для исследования организаций были основаны на аппаратах системной динамики и исследования операций. Эти методы абстрагировались от отдельных событий, объектов и акторов организаций, тем самым рассматривая организационную динамику на обобщенном уровне. Однако при этом значительная часть полезной информации о локальных событиях и взаимодействии акторов терялась и не могла быть использована для анализа. В настоящий момент принято различать три уровня, на которых организационные структуры и процессы могут быть исследованы: макро (уровень взаимодействия организации с окружающей ее средой), мезо (уровень взаимодействия акторов и/или групп акторов в организационном контексте) и микро (уровень отдельного актора, его характеристики и поведение в организации). Все три уровня находятся во взаимосвязи друг с другом, поэтому

понимание поведения организации на каждом из этих уровней важно для достоверного анализа. Сейчас для моделирования поведения акторов и взаимодействий между ними часто применяют концепцию мультиагентных систем. Агент представляет собой автономный объект, способный самостоятельно принимать решения и взаимодействовать с окружающей средой (например, с другими агентами) путем наблюдений и действий. В данном контексте модель организации описывает назначение акторов (или агентов) позициям (или ролям) организации, описываемых множествами организационных функций и находящихся в определенных отношениях друг с другом. С одной стороны, действия отдельных агентов способны оказывать влияние на поведение организации в целом. С другой стороны, правила и нормы организации влияют на поведение агентов. В зависимости от типа организации, агенты предоставляется различная степень свободы. Поэтому необходимо, чтобы формальный язык для описания организационных моделей предоставлял возможность представления как формальных предписаний (норм, правил) и структур организации, так и автономного поведения акторов (агентов). Современные подходы в области мультиагентных систем фокусируются в большей степени на последнем аспекте, предоставляя широкие возможности описания внутренней и внешне наблюдаемой динамики поведения агентов. В то же время, описанию формальных структур и поведения организаций уделено мало внимания. Одной из причин этому является то, что в этих подходах организационная парадигма используется с целью улучшить вычислительные свойства распределенных алгоритмов, реализованных с помощью мультиагентных систем.

Для создания правдоподобных моделей реальных организаций различные аспекты формальных организационных структур и поведения организаций должны быть явно представлены. С целью понижения сложности моделирования различные аспекты организаций рассматриваются в данной диссертации с четырех взаимосвязанных перспектив (или видов): производительно-ориентированной, процессуально-ориентированной, структурно-ориентированной и агентно-ориентированной.

В рамках производительно-ориентированного вида рассматриваются взаимосвязанные структуры целей и индикаторов производительности организации и агентов. Процессуально-ориентированный вид описывает структуры процессов и ресурсов организации, а также динамические потоки управления процессами организации. В рамках структурно-ориентированного вида рассматриваются роли (или позиции) организации, а также отношения взаимодействия и власти между ними. Агентно-ориентированный вид описывает свойства и поведение агентов в организационном контексте. Виды взаимосвязаны посредством явно определенных отношений: в частности, цели организации достигаются посредством выполнения процессов, а процессы в свою очередь связаны с ролями, которые впоследствии назначаются агентам.

Разработанные формальные языки, используемые для описания понятий и отношений в рамках каждого вида базируются на выразительной многосортной логике предикатов. Для описания динамических аспектов организаций и логических рассуждений о поведении организации используется язык TTL (Temporal Trace Language), формальный фундамент (синтаксис и семантика) для

которого был разработан в рамках данной диссертации. Языки видов и TTL обладают выразительными свойствами для описания как количественных, так и качественных свойств систем (например, организаций).

Посредством предлагаемых языков даны точные определения понятиям и отношениям видов. Также определены аксиомы, описывающие правила использования элементов языка при построении моделей и ограничения целостности модели. При определении этих правил и ограничений были использованы теоретические наработки из области социальных наук (в частности, теории организаций), тем самым обеспечивая связь между формальным логическим фундаментом и результатами эмпирических исследований. Диссертация описывает методологические принципы построения моделей организаций, используя разработанные языки.

Для автоматизированного анализа организационных моделей язык моделирования должен быть исполнимым, то есть должен предоставлять возможности создания и обработки организационных моделей на компьютере. С этой целью диссертация описывает несколько исполнимых подмножеств языка TTL. Кроме того, в рамках диссертации разработана автоматизированная процедура трансформации сложных неисполнимых спецификаций на языке TTL в исполнимую форму, подходящую для автоматизированного анализа.

Диссертация описывает разнообразные автоматизированные методы анализа организаций с целью обнаружения ошибок, несоответствий и узких мест. Некоторые из них основаны на общих подходах к анализу логических моделей (например, метод проверки моделей (model checking)). Другие основаны на специализированных алгоритмах проверки определенных свойств организации (например, непротиворечивости структуры целей, требований к исполнению процессов, целостности каналов взаимодействия и структуры власти). Следует заметить, что вычислительные свойства специализированных алгоритмов лучше, чем у общих. Кроме того, разработаны методы для проведения симуляций различных сценариев поведения организации и автоматической проверки свойств системы на основе результатов этих симуляций. Также разработан похожий подход для автоматизированного анализа эмпирических данных о функционировании организации.

Предложенные в диссертации методы были применены на практике в рамках трех проектов из областей логистики, управления инцидентами и управления авиационным трафиком. Выразительные свойства языков моделирования позволили создать реалистичные модели организаций из этих проектов. Предложенные методы анализа позволили обнаружить неизвестные ранее проблемы разного типа в этих организациях, достоверность которых была подтверждена экспертами.

Подводя итог, формальные методы моделирования и анализа структур и поведения организаций разных типов, предложенные в диссертации, оказались применимыми на практике и полезными для понимания принципов функционирования организаций, обнаружения разных типов ошибок и несоответствий и для исследования динамики поведения организаций в разных условиях окружающей среды.

SIKS Dissertation Series

1998

Johan van den Akker, *DEGAS - An Active,Temporal Database of Autonomous Objects*, CWI, 1998-1

Floris Wiesman, *Information Retrieval by Graphically Browsing Meta-Information*, UM, 1998-2

Ans Steuten, *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*, TUD, 1998-3

Dennis Breuker, *Memory versus Search in Games*, UM, 1998-4

E.W. Oskamp, *Computerondersteuning bij Straftoemeting*, RUL, 1998-5

1999

Mark Sloof, *Physiology of Quality Change Modelling*; *Automated modelling of Quality Change of Agricultural Products*, VU,1999-1

Rob Potharst, *Classification using decision trees and neural nets*, EUR, 1999-2

Don Beal, *The Nature of Minimax Search*, UM, 1999-3

Jacques Penders, *The practical Art of Moving Physical Objects*, UM, 1999-4

Aldo de Moor, *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*, KUB, 1999-5

Niek J.E. Wijngaards, *Re-design of compositional systems*, VU, 1999-6

David Spelt, *Verification support for object database design*, UT, 1999-7

Jacques H.J. *Lenting, Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation*, UM, 1999-8

2000

Frank Niessink, *Perspectives on Improving Software Maintenance*, VU, 2000-1

Koen Holtman, *Prototyping of CMS Storage Management*, TUE, 2000-2

Carolien M.T. Metselaar, *Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief*, UVA, 2000-3

Geert de Haan, *ETAG, A Formal Model of Competence Knowledge for User Interface Design*, VU, 2000-4

Ruud van der Pol, *Knowledge-based Query Formulation in Information Retrieval*, UM, 2000-5

Rogier van Eijk, *Programming Languages for Agent Communication*, UU, 2000-6

Niels Peek, *Decision-theoretic Planning of Clinical Patient Management*, UU, 2000-7

Veerle Coup, *Sensitivity Analyis of Decision-Theoretic Networks*, EUR, 2000-8

Florian Waas, *Principles of Probabilistic Query Optimization*, CWI, 2000-9

Niels Nes, *Image Database Management System Design Considerations, Algorithms and Architecture*, CWI, 2000-10

Jonas Karlsson, *Scalable Distributed Data Structures for Database Management*, CWI, 2000-11

2001

Silja Renooij, *Qualitative Approaches to Quantifying Probabilistic Networks*, UU, 2001-1

Koen Hindriks, *Agent Programming Languages: Programming with Mental Models*, UU, 2001-2

Maarten van Someren, *Learning as problem solving*, UvA, 2001-3

Evgueni Smirnov, *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*, UM, 2001-4

Jacco van Ossenbruggen, *Processing Structured Hypermedia: A Matter of Style*, VU, 2001-5

Martijn van Welie, *Task-based User Interface Design*, VU, 2001-6

Bastiaan Schonhage, *Diva: Architectural Perspectives on Information Visualization*, VU, 2001-7

Pascal van Eck, *A Compositional Semantic Structure for Multi-Agent Systems Dynamics*, VU, 2001-8

Pieter Jan 't Hoen, *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*, RUL, 2001-9

Maarten Sierhuis, *Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design*, UvA, 2001-10

Tom M. van Engers, *Knowledge Management: The Role of Mental Models in Business Systems Design*, VUA, 2001-11

2002

Nico Lassing, *Architecture-Level Modifiability Analysis*, VU, 2002-01

Roelof van Zwol, *Modelling and searching web-based document collections*, UT, 2002-02

Henk Ernst Blok, *Database Optimization Aspects for Information Retrieval*, UT, 2002-03

Juan Roberto Castelo Valdueza, *The Discrete Acyclic Digraph Markov Model in Data Mining*, UU, 2002-04

Radu Serban, *The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents*, VU, 2002-05

Laurens Mommers, *Applied legal epistemology; Building a knowledge-based ontology of the legal domain*, UL, 2002-06

Peter Boncz, *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*, CWI, 2002-07

Jaap Gordijn, *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*, VU, 2002-08

Willem-Jan van den Heuvel, *Integrating Modern Business Applications with Objectified Legacy Systems*, KUB, 2002-09

Brian Sheppard, *Towards Perfect Play of Scrabble*, UM, 2002-10

Wouter C.A. Wijngaards, *Agent Based Modelling of Dynamics: Biological and Organisational Applications*, VU, 2002-11

Albrecht Schmidt, *Processing XML in Database Systems*, UVA, 2002-12

Hongjing Wu, *A Reference Architecture for Adaptive Hypermedia Applications*, TUE, 2002-13

Wieke de Vries, *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*, UU, 2002-14

Rik Eshuis, *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*, UT, 2002-15

Pieter van Langen, *The Anatomy of Design: Foundations, Models and Applications*, VU, 2002-16

Stefan Manegold, *Understanding, Modeling, and Improving Main-Memory Database Performance*, UVA, 2002-17

2003

Heiner Stuckenschmidt, *Onotology-Based Information Sharing In Weakly Structured Environments*, VU, 2003-1

Jan Broersen, *Modal Action Logics for Reasoning About Reactive Systems*, VU, 2003-02

Martijn Schuemie, *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*, TUD, 2003-03

Milan Petkovic, *Content-Based Video Retrieval Supported by Database Technology*, UT, 2003-04

Jos Lehmann, *Causation in Artificial Intelligence and Law - A modelling approach*, UVA, 2003-05

Boris van Schooten, *Development and specification of virtual environments*, UT, 2003-06

Machiel Jansen, *Formal Explorations of Knowledge Intensive Tasks*, UvA, 2003-07

Yongping Ran, *Repair Based Scheduling*, UM, 2003-08

Rens Kortmann, *The resolution of visually guided behaviour*, UM, 2003-09

Andreas Lincke, *Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture*, UvT, 2003-10

Simon Keizer, *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*, UT, 2003-11

Roeland Ordelman, *Dutch speech recognition in multimedia information retrieval*, UT, 2003-12

Jeroen Donkers, *Nosce Hostem - Searching with Opponent Models*, UM, 2003-13

Stijn Hoppenbrouwers, *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*, KUN, 2003-14

Mathijs de Weerdt, *Plan Merging in Multi-Agent Systems*, TUD, 2003-15

Menzo Windhouwer, *Feature Grammar Systems-Incremental Maintenance of Indexes to Digital Media Warehouses*, CWI, 2003-16

David Jansen, *Extensions of Statecharts with Probability, Time, and Stochastic Timing*, UT, 2003-17

Levente Kocsis, *Learning Search Decisions*, UM, 2003-18

2004

Virginia Dignum, *A Model for Organizational Interaction: Based on Agents, Founded in Logic*, UU, 2004-01

Lai Xu, *Monitoring Multi-party Contracts for E-business*, UvT, 2004-02

Perry Groot, *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*, VU, 2004-03

Chris van Aart, *Organizational Principles for Multi-Agent Architectures*, UVA, 2004-04

Viara Popova, *Knowledge discovery and monotonicity*, EUR, 2004-05

Bart-Jan Hommes, *The Evaluation of Business Process Modeling Techniques*, TUD, 2004-06

Elise Boltjes, *Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes*, UM, 2004-07

Joop Verbeek, *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise*, UM, 2004-08

Martin Caminada, *For the Sake of the Argument; explorations into argument-based reasoning*, VU, 2004-09

Suzanne Kabel, *Knowledge-rich indexing of learning-objects*, UVA, 2004-10

Michel Klein, *Change Management for Distributed Ontologies*, VU, 2004-11

The Duy Bui, *Creating emotions and facial expressions for embodied agents*, UT, 2004-12

Wojciech Jamroga, *Using Multiple Models of Reality: On Agents who Know how to Play*, UT, 2004-13

Paul Harrenstein, *Logic in Conflict. Logical Explorations in Strategic Equilibrium*, UU, 2004-14

Arno Knobbe, *Multi-Relational Data Mining*, UU, 2004-15

Federico Divina, *Hybrid Genetic Relational Search for Inductive Learning*, VU, 2004-16

Mark Winands, *Informed Search in Complex Games*, UM, 2004-17

Vania Bessa Machado, *Supporting the Construction of Qualitative Knowledge Models*, UvA, 2004-18

Thijs Westerveld, *Using generative probabilistic models for multimedia retrieval*, UT, 2004-19

Madelon Evers, *Learning from Design: facilitating multidisciplinary design teams*, Nyenrode, 2004-20

2005

Floor Verdenius, *Methodological Aspects of Designing Induction-Based Applications, UVA*, 2005-01

Erik van der Werf, *AI techniques for the game of Go*, UM, 2005-02

Franc Grootjen, *A Pragmatic Approach to the Conceptualisation of Language*, RUN, 2005-03

Nirvana Meratnia, *Towards Database Support for Moving Object data*, UT, 2005-04

Gabriel Infante-Lopez, *Two-Level Probabilistic Grammars for Natural Language Parsing*, UVA, 2005-05

Pieter Spronck, *Adaptive Game AI*, UM, 2005-06

Flavius Frasincar, *Hypermedia Presentation Generation for Semantic Web Information Systems*, TUE, 2005-07

Richard Vdovjak, *A Model-driven Approach for Building Distributed Ontology-based Web Applications*, TUE, 2005-08

Jeen Broekstra, *Storage, Querying and Inferencing for Semantic Web Languages*, VU, 2005-09

Anders Bouwer, *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*, UVA, 2005-10

Elth Ogston, *Agent Based Matchmaking and Clustering - A Decentralized Approach to Search*, VU, 2005-11

Csaba Boer, *Distributed Simulation in Industry*, EUR, 2005-12

Fred Hamburg, *Een Computermodel voor het Ondersteunen van Euthanasie-beslissingen*, UL, 2005-13

Borys Omelayenko, *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*, VU, 2005-14

Tibor Bosse, *Analysis of the Dynamics of Cognitive Processes*, VU, 2005-15

Joris Graaumans, *Usability of XML Query Languages*, UU, 2005-16

Boris Shishkov, *Software Specification Based on Re-usable Business Components*, TUD, 2005-17

Danielle Sent, *Test-selection strategies for probabilistic networks*, UU, 2005-18

Michel van Dartel, *Situated Representation*, UM, 2005-19

Cristina Coteanu, *Cyber Consumer Law, State of the Art and Perspectives*, UL, 2005-20

Wijnand Derks, *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*, UT, 2005-21

2006

Samuil Angelov, *Foundations of B2B Electronic Contracting*, TUE, 2006-01

Cristina Chisalita, *Contextual issues in the design and use of information technology in organizations*, VU, 2006-02

Noor Christoph, *The role of metacognitive skills in learning to solve problems*, UVA, 2006-03

Marta Sabou, *Building Web Service Ontologies*, VU, 2006-04

Cees Pierik, *Validation Techniques for Object-Oriented Proof Outlines*, UU, 2006-05

Ziv Baida, *Software-aided Service Bundling – Intelligent Methods & Tools for Graphical Service Modeling*, VU, 2006-06

Marko Smiljanic, *XML schema matching – balancing efficiency and effectiveness by means of clustering*, UT, 2006-07

Eelco Herder, *Forward, Back and Home Again - Analyzing User Behavior on the Web*, UT, 2006-08

Mohamed Wahdan, *Automatic Formulation of the Auditor's Opinion*, UM, 2006-09

Ronny Siebes, *Semantic Routing in Peer-to-Peer Systems*, VU, 2006-10

Joeri van Ruth, *Flattening Queries over Nested Data Types*, UT, 2006-11

Bert Bongers, *Interactivation - Towards an e-cology of people, our technological environment, and the arts*, VU, 2006-12

Henk-Jan Lebbink, *Dialogue and Decision Games for Information Exchanging Agents*, UU, 2006-13

Johan Hoorn, *Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change*, VU, 2006-14

Rainer Malik, *CONAN: Text Mining in the Biomedical Domain*, UU, 2006-15

Carsten Riggelsen, *Approximation Methods for Efficient Learning of Bayesian Networks*, UU, 2006-16

Stacey Nagata, *User Assistance for Multitasking with Interruptions on a Mobile Device*, UU, 2006-17

Valentin Zhizhkun, *Graph transformation for Natural Language Processing*, UVA, 2006-18

Birna van Riemsdijk, *Cognitive Agent Programming: A Semantic Approach*, UU, 2006-19

Marina Velikova, *Monotone models for prediction in data mining*, UvT, 2006-20

Bas van Gils, *Aptness on the Web*, RUN, 2006-21

Paul de Vrieze, *Fundaments of Adaptive Personalisation*, RUN, 2006-22

Ion Juvina, *Development of Cognitive Model for Navigating on the Web*, UU, 2006-23

Laura Hollink, *Semantic Annotation for Retrieval of Visual Resources*, VU, 2006-24

Madalina Drugan, *Conditional log-likelihood MDL and Evolutionary MCMC*, UU, 2006-25

Vojkan Mihajlovic, *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*, UT, 2006-26

Stefano Bocconi, *Vox Populi: generating video documentaries from semantically*

annotated media repositories, CWI, 2006-27

Borkur Sigurbjornsson, *Focused Information Access using XML Element Retrieval*, UVA, 2006-28

2007

Kees Leune, *Access Control and Service-Oriented Architectures*, UvT, 2007-01

Wouter Teepe, *Reconciling Information Exchange and Confidentiality: A Formal Approach*, RUG, 2007-02

Peter Mika, *Social Networks and the Semantic Web*, VU, 2007-03

Jurriaan van Diggelen, *Achieving Semantic Interoperability in Multi-agent Systems: A Dialogue-based Approach*, UU, 2007-04

Bart Schermer, *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance*, UL, 2007-05

Gilad Mishne, *Applied Text Analytics for Blogs*, UVA, 2007-06

Natasa Jovanoviç, *To Who It May Concern – Addressee Identification in Face-to-Face Meetings*, UT, 2007-08

Mark Hoogendoorn, *Modeling of Change in Multi-Agent Organizations*, VU, 2007-09

David Mobach, *Agent-Based Mediated Service Negotiation*, VU, 2007-09

Huib Aldewereld, *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*, UU, 2007-10

Natalia Stash, *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System*, TUE, 2007-11

Marcel van Gerven, *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty*, RUN, 2007-12

Rutger Rienks, *Meetings in Smart Environments; Implications of Progressing Technology*, UT, 2007-13

Niek Bergboer, *Context-Based Image Analysis*, UM, 2007-14

Joyca Lacroix, *NIM: a situated computational memory model* , UM, 2007-15

Davide Grossi, *Designing Invisible Handcuffs: Formal Investigations in Institutions and Organizations for Multi-agent Systems*, UU, 2007-16

Theodore Charitos, *Reasoning with Dynamic Networks in Practice*, UU, 2007-17

Bart Orriens, *On the development an management of adaptive business collaborations*, UvT, 2007-18

David Levy, *Intimate relationships with artificial partners*, UM, 2007-19

Slinger Jansen, *Customer Configuration Updating in a Software Supply Network*, UU, 2007-20

Karianne Vermaas, *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005*, UU, 2007-21

Zlatko Zlatev, *Goal-oriented design of value and process models from patterns*, UT, 2007-22

Peter Barna, *Specification of Application Logic in Web Information Systems*, TUE, 2007-23

Georgina Ramirez Camps, *Structural Features in XML Retrieval*, CWI, 2007-24

Joost Schalken, *Empirical Investigations in Software Process Improvement*, VU, 2007-25

2008

Katalin Boer-Sorbán, *Agent-Based Simulation of Financial Markets: A modular, continuous-time approach*, EUR, 2008-01