Scientific Annals of Computer Science vol. 25 (2), 2015, pp. 317–355
doi: 10.7561/SACS.2015.2.317

Verification and Validation of Formal Data-Centric Business Models

Timur UMAROV[1]

Abstract

This paper addresses the problem of describing and analysing internally consistent data within business process workflow specifications. We use Rodin platform for verifying the correctness of the Event-B models. These models we obtain from an ontology and an associated set of normative constraints by applying mapping rules. The latter enable us to transform these specifications into Event-B modular artefacts. The resulting model, by virtue of the Event-B formalism, is very close to a typical loosely coupled component-based implementation of a business system workflow, but has the additional value of being amenable to theorem proving techniques to check and refine data representation with respect to process evolution. In this paper, we give a formal account of the design specifications defined by Event-B modules and perform verification and validation by using theorem proving techniques provided by Rodin platform.

Keywords: Event-B, verification, validation, business process, formal specifications, Rodin platform.

1 Introduction

Business process management (BPM) is a challenging aspect of the enterprise. Middleware support for BPM, as provided by, for example, Oracle, Biztalk and the recent Windows Workflow Framework, has met some challenges with respect to performance and maintenance of workflow, which can

[1]Department of Management Information Systems, Faculty of Information Technology, Kazakh-British Technical University, 59 Tole bi str., 050000 Almaty, Kazakhstan, E-mail: t.umarov@kbtu.kz

be directly related to the efficiency of business modeling in terms of preserving semantics. The process of developing business models often leaves out meeting a requirement of specifying *data* in workflows, but mainly defines dataflow.

The central challenge to BPM is complexity: business processes are becoming widely distributed, interoperating across a range of inter- and intra-organizational vocabularies and semantics. It is important that complex business workflows are checked and analysed for optimality and trustworthiness prior to deployment. The problem becomes worse when we consider the enterprise's demand to regularly adapt and change processes. For example, the growth of a company, changes to the market, reevaluation of tasks to minimize cost. All these factors require re-engineering or adaptation of business processes and continuous improvement of individual activities for achieving dramatic improvements of performance critical parameters such as quality (of a product or service), cost, and speed [36]. Re-engineering of a complex workflow implementation is dangerous, due to existing dependencies between tasks.

Martin Hepp, et al. [24] are emphasising the notion of *workflow-centricity* of business processes, the main weakness of which is the focus exclusively on control flow patterns. The workflow-centricity is particularly true for BPEL. This brings a major disadvantage to such workflows: it becomes impossible to access a business process space at the knowledge level which could potentially facilitate discovering of processes or their fragments for serving particular purposes. It is also impossible to query process space at the semantic level, e.g. using logical expressions and machine reasoning, which could help automate tasks execution.

Formal methods can assist in meeting the challenge of complexity, as their mathematical basis enable us to analyze and refine a system specification. Petri nets [30] are famous for modeling and analyzing business processes. However, complex systems often involve a number of different aspects that entail separate kinds of analysis and, consequently, the use of a number of different formal methods. When using formal methods engineers define specifications and attempt to prove their correctness applying a range of different tools. Verification of the specifications in this case is useful for it ensures that the system under question is internally consistent. It gives the engineers a sense of confidence that the developed specifications are error free and can operate correctly when implemented in a programming language. Validation is the process of ensuring that the developed specification is in line with the wishes of the clients. In other words, developers check the specifications against the unstructured and informal documentations of requirements.

A business process implementation within a BPM middleware requires detailed treatment of both information flow and *information content*. The abstraction gap is identified as follows: an abstract workflow that ignores information content provides an abstract view of business processes that does not fully define the key aspects necessary for BPM implementation [24]. We argue that this abstraction gap can be addressed by developing event-driven *data-centric workflow* models in the Event-B language from an initial business process requirements specification. We employ a Model Driven Architecture approach. The initial computational independent model (CIM) is written as a number of requirements specifications. For this purpose we use ontologies and a normative language of MEASUR [17]. The method has an almost 30-year history and is widely used within the organisational semiotics community, but less well-known in Computer Science. Its roots lie in the philosophical pragmatism of Peirce, the semiotics of Saussure and Austin's speech act theory. It is model-based, with ontologies and normative constraints forming the central deliverables of a requirements document. We employ MEASUR notation because our starting point is information systems analysis, where MEASUR has found the most application. We have found that MEASUR's normative constraints lend themselves to transformation into our languages related to the platform independent model (PIM), which is in our approach defined using modular artifacts of formal specifications written in Event-B. These modular specifications are represented as a collection of machines that are interacting with each other via external variables. However, our approach for defining CIM, which is formally defined as a collection of norms and definitions for prescribing agents different patterns of behavior depending on the conditions [7], should be readily adaptable to a number of similar notations in use in the multi-agents and normative specification research communities. To further our research, in this paper we attempt to demonstrate a formal account of the design specifications defined by Event-B modules and perform verification and validation by using theorem proving techniques provided by Rodin platform.

The Event-B language is used for specifying, designing, and implementing software systems. The language may be used to develop software by a process of gradual refinement, from an abstract, possibly nonexecutable system model, to intermediate system models that contain more detail on how to treat data and algorithms, to a final, optimised, executable system. In this process,

- the first abstract model in this refinement chain should be verified for consistency,
- each step in the refinement should be formally checked for semantic preservation.

Consistency will then be preserved throughout the chain. This means that the final executable refinement can be trusted to implement the initial abstract specification. Our approach defines a transformation of MEASUR models to Event-B machines, permitting

- a full B-based formal semantics for vocabularies and data manipulation carried out within the modeled workflow, which can be validated for consistency; and
- an initial, abstract B model that can be further refined using the B-method to a final optimal executable system in an object-oriented workflow middleware, such as Windows Workflow Foundation.

A notion of semantic compatibility holds over the transformed models, so that any property derived over the normative-ontological view of the system will hold over potential processes that arise from the Event-B machine.

Different manipulations towards changing and adapting business processes are most often performed by domain experts at the initial requirements specifications level. Use of simple tools for this purpose helps to solve the problem of complexity by verification of correctness and consistency of business workflows using different properties inherently provided by those tools. However, they do not define the nature of data flowing through the processes and activities. This leads to final requirements documents developed by domain experts that are not semantically descriptive enough to reflect data computations involved in business process executions. On the contrary, design level implementations can be semantically rich and formal, reflecting all the important information necessary for correct and effective business process executions at the expense of excessive complexity for domain experts. Moreover, all good designs must be modular which represents yet another challenge in terms of implementing them. The distance between requirements and design in terms of the degree of their semantical richness brings about a pronounced semantic gap.

This paper is a part of the research in which we develop a model-driven transformation from normative requirements to formal designs. This type of transformation is high level and requires use of non-trivial mapping rules. In what follows, we describe an approach of using Event-B as a formal language for describing the design patterns of the system [13] and demonstrate how these specifications can be checked for consistency. Additionally, we are outlining our future work of roundtrip engineering, by which we can detect inconsistencies in high level normative specifications.

The paper proceeds as follows:

- In Section 2, we sketch the nature of our normative ontology language of MEASUR.
- Section 3 provides a brief introduction to Event-B specifications and provides its formal account by focusing on the definitions of context and machines;
- Section 4 then outlines a discussion on soundness of transformation, definitions for normative requirements and gives a description on verification and validation of the generated design specifications;
- Section 5 discusses related work on enriching workflow models, ontological lifting, correctness of a workflow, and integrating different languages.
- Section 6 discusses the future work towards the concept of "roundtrip engineering" for ensuring correctness of the requirements specifications.

2 Normative Ontology

Business workflows modelled by MEASUR's ontologies and normative definitions with the elements of responsibility, agency and modality of the theory of normative positions specify business processes as modular actions, execution of which depends on the modality positions and agents which bear *responsibility* for execution of those actions. The execution of actions is understood as a change of the state of affairs. For agents to be responsible to see to it that some state of affairs is obliged / permitted / not permitted to be true (the collocation "to see to it that" is taken from the original research on logic and theory of normative systems and means that any agent in question is responsible for some state of affairs to be true), there are one or more pre-conditions which need to hold. In what follows, we attempt to describe our approach of formalizing normative specifications.

2.1 Methodology MEASUR

Method for Eliciting, Analyzing and Specifying User Requirements (MEASUR) represents a radically new set of norm-oriented methods for business systems modeling and requirements specification for software development [17]. MEASUR is a result of more than 30 years of research in the area of organizational semiotics. A number of related papers and books were published over these years, of which the most recent ones can be found in [9]. The overall spirit of these publications is that they see organizations as information systems with the set of agents, their corresponding set of potential actions (affordances) and set of norms which govern agents' behavior. MEASUR offers five phases for business modeling and software development, of which three the most important phases are problem articulation methods, semantic analysis method and norm analysis method.

The general ontological theory is concerned with fundamental questions of classifying everything that exists in the world into different categories, describing that *everything* while it exists, and seeking to elicit possible hierarchies and dependencies among the things that make up that everything. Liu [17] defines the term everything as constituent notions such as thing, entity, individual, universal, particular, substance, event, process and state. These notions are embraced with the general ontological study.

Although there are different types of ontologies related to knowledge and its representation, the only type of ontology that semantic analysis is relevant to is that which recognizes only our own behavior in accordance with our own ambiance. This type of ontology can be defined as a collection of representational data that models a certain domain of knowledge or discourse. Everything that exists in the world is dependent on the agent's behavior. In other words, the meaning of a thing is related to the ability of the agent to recognize that thing as a particular entity.

Semantic Analysis has its roots in semiotics, the philosophical investigation of signs. MEASUR applies to information system analysis a number of ideas and approaches from philosophy of language, drawing on the pragmatism of Peirce, semiotics of Saussure and the epistemology of Wittgenstein and Austin. The method's core assumption is knowledge and information exists only in relation to a knowing *agent* (a single human, a machine, or a social organization). There is no Platonic reality which defines Truth. Instead, Truth is a derived concept that might be defined as an *agreement* between a group of agents. An agent is *responsible* for its knowledge. When a group of agents agree on what is true and what is false, they accept responsibility for that judgment. Following Wittgenstein [6], MEASUR considers an information system as a "language game", a form of activity involving a party of agents that generates meaning. In an information-system-as-language-game, the meaning of data derives from usage by agents, rather than from a universal semantics.

There have been a number of attempts to use semantic analysis normative ontologies as the language for a business process management engine. The most widely used is Liu's NORMBASE system [17]. In such systems, the ontology serves as a *classification* schema for business data, actions and agents, while norms define the conditions under which actions may be invoked to create and manipulate data.

In MEASUR, an information system is viewed as a business domain model which identifies agents, set of actions these agents can perform (i.e. agents' repertoire of behavior) and relationships, and forms of knowledge that can result from these actions. These concepts are identified as types of *affordances*. An affordance is a pattern of behavior of a certain agent provided by the environment that he is directly related to. In other words, environment offers a knowledge to an agent according to which the agent forms certain repertoire of behavior. Effectively, every concept in a MEASUR ontology is an affordance.

MEASUR subclasses the notion of affordance as follows:

- A *business entity* – such as a user account or a bank loan – is an affordance in the sense that it is associated with a set of permissible behaviours and possibilities of use. For the purpose of business process analysis, business entities are used to identify the main kinds of data that are of importance in an organization's processes.

- A *relationship* – such as a contract – between business entities or agents is an affordance in the sense that it is defined by the behaviour it generates for the parties involved in the contract.

- *Agents* are affordances in terms of the actions they can perform and the things that may be done to them. Agents then occupy a special status in that they take responsibility for their own actions and the actions of others and can authorize patterns of behavior.

- *Communication acts* are affordances in the sense of the available actions that the appropriate related agent or agents may perform depending on their possessed knowledge and experience.

- The structure of a business entity, relationship or agent is given via a list of associated properties, called *determiners*. Determiners are properties and attributes of affordances, such as address or telephone number associated with a user account.

- *Units of measurement* are typical data types that type determiners and other values associated with affordances. The latter two concepts are considered as affordances as their values constrain the possible behavior of their owners. We will assume the units of measurement comprise the usual basic data types available to most programming and design languages: that is, *Bool* (booleans), *String* (strings), *Int* (integers), *Float* (floating point numbers) and finite enumerations (of the form $\{v_1, \ldots, v_n\}$, where each v_i enumerates a permissible value of the type). However, depending on the information system under consideration, the business analyst may add any other kind of simple, unstructured data type. Structured data (for example, details relating to a client's bank account) should be provided as a business entity.

The ultimate deliverable of these three stages is what we will refer to as a *normative ontology*. A normative ontology consists of what might be called a semi-formal requirements document, in the sense that it can be understood readily by clients, business analysts and developers. This model breaks down an information system into a set of business data, communicating agents (stakeholders, departments, people, computer programs) and business processes that agents can invoke in order to manipulate data between each other. The model consists of a role-and-relationships ontology together with a set of norms that formally define the structure and expected and permitted interactions within an organization and its processes.

2.2 The Normative Perspective

The theory of normative positions is based on the fundamental principles of deontic logic. Deontic logic is a formal system and represents a field of symbolic logic which is concerned with normative concepts such as obligation, permission, and prohibition used in contractual relationships for classifying actions and states of affairs [27]. The word deontic comes from Greek and means “of that which is binding”. Standard Deontic Logic (SDL, KD, or simply D) is the most studied and axiomatically defined system of deontic logic. SDL 1) represents a monadic deontic logic because its deontic operator is a one-place operator, 2) builds on propositional logic, and 3) is formally specified by a Kripke-style semantics. SDL uses the following notation to denote its main concepts: ObA stands for it is obligatory that A and PeA – it is permissible that A.

Deontic logic have not yet found wide application in technical areas such as Computer Science. However, deontic modalities could be efficiently used in specifying technical requirements for different business domains. For instance, modality “obligatory” may well model the requirement that a system must check whether a buyer have enough money in his back account before making a purchase in an on-line or physical store; and modality “permissible” can be applied to specify a rule that a bank may check (it is permissible that bank checks) credit history of a client before issuing a loan. More on the definition and a Kripke-style possible world semantics for SDL can be found in [26].

The theory of normative positions was originally inspired by analytical study of law and originated in the works of Stig Kanger [32, 33], Ingmar Pörn [10, 11], and Lars Lindahl [20, 21]. The Kanger-Lindahl theory is characterized by attempts to apply modal logic – mainly standard deontic logic, the field of logic concerned with obligations and permissions – and the logic of action and agency to the concepts of legal and normative relations which Wesley Newcomb Hohfeld (1879–1918), an American jurist, had regarded to as the fundamental legal conceptions of jurisprudence [35]. These normative relations are alternatively referred to as normative positions that take the forms of obligations, permissions, duties, and rights of agents of a community, society or some other form of organization. By agents here we mean humans, machines, or both. The Kanger-Lindahl theory also embraces the formal representation of more complex normative positions such as entitlement, authorization and responsibility.

Besides the areas of legal knowledge representation (e.g. representation of laws, regulations, legal contracts, etc.), where the theory of normative positions found its initial application, there are also other areas such as Computer Science, where the theory contributed much for formal representation of relations between agents. For example, Jones and Sergot [1, 2] describe a modified version of the Kanger-Lindahl theory and attempt to apply it to the problem of access control and security policies specifications and analysis for databases. In their attempts [2], authors illustrate by an example of library regulations for governing the procedures of loaning books that the use of formal methods in developing system specifications have to be taken seriously whenever it is necessary to analyze an ideal case and an actual one and see how the actual behavior deviates from that of ideal. Use of such formal methods as deontic logic, as an integral part of the normative system proposed by Jones and Sergot, can help in revealing the possibility of violations, i.e. those deviations that had actually occurred. According to Jones and Sergot [2], the use of deontic logic will in general allow (i) to reason with the specifications developed, (ii) to be able to test the internal consistency of the system specifications as a whole, and (iii) to use theorem provers to implement and test different components of the system.

In the works of Kanger, Pörn, and Lindahl [32, 33, 10, 11, 20, 21], deontic logic was merged with logic of action and agency to provide a formal account of complex social interactions within organizations, which can be related to the technical context by applying it to the multi-agent environment. They introduce a so-called relativised modal operator which is designated as E_a, where a represents a responsible agent. The approach is partially similar to that of dynamic logic in the sense that it also assumes that an action, if performed, should bring about a certain state of affairs. For example, expression $[a]A$ would mean that after performing action a it is necessarily the case that A holds. In other words, a must bring about A. Analogously, $\langle a\rangle A$ means that after performing action a it is possibly the case that A holds, or a might bring about A.

However, in the theory of normative positions all actions are associated with their respective responsible agents which makes the semantics more expressive. When modeling business systems using this theory we now have to deal with the element of agent's responsibility embedded in the process of bringing about new states of affairs. The overall concept is formalized by

$$E_a A, \tag{1}$$

which is read as "agent a sees to it that A is the case" or similarly "agent a brings it about that A". It is important to note that actions in this case represent a relationship between agents and the state of affairs that these agents bring about (or are responsible for the respective states of affairs to be true).

The following expressions are properties for the action operator. The first axiom schema implies that the action operator is a *success* operator:

$$\vdash E_a A \rightarrow A. \tag{2}$$

It is read as: if agent a brings it about that A then A is indeed the case. The second property represents a rule of inference:

$$\textit{If} \vdash A \leftrightarrow B \textit{ then} \vdash (E_a A \leftrightarrow E_a B). \tag{3}$$

Although the approach of combination of deontic logic with action logic is reminiscent of that of dynamic logic by its rules and operators, the theory of normative positions provides higher expressivity in a way that, unlike in dynamic logic: (i) by using operator $E_a A$ one can express different atomic positions agent a can be in with respect to a particular state of affairs A; and (ii) using E_a operator gives another important advantage, namely, one can also formalize a normative interpersonal relationship by means of iterating the action operators:

$$E_a E_b A. \tag{4}$$

The examples of using this form of iteration can be demonstrated by the following:

$$E_a \neg E_a A \quad \textit{and} \quad \neg PeE_b \neg E_a A \tag{5}$$

where the former effectively implies that agent a *refrains* from seeing to it that A and the latter means that the agent b is forbidden to prevent agent a from seeing to it that A. The detailed definition is described by Jones and Sergot in [2].

2.3 Formalizing MEASUR

MEASUR's originator, Roland Stamper, while having a firmly applied background in system development, was influenced by ideas from philosophy of language and the expressive possibilities of deontic logic. However, in spite of drawing upon these ideas to develop the language and approach, it was always, first and foremost, a semiformal approach to requirements analysis, lacking a full formal semantics for analysis and reasoning. His latest ideas can be found in [18, 19] and in other of his publications.

The way in which MEASUR currently treats norms is analogous to the use of OCL within UML, that is syntaxes of OCL and UML are based on formal languages, but within their respective methods they do not have a precise semantics, which we give for MEASUR. In this paper, we provide what might be called a faithful formalization of MEASUR by restricting its norms to precisely the kind of logical language that inspired its notion of behavioral norm: that is, a limited version of deontic logic, combined with notions of agency inspired by the theory of normative positions. This will be defined as a first-order logic together with a straightforward semantics.

The language proposed in this paper is somewhat more complex, due to its relational nature and conformance to an ontology. However, we need not study the full set of formulae given: we *restrict* our attention to a subset of normative formulae that correspond to the informal schema for behavioural norms given in Definition 1.

A MEASUR normative definition is of the form:

if *trigger* occurs and the *pre-condition is satisfied,*
then *agent* performs an action so that *post-condition*
is Obliged/Permitted/Not permissible from resulting.

Importantly, MEASUR norms never contain deontic quantifiers within the trigger, pre-condition or post-condition statements. Furthermore, actual communication acts are only mentioned in the post-condition: the trigger and pre-condition statement only refer to relations from the ontology. All three elements of the definition can refer to agents and entities, however. Finally, the prescription of an agent's responsibility and a given deontic obligation are only given in the implication of the constraint.

We can therefore restrict our attention to a subset of behavioral norms for a given ontology that naturally preserves these syntactic constraints within our formalization, now defined.

Definition 1 (Formal behavioral norms for an ontology) *Consider any relational ontology*

$$\langle \mathit{UNIT}, \mathit{ENTITY}, \mathit{AGENT}, \mathit{CN}, \mathit{COMMACT}, \mathit{RN}, \mathit{REL}, \mathit{DETERM} \rangle \qquad (6)$$

where

- *UNIT is the set of possible units of measurement (basic data types) available.*
- *ENTITY is a set of entities. Each entity is of the form $N\{a_1 : T_1, \ldots, a_n : T_n\}$, where N is a unique name, each a_i is a unique determiner name and each $T_i \in \mathit{UNIT}$, $i = 1, \ldots, n$. If there are no arguments, then we simply denote it by N.*

- *$AGENT$ is a set of agents. Each agent is of the form $N\{a_1 : T_1, \ldots, a_n : T_n\}$, where N is a unique name, each a_i is a unique determiner name and each $T_i \in UNIT$, $i = 1, \ldots, n$. If there are no arguments, then we simply denote it by N. Sets $ENTITY$ and $AGENT$ are disjoint.*

- *$COMMACT$ is a mapping from a unique finite set of communication act names CN to the set of communication acts, consists of trinary relationships between two agents and one entity:*

$$CN \rightarrow AGENT \times AGENT \times ENTITY$$

- *REL is a mapping from a finite set of unique relationship names RN to a set of binary relationships between agents and/or entities:*

$$RN \rightarrow AGENT \cup ENTITY \times AGENT \cup ENTITY$$

- *$DETERM$ is a determiner, which represents a property and/or attributes of affordances.*

The set of behavioral norms *over a given ontology O, BN_O, is defined to be any formula from $Formulae_O$ (the set of all possible formulas in ontology) of the form*

$$G \rightarrow E_a \mathbf{D} Post \tag{7}$$

where

- $\mathbf{D}$ *is a deontic operator Ob or Pe.*

- *The only free variables occurring in the set of guards G and the set of definitions of norms DEF (the set of all possible definitions for communication acts) are agent or entity variables from Var_{AGENT} and Var_{ENTITY}.*

The idea of a behavioral norm is to associate knowledge and information with agents, who produce and are responsible for it. From a philosophical perspective, truth is then defined as something that an agent brings about and is responsible for. From the perspective of determining how to implement a normative ontology as a workflow-based system, we view agents as corresponding to subsystems, business entities to specifications of data and behavioral norms to expected dynamic interaction protocols between subsystems.

MEASUR and, in particular, our logical restriction of MEASUR, allows much flexibility when detailing the intended meaning of communication acts. This can be done by clarifying assertions. When it comes to the question of implementation of a communication act, an analyst will always ask the client: what is entailed by this act? The client will then explain what changes the act is expected to make on the elements of the ontology.

We can encode this description as a *definition* DEF of the act A, of the form

$$A \rightarrow DEF \tag{8}$$

and

$$DEF \rightarrow A \tag{9}$$

henceforth abbreviated as

$$A \leftrightarrow DEF \tag{10}$$

where DEF is any MEASUR formula not involving communication acts, both A and DEF sharing the same free variables.

We are now ready to define our formal notion of a MEASUR requirement analysis document. Essentially, it consists of an ontology and pairs of behavioral norms and definitions. These norms specify, given certain conditions are true, an agent is obliged / permitted / not permitted to see to it that certain state of affairs is true. These types of normative specifications are not descriptive enough for requirements engineers to model changes in states of affairs with the necessary depth of data definition. Therefore, we need additional annotations for our norms, which we refer to as *definitions*. We hereafter present this norm/definition pair as so called *normative tableaux*.

Definition 2 (Normative tableaux) *Consider any relational ontology of the form (6). A requirements specification consists of pairs of behavioural norms over O from BN_O, each paired with definitions from $Formulae_O$ of the form*

$$REQ = \{(G_i \rightarrow E_{b\,i} \;\; D_i \;\; A_i, A_i \leftrightarrow DEF_i) \quad | \quad i = 1, \ldots, n\} \tag{11}$$

where

- *each G_i is a single guard, particular for a given norm.*
- *each D_i is a deontic operator Ob or Pe.*

- *A_i is a single communication act.*
- *The only free variables occurring in G_i, DEF_i and A_i are agent or entity variables from Var_{AGENT} and Var_{ENTITY}.*

Each pair consists of a norm and a respective definition of that norm, and is called a normative tableaux.

A model $\mathcal{M}$ is said to satisfy REQ if it validates the quantified conjunction of all REQ's normative tableaux. That is, $\mathcal{M}$ satisfies REQ when

$$\mathcal{M} \models \left(\begin{array}{l} \forall\, x_1 : T_1 \bullet \ldots, x_n : T_k \bullet \\ G_1 \to E_{b_1}\ D_1\ A_1 \wedge A_1 \leftrightarrow DEF_1 \\ \wedge \ldots \wedge \\ G_n \to E_{b_n}\ D_n\ A_n \wedge A_n \leftrightarrow DEF_n \end{array} \right) \tag{12}$$

where $x_1 : T_1, \ldots, x_n : T_k$ is the list of all free variables contained in the formulae of the tableaux, and the sign "$\bullet$" is used as a divider between declarations.

We will employ two important conventions within the rest of this paper.

Assumption 1 *We will only deal with* deterministic *tableaux. These are requirements of the form*

$$REQ = \{(G_i \wedge NEG(G_i) \to E_{b\ i}\ D_i\ A_i, A_i \leftrightarrow DEF_i) \mid i = 1, \ldots, n\} \tag{13}$$

where

$$NEG(G_i) = \bigwedge\{G_j \mid j \neq i\} \tag{14}$$

However, for readability's sake, we will always write

$$REQ = \{(G_i \to E_{b\ i}\ D_i\ A_i, A_i \leftrightarrow DEF_i) \mid i = 1, \ldots, n\} \tag{15}$$

for equation (13) keeping the $NEG(G_i)$ always implicit within the pre-conditions.

Assumption 2 *We assume that all definitions are either* atomic *formulae or else a* conjunction *of atomic formulae. While in theory this weakens the power of our specification language, it has been observed that MEASUR specifications must be reduced to a deterministic form at some stage of requirements, prior to being handed over to implementation and design experts.*

This assumption is reasonable to employ for a wide range of business systems: it means that, at any point in time, at most only one norm is applicable – we can never have two norms being applicable at the same time.

Assumption 3 *Given a normative requirements specification of the form (13), when we specify an individual norm or definition from a tableaux,*

$$(G \rightarrow E_b \ Ob \ A) \tag{16}$$

we will often write full quantification

$$\forall x_1 : T_1 \bullet \ldots \bullet \forall x_n : T_n \bullet (G \rightarrow E_b \ Ob \ A) \tag{17}$$

The purpose of this is merely to indicate the types of the free variables contained within the formulae of the norm or definition. This notation should not be understood as restricting the interpretation of variables to be local to the particular norm or definition.

It is important to note that, within the intended meaning of an entire specification REQ, quantification is, in fact, understood to be given on the outside of an entire conjunction of all tableaux. That is, if a variable x occurs in a number of norms within REQ, it is to be interpreted by the same object in its model.

A model for an ontology together with a set of behavioral norms B_1, ... B_n is one in which each norm is true for $\mathcal{M}$. Depending on the model and the interpretation, this might be an abstract representation of a system execution, or might actually be an implementation of the specification: for example, one in which each possible world corresponds to an actual system state.

2.4 Example

In general This section will describe application of the methodology of specifying norms within the framework of new improved MEASUR. In formalizing MEASUR, we illustrate norms as communication acts that are enabled by an agent, using the elements of agency and action of the theory of normative positions. In the framework of new improved MEASUR, in order to specify norms we use so called normative tableaux. Tableau is a table for defining norms and their data definitions. In other words, each normative tableau illustrates a norm as a pre-condition and a responsibility expression that relates an agent to a particular communication act taken from MEASUR ontology. A tableau also provides further notation for these norms in the form of definitions, so that each norm is paired with its definition. All norms have their definitions to obtain a certain depth of data definition and to be able to manipulate with these data. We need these definitions within our new MEASUR in order to provide ways for requirement engineers to define and describe data in conjunction with ontologies.

The subject of our example is to process an order made by a customer using her credit card. The meaning of processing an order embraces herein such activities as receiving a new order, processing its data (price, customer's credit card information), invoicing and dispatching an order, interactions with warehouse and rejecting an order. In performing these activities in the order specified, we examine interaction between three involved players in an organized manner.

The first player, which starts the overall process of ordering is Customer. This player orders products, thereby creating a new instance of order and sends it to the electronic ordering system, which further processes it in a sequence specified below. The electronic ordering system eOrder initiates one activity after another and at some point starts interaction with yet another player called Warehouse. All these activities and communications between different players shift the order from one state to another. All orders have a strict correspondence with its particular customer and all orders are stored in the system during their processing until they are rejected. The activity of rejection removes the order from the system and cancels the correspondence with its particular customer.

The activities of making a purchase using our system is of the following order:

1. When a customer makes an on-line order, he initiates an interaction with the electronic ordering system by creating a new instance of the order and changing the status of this order to "received".

2. Electronic ordering system initiates the next activity which processes this new order by checking order's data and customer's solvency. If this check is successful, then the status of this order changes from "received" to "pending". Otherwise, if the order's total cost is higher than customer's credit card limit, then the status of his order changes to "rejected", which starts the rejecting process by removing the order from the system.

3. When the order's state changes to "pending" and if the product or-

dered is in stock, the system initiates the process of invoicing it by changing its status to “invoiced”.

4. If the order is not available in the stock, then the electronic ordering system initiates a communication to the warehouse by starting the “request_increase” activity.

5. After successful invoicing, the order’s status changes to “dispatched” to indicate that the order is dispatched.

The diagram for this example may be similar to UML’s sequence diagrams. But, it is different in the way of providing more expressive definitions for activities, interactions, and the notion of responsibility. We will employ this example to illustrate the use of the MEASUR methodology and its formalized version.

Applying methodology In our treatment, affordances are viewed as *classifications* of things within a business system, with an ontology defining a type structure for the system. An actual executing system consists of a collection of affordance *instances* which possess the structure prescribed by the ontology and obey any further constraints imposed by an associated set of norms.

In our case study, we model an electronic ordering system. The purchasing system player *eOrder* is related to the *Customer* agent and the *Warehouse* agent. *Customer* orders products from *eOrder* by initializing the interaction. *eOrder* receives data describing the order and performs further processing which normally includes checking availability of the product in the system, customer’s solvency, invoicing, dispatching, and rejecting. If processing is successful, the system files an invoice for the purchase and subsequently dispatches it. If the product is unavailable, *eOrder* sends a request to the warehouse to increase the stock to further proceed with the order. If processing is unsuccessful, *eOrder* rejects the order.

An ontology for the purchasing system is given in Fig. 1. Agents are represented as ovals and business entities as rectangles with curved edges. Communication acts and relations are shown as rectangles, with the former differentiated by the use of an exclamation mark *!* before the act’s name.

All affordances (including agents and business entities) have a number of typed attributes, defining the kinds of states they may be in. We permit navigation through an affordance’s attributes and related affordances in the object-oriented style of the OCL.

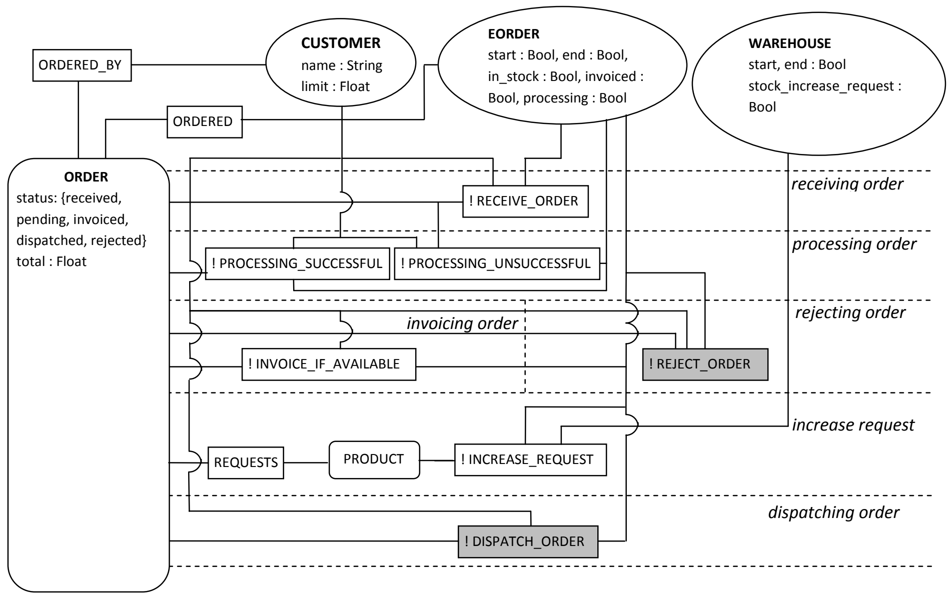


Figure 1: Example normative ontology

The system involves processes that cross the boundaries of three subsystems: a customer agent, an order processing system, and a product warehouse system. These three subsystems are represented as agents in the ontology, *Customer*, *eOrder* and *Warehouse*, respectively. By default all agents except for *Customer* contain start and end attributes. All communication acts are associated with an instance of the entity class *ORDER*, namely *oo*. The whole diagram is divided by the respective tracks using dashed lines. Each communication act or acts that occur simultaneously (fork or choice) are positioned in these tracks. Shaded communication acts are the final acts, with which the whole process terminates. The solid lines between the elements of the diagram represent the need for the information to be exchanged between them (passing arguments, defining instances for the relationships, etc.): agents, communication acts, entities and relationships.

An order is associated with its customer, defined by the *ordered_by* relationship holding between the *customer* agent and *order* entity. An order can stand in an *ordered* relationship with the *eOrder* agent, after it has been successfully processed. Communication act *!receive_order* corresponds to the initial reception of data. The *Processing* communication act further deals with the newly arrived order and checks whether the client's credit limit allows for the purchase. Namely, it checks whether the total cost of the purchase is less than the credit limit of the customer. This condition results in the following outcomes: if the credit limit is lower than the total cost of the purchase then the system rejects the order, otherwise it initiates the invoicing process (denoted by the *invoice_if_available* communication act). It does so if the stock contains enough amount of the product for the order. If not, then the system requests to increase the stock by initiating *request_increase*, which sets flag *stock_increase_request* to true. Finally, the system dispatches the order by *dispatch_order*.

Consider the communication act *!receive_order* from our example, corresponding to the initial reception of data by the order processing system. The idea that this reception can only occur over orders that are not yet processed is captured by the normative tableaux shown in (18) and (19). Both relationships and communication acts are represented as logical relations in our language, but communication acts are not used in pre-conditions, and may only be placed *after* a Deontic operator.

Communication acts often define resulting changes of state on related agents and entities. As shown in our ontology in Fig. 1, *receive_order* relates three affordances: agents *Customer* and *eOrder* and business entity *Order*, instances of which are used as arguments for this communication act. As such, this communication act should affect the following relationships *ORDERED* and *ORDERED_BY* that are involved in relating the pertinent affordances. The meaning of the behavioral norms are extracted from additional definitions, which represent their equivalent meaning. An equivalent meaning of the reception of an order entails a change of state of affairs to include a newly arrived order, the status becomes set to "received", and the system initiates the processing stage by setting its attribute to *true*. This is formalized by the norm

$$\begin{aligned}&\forall\, cc : Customer \bullet \forall\, oo : Order \bullet \forall\, e : eOrder \bullet\\ &\qquad \neg ordered\_by(oo, cc) \land \neg ordered(oo, e) \rightarrow\\ &\qquad\qquad E_e\ Ob\ \mathbf{receive\_order(oo, cc, e)} \qquad (18)\end{aligned}$$

and its definition

$$\begin{aligned}&\forall\, cc : \mathit{Customer} \bullet \forall\, oo : \mathit{Order} \bullet \\ &\qquad \forall\, e : \mathit{eOrder} \bullet \mathbf{receive\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \leftrightarrow \\ &\qquad\quad \mathit{ordered\_by}(oo, cc) \wedge \mathit{ordered}(oo, e) \wedge \\ &\qquad\qquad oo.\mathit{status} = \mathit{received} \wedge e.\mathit{processing} = \top \end{aligned} \tag{19}$$

The norm (18) and its definition (19) form a single normative tableau for the communication act *!receive_order*. Note that we have employed the conventions and assumptions just given. So determinacy is implicit, and we are quantifying over individual formulae, to indicate types of variables. However, importantly, all variables – for example, *oo* and *cc* – should be understood as denoting the same possible interpreting object in the semantics.

3 Event-B

Formal languages for specifying both computer programs and systems represent a mathematically-flavoured approach to software development which did not yet find a wide application in the industry in general and in the area of enterprise business modeling in particular. This is explained by the complexity of these languages as well as the high-level reasoning process that engineers are exposed to while developing systems. But computer programs and systems that are developed using formal methods have practically proven that use of such methods leads to final designs that are much less error-prone and behave exactly as prescribed in their initial (abstract) specifications.

In this section, we are describing a formal language which represents a recent evolution of the B-method, and is inspired by the action systems approach [28] and oriented towards specifying and reasoning about systems that behave in a concurrent and discrete manner. This language is used to define a semantically enriched and consistent model that we obtain from our requirements specifications. We further explicate the model that is obtained by virtue of the model-driven transformation from normative requirements and regard it as a target Event-B model of the enterprise system.

3.1 Background

The process of building computer systems that behave correctly, do not involve heavy post-implementation testing and maintenance efforts, and are free of impossible executions, represents a difficult task that is normally approached by initiating software development projects with the simplest and most general specifications. These specifications are more often regarded to as artefacts or models of future systems. These models are by no means executable and therefore must not be considered as systems themselves but they help us to specify the properties and the behavior of the system to be implemented. As [12] puts it, the model of a program and more generally of a complex computer system, although non-executable, allows one to clearly identify the properties of the future system and to prove that they will be present in it.

This approach is embraced by the formal language used for modeling and reasoning about systems that behave in a discrete fashion. The notion of discrete modeling is referred to as *Event-B*. The Event-B language allows one to describe and model systems that are operated as an execution of successive concurrent states. Event-B has an operational semantics based upon predicate transformer semantics, with its origin in the Floyd-Hoare Logic of Dijkstra [5] and abstract state machines [38]. The number of possible states that the system can be in is enormous and the frequency of their occurrences is very high. Therefore, the behavior of such systems is observed as continuous, but this does not change the very nature of the problem: such systems are intrinsically discrete [12].

In general, Event-B can define a number of different contexts, that serve the same role as an algebraic theory in mathematics. For example, we might define one context to describe the structure and axioms for the theory of boolean values, and another context for the structure and axioms of the natural numbers. The definition for context is as follows.

Definition 3 (Event-B context) *A well-formed Event-B context C for a set of variables is a tuple*

$$C = \langle \Sigma_C, \mathit{Var}, A_C \rangle$$

where

- Σ_C *is a signature, which is a set of collections of basic carrier set names and collections of constant names.*

- $A_C = \{A_1, ..., A_m\}$ *is a set of axioms consisting of well-formed formulas of* $WFF_\Sigma(Var)$ containing no free variables *(*$FV(A_i) = \emptyset$ *for* $i = 1, \ldots, m$*). Well-formed formula is a mathematical statement that is syntactically and semantically correct.*

We now define the form of the Event-B machines that we consider in this paper.

Definition 4 (Event-B machine) *A well-formed Event-B machine M is a tuple*

$$M = \langle N_M, C_M, Var_M, I_M, E_M \rangle$$

where

- N_M *is a unique name for the machine*
- C_M *is a context for the machine* $\langle \Sigma, Var_C, A_C \rangle$
- Var_M *is a set of* state *variables* $\{a_1, ..., a_j\} = Var_M$ *disjoint from* Var_C*. Disjointedness ensures that we can use variables from the set* Var_C *as variables to be bound by existential and universal quantifiers in formulas, while variables from* Var_M *are employed exclusively to denote mutable state variables of the machine, whose values affect and are affected by the triggering of events.*
- $I_M = \{inv_1, ..., inv_k\}$ *is a set of invariants consisting of well-formed formulas of* $WFF_\Sigma(Var_M \cup Var_C)$ *so that* $FV(inv_i) \subseteq Var_M$ *and* $BV(inv_i) \cap Var_M = \varnothing$ $(i = 1, \ldots, k)$*, where BV stands for bound variables. The last two conditions ensure all variables used from* Var_C *in each invariant is quantified, but state variables are never quantified.*
- E_M *is a set of* events*, of the form*

```
Event $e_i$ $\widehat{=}$
  WHEN $G_i$
  THEN $S_i$
  END
```

where for some $i \geqslant 0$*, each* e_i *is a unique event name,* G_i *is a set of* guard formulas *taken from* $WFF_\Sigma(Var_M \cup Var_C)$ *and* S_i *is a* substitution *taken from* $Subst_\Sigma(Var_M)$*. As with the invariants, we require that the only free variables of each guard are state variables and that state variables are never bound: that is* $FV(G_i) \subseteq Var_M$ *and* $BV(inv_i) \cap Var_M = \varnothing$ *for* $i = 1, \ldots, l$.

The machine M is written in Event-B notation as follows:

$$\begin{array}{llll}
\text{Machine } N_M & & & \\
\quad \text{Var } a_1, ..., a_j & \text{Event } e_1 \mathrel{\widehat{=}} & \ldots & \text{Event } e_l \mathrel{\widehat{=}} \\
\quad \text{Inv } inv_1, ..., inv_k & \quad \text{WHEN } G_1 & & \quad \text{WHEN } G_l \\
 & \quad \text{THEN } S_1 & & \quad \text{THEN } S_l \\
 & \quad \text{END} & & \quad \text{END}
\end{array}$$

Without loss of generality, it is assumed that all Event-B machines are defined with respect to some overall single context C and a contextual signature Σ.

4 Verification and Validation

Any Event-B specification does not guarantee correctness or consistency. We can easily write an inconsistent specification by, for example, adding axioms of the form $0 = 0$ and $0 \neq 0$ to a context that includes the usual axioms for natural numbers. Again, it is not possible to automatically detect inconsistencies in a specification. It is up to a specification expert to ultimately ensure the specification is consistent. However, given a specification S (a set of machines) of the form $S = \{M_1, \ldots, M_n\}$, a number of properties can be set that, if proven, ensure a safer, more trustworthy specification. In particular, given a specification, it is important to prove

- Well-formedness of any functional application and relational statements. Well-formedness of function application and relational assertions is taken as a *semantic* property of the model of statements, rather than being treated syntactically as in a typed formalism. However, by examining definitions of sets and set memberships given by invariants and the context of a specification, it is possible to prove that all function applications and relational assertions are well-formed. Such a proof entails that all models of the specification will always interpret function applications and relations as appropriate.

- Invariant preservation. By proving that all invariants are entailed by both guards and substitutions of all events, it is possible to show that a specification satisfies its invariants.

Because interpretations are given over set theory, these proofs can be done using the axioms of set theory. Toolkits like Rodin will automatically generate proof obligations for these conditions from a specification and allow an interactive, tactic-based approach to proving proof obligations. One of the motivations behind mapping *requirements* specifications into Event-B *design* specifications is that toolkits like Rodin will then be of assistance in improving trustworthiness of designs by generating and discharging these types of proof obligations, prior to implementation.

4.1 Soundness of Transformation

The language proposed to define our source model is somewhat more complex, due to its relational nature and conformance to an ontology. However, we need not study the full set of formulas given: we *restrict* our attention to a subset of normative formulas that correspond to the informal schema for behavioural norms. Due to its complexity, in what follows we provide its short definition.

4.1.1 Preservation of Typing Theorem for Event-B

The approach described in this paper describes a transformation mapping from normative requirements to Event-B constructs. This transformation mapping is formalized by the function ϕ. This function thoroughly defines which element of the normative specifications transforms (or maps) to which element of the formal specification. Admittedly, this is a not quite trivial transformation function. From the B research community perspective let us attempt to prove a number of desirable properties of ϕ that define how it provides an adequate semantics for MEASUR. Formal transformation of a norm is provided in [7]. The complete semantic preservation theorem and the proof that the design implementations preserve requirements are described in [8].

Before the theorem we need to define well-formed formulas.

Definition 5 (Well-formed formulas) *The* basic *well-formed formulas of Event-B with signature* Σ *over variables* Var*,* $BWFF_{\Sigma}(Var)$*, take the form*

$$S_1 \subseteq S_2$$

$$S_1 \subset S_2$$

$$e \in S$$

$$e_1 = e_2$$

$$P(e_1, e_2)$$

for any $P \in C_\Sigma$. $e, e_1, e_2 \in Term_\Sigma(Var)$, where $Term_\Sigma(Var)$ is a set of all variable terms in signatures; and $S, S_1, S_2 \in \hat{S}_\Sigma$, where $\hat{S}_\Sigma$ is a collection of all set names for Σ, which is defined as follows:

- *if $s \in S$, then $s \in \hat{S}_\Sigma$*
- *if $s_1, s_2 \in S$, then $s_1 \to s_2 \in \hat{S}_\Sigma$*
- *if $s_1, s_2 \in S$, then $s_1 \times s_2 \in \hat{S}_\Sigma$*

The well-formed formulas of Event-B with signature Σ over variables Var, $WFF_\Sigma(Var)$ are defined recursively:

- *if $F \in BWFF_\Sigma(Var)$ then $F \in WFF_\Sigma(Var)$*
- *if $F_1, F_2 \in WFF_\Sigma(Var)$ then $F_1 \wedge F_2, F_1 \to F_2, \neg F_1 \in WFF_\Sigma(Var)$*
- *if $FWFF_\Sigma(Var)$ and $x \in Var$ then $\forall x.F, \exists x.F \in WFF_\Sigma(Var)$*

Theorem 1 (Preservation of typing) *Assume REQ is a set of behavioral norm/definition pairs over an ontology O each of the form*

$$REQ = \{(G_i \to E_{b\ i}\ Ob\ A_i, A_i \leftrightarrow DEF_i) \mid i = 1, \ldots, n\} \tag{20}$$

Let S be the set of machines generated by applying a model-driven transformation function ϕ over each norm in REQ

$$S = \phi(REQ) \tag{21}$$

so that S consists of $BWFF_\Sigma(Var)$ formulas, where Var are taken from the agent and entity variables of O and Σ is taken from the signature of O.

Consider any norm/definition pair (otherwise referred to as 'tableaux' in [29]) defined as

$$N = (G \to E_{b:B\{\ldots\}}\ \mathbf{M}\ A, A \leftrightarrow DEF) \in REQ \tag{22}$$

where **M** *is a modality operator taken from deontic logic. Then*

- *Given any agent variable $\vdash_O a : A\{\ldots\}$ occurring in N, there is a machine M_a in S corresponding to a.*

- *Given any entity variable* $\vdash_O e : E\{l_1 : T_1, \ldots, l_n : T_n\}$ *occurring in REQ, we can use the notation of delimiters to denote that the following expressions are located in machine* M_b*:*

$$\ulcorner e \in E \urcorner \in M_b$$

$$\ulcorner f_1 \in E \to T_1 \urcorner \in M_b$$

$$\ldots$$

$$\ulcorner f_n \in E \to T_n \urcorner \in M_b$$

Proof: By inspection of each of the cases in the transformation. □

Definition 6 *Assume REQ is a set of behavioral norm/definition pairs, each of the form (20). Let* S *be the set of machines generated by applying* ϕ *over each norm in REQ (21). Given a model* $\mathcal{M}$ *of a* S*, we define a state* σ *of* $\mathcal{M}$ *to be a* non-intermediate state *if, and only if,* $\mathcal{M}, \sigma \models R_N = \bot$ *for any flag variable* R_N*, any norm* $N \in REQ$*. Flag variable* R_N *is a shared variable of type boolean that serves as a flag between two communicating agents.*

Because inter-agent communication is always modeled using the boolean flag variables, each of the form R_N (any norm N), a non-intermediate state denotes the state of the system that is not between stages of transmission of information from one machine to another via the shared flag variables.

Assumption 4 *We will be assuming all flag variables are initialized to be* $\bot$ *within any initial state of the transition semantics. That is, we will assume a machine always begins in a non-intermediate stage: a reasonable assumption.*

Lemma 1 *Assume REQ is a set of behavioral norm/definition pairs, each of the form (20). Let* S *be the set of machines generated by applying* ϕ *over each norm in REQ (21). Take any model* $\mathcal{M}$ *that satisfies* S*. Consider any norm/definition tableaux of the form (22). If we have a state* σ *and* σ' *such that the model transformation of the definition provides a shift in the state from* σ *to* σ'*:* $\sigma' = [\phi(DEF)]\sigma$*; then we know that*

$$\mathcal{M}, \sigma' \models \mathsf{toB}(DEF)$$

where $\mathsf{toB}(DEF)$ *is a B representation (written in Event-B) of the definition.*

Proof: By a straightforward induction over the possible forms of *DEF*, using the definition of ϕ. $\square$

We are now ready to show an important soundness property of the transformation. The intuitive meaning of a norm $G \rightarrow E_{b:B\{\ldots\}}\ Ob\ A$ is that, given G holds, the agent b *must* make A hold. Because we map G to a guard and A to an action of a particular event and b to a machine that contains the event, we would expect ϕ to preserve this intuitive meaning: that is, we would expect that whenever the guard corresponding to G is satisfied, the *machine* corresponding to b *must* always perform the action associated with A.

We prove this now, formally.

Theorem 2 *Assume REQ is a set of behavioral norm/definition pairs, each of the form (20). Let S be the set of machines generated by applying ϕ over each norm in REQ (21). Take any model $\mathcal{M}$ that satisfies S. Consider any norm/definition tableaux of the form (22). For any non-intermediate state σ of $\mathcal{M}$ such that*

$$\mathcal{M}, \sigma \models \mathsf{toB}(G)$$

there is a transition to state σ' caused by event $event_N \in M_B$ such that

$$\langle S, \sigma \rangle \overset{*}{\rightarrow} \langle S, \sigma_1 \rangle \overset{*}{\rightarrow} \ldots \overset{*}{\rightarrow} \langle S, \sigma_n \rangle \overset{M_B, event_N}{\rightarrow} \langle S, \sigma' \rangle$$

where we know that the B machine interpretation of A's definition becomes true in state σ'

$$\mathcal{M}, \sigma' \models \mathsf{toB}(DEF)$$

Proof: We use the definition of ϕ and the transitional semantics of B machines. There are two cases, depending on the form of A.

- If A is of the form $R(p, b', b)$, where $p : P\{\ldots\}$, $P\{\ldots\} \in ENTITY$, $b : B\{\ldots\}$, $B\{\ldots\} \in AGENT$, $b' : B'\{\ldots\}$, $B'\{\ldots\} \in AGENT$ and $R \in COMMACT$ (type for communication acts used in the requirements definitions) and where the two agent types are *different*, so $B\{\ldots\} \neq B'\{\ldots\}$, then we take the post-condition $E_b\ Ob\ R(p, b', b)$, in which case by the definition of ϕ we know that there is an event named $event_N$ in the machine $M_B \in S$ of the form

$$\left\ulcorner \begin{array}{l} \mathit{Event}\ \mathit{event}_N \mathrel{\widehat{=}} \\ \text{WHEN } \mathit{GUARD}_N(G)^* \\ \text{THEN } \phi_N(DEF);\ R_N := \bot \\ \text{END} \end{array} \right\urcorner \in E_{M_B} \tag{23}$$

where

$$\begin{aligned} \mathit{GUARD}_N(G)^* \equiv \mathit{GUARD}_N(G)/G' \wedge R_N = \top \equiv \\ (\mathsf{toB}(G)/G') \wedge R_N = \top \end{aligned} \quad (24)$$

and there is an event $\mathit{comEvent}_N$ in machine $M_{B'}$:

$$\begin{array}{ll} \ulcorner \mathit{Event\ comEvent}_N \mathrel{\widehat{=}} \urcorner & \\ \quad \text{WHEN } G'_N & \in E_{M_{B'}}, \\ \quad \text{THEN } R_N := \top & \\ \quad \text{END} & \end{array} \quad (25)$$

where $G' \equiv \mathit{GUARD}_N(G) \mid_{\mathit{ExtVAR}(M_{B'})} \equiv \mathsf{toB}(G) \mid_{\mathit{ExtVAR}(M_{B'})}$ and ExtVAR are a class of external variables, by virtue of which a variable sharing is implemented in Event-B.

Now, assuming a state σ such that

$$\mathcal{M}, \sigma \models \mathsf{toB}(G) \quad (26)$$

it *cannot* be the case that $\mathcal{M}, \sigma \models \mathit{GUARD}_N(G)^*$ holds by (24), because

$$\mathcal{M}, \sigma \models R_N = \bot$$

by assumption. However, by (26) it must be the case that $\mathcal{M}, \sigma \models G'_N$ (because $G'_N \equiv \mathit{GUARD}_N(G) \mid_{\mathit{ExtVAR}(M_{B'})}$, and the restricted form of a conjunctive formula should hold if the original formula holds over a state). Consequently, we know that

$$\langle S, \sigma \rangle \overset{M_{B'}, \mathit{comEvent}_N}{\rightarrow} \langle S, \sigma_1 \rangle$$

where $\sigma_1 = [R_{\mathit{call}} := \top]\sigma$. So

$$\mathcal{M}, \sigma_1 \models R_{\mathit{call}} = \top \quad (27)$$

Furthermore, because all other events are generated from norms that exclude each other's guards, σ_1 is the *only* such state that can follow from σ. By (26) and the fact that G (and so toB) do not contain any reference to R_N, we have

$$\mathcal{M}, \sigma_1 \models \mathsf{toB}(G) \quad (28)$$

But then it must be the case that

$$\mathcal{M}, \sigma_1 \models \mathit{GUARD}_N(G)^* \tag{29}$$

by (24), (27) and (28) because $\mathit{GUARD}_N(G)^* \equiv (\mathsf{toB}(G)/G') \wedge R_N = \top$. Finally, by definition of the executable semantics, we know that there is a state σ' such that

$$\langle S, \sigma_1 \rangle \overset{M_B, event_N}{\rightarrow} \langle S, \sigma' \rangle$$

where $\sigma' = [\phi(D);\ R_{call} := \bot]\sigma_1$ and so, by Lemma 1, $\mathcal{M}, \sigma' \models \mathsf{toB}(\mathit{DEF})$.

Furthermore, because all other events are generated from norms that exclude each other's guards, σ' is the *only* such state that can follow from σ_1, as required.

- The proof is very similar for the case where $R(p, b', b)$, where $p : P\{\ldots\}$, $P\{\ldots\} \in \mathit{ENTITY}$, $b : B\{\ldots\}$, $B\{\ldots\} \in \mathit{AGENT}$, $b' : B'\{\ldots\}$, $B'\{\ldots\} \in \mathit{AGENT}$ and $R \in \mathit{COMMACT}$ and where the two agent types are *the same*, so $B\{\ldots\} = B'\{\ldots\}$. The main difference is that the transitions are occurring within the same machine, rather than out of it – however this does not affect the argument over the transition semantics, which is essentially the same as the dual machine case above. $\Box$

The analogous case for the Pe modality holds similarly.

4.2 Validation of the Specification

One of the requirements of formal specifications is to prove that they are internally consistent. We are particularly interested in checking whether assignment statements of events preserve the guards and invariants of the generated Event-B model. The Event-B language (similarly to B-method) defines proof obligations for substitutions (or events). Discharging these proof obligations presents a form of specification validation.

For the purposes of validation of our specifications we have used the Rodin platform. The Rodin platform is an extensible application for refinements and mathematical proofs in Event-B, which is based on the Eclipse integrated development environment. Figs. 2 and 3 show our Event-B specifications in the Rodin platform.

Let us now illustrate the *proving* perspective of the Rodin platform. Table 1 shows that 12 proof obligations were proven automatically. Formal designs with proof obligations that are proven automatically are said to be well-specified. For example, proof obligation of event *receive_order* is the following predicate:

$$\begin{array}{ll} (1) & status \in Order \rightarrow STATUS \wedge \\ (2) & receive\_order_{call} = \top \wedge \neg oo \in ordered \\ & \Rightarrow \\ (3) & status \lhd \{oo \mapsto received\} \in Order \rightarrow STATUS \end{array} \tag{30}$$

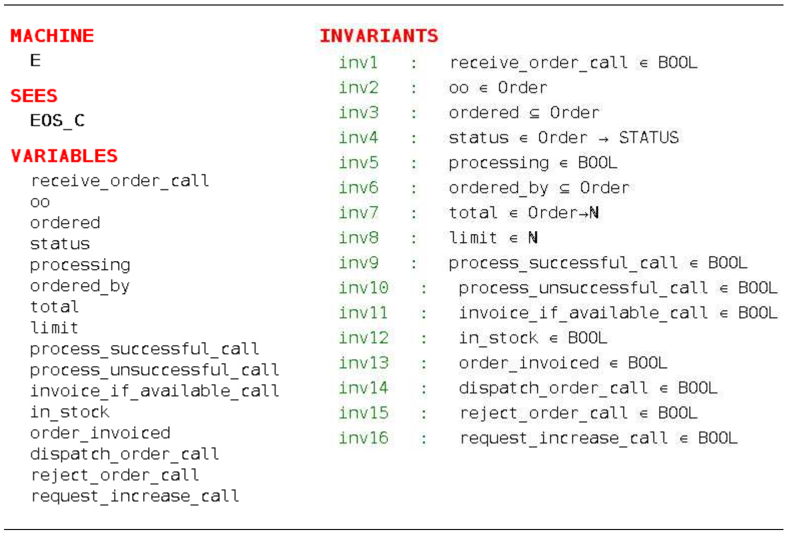
This predicate means that given invariant (1) and guards (2) hold, the assignment expression $status(oo) := received$ applied to the invariant must also hold (3). According to Table 1, all 12 generated proof obligations for machine e were proven automatically. Machines cc and w do not have expressions that change values of state variables and therefore no proof obligations were generated for these machines. It is important to define specifications which provide proof obligations that are easy for the theorem provers to discharge. Otherwise the specifications have to be rewritten in a different manner which helps simplify the proving process.

Table 1: Proof obligations

Machine	Proof Obligations	Automatic	Interactive
cc	0	0	0
e	12	12	0
w	0	0	0
Total	12	12	0

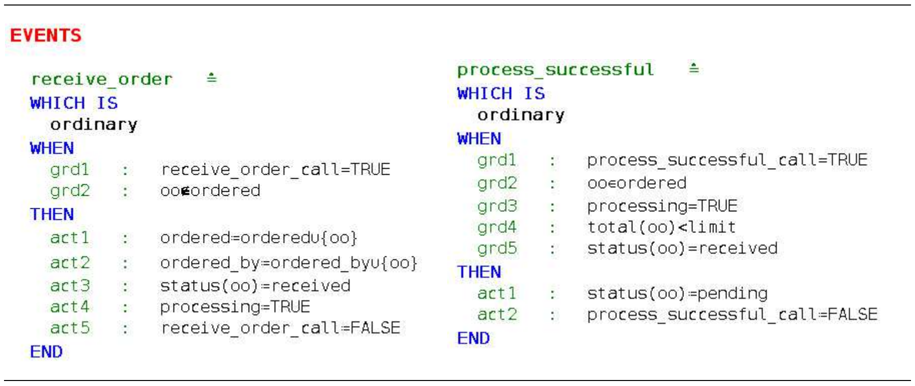
5 Related Work

There are a number of related approaches for enriching workflow models [22, 23]. In one of them the authors describe a transformation from BPEL4WS to full OWL-S ontology to provide missing semantics in BPEL4WS. BPEL4WS does not present meaning of a business process so that business process can be automated in a computer understandable way [34]. They are using overlap which exists in the conceptual models of BPEL4WS and OWL-S and perform mapping from BPEL4WS to OWL-S to avoid this lack of semantics.



```
MACHINE                          INVARIANTS
  E                                inv1   :  receive_order_call ∈ BOOL
SEES                               inv2   :  oo ∈ Order
  EOS_C                            inv3   :  ordered ⊆ Order
VARIABLES                          inv4   :  status ∈ Order → STATUS
  receive_order_call               inv5   :  processing ∈ BOOL
  oo                               inv6   :  ordered_by ⊆ Order
  ordered                          inv7   :  total ∈ Order→ℕ
  status                           inv8   :  limit ∈ ℕ
  processing                       inv9   :  process_successful_call ∈ BOOL
  ordered_by                       inv10  :  process_unsuccessful_call ∈ BOOL
  total                            inv11  :  invoice_if_available_call ∈ BOOL
  limit                            inv12  :  in_stock ∈ BOOL
  process_successful_call          inv13  :  order_invoiced ∈ BOOL
  process_unsuccessful_call        inv14  :  dispatch_order_call ∈ BOOL
  invoice_if_available_call        inv15  :  reject_order_call ∈ BOOL
  in_stock                         inv16  :  request_increase_call ∈ BOOL
  order_invoiced
  dispatch_order_call
  reject_order_call
  request_increase_call
```

Figure 2: Variables and invariants of machine *e* represented in the Rodin platform



```
EVENTS
                                         process_successful  ≙
  receive_order  ≙                       WHICH IS
  WHICH IS                                 ordinary
    ordinary                             WHEN
  WHEN                                     grd1  :  process_successful_call=TRUE
    grd1  :  receive_order_call=TRUE       grd2  :  oo∈ordered
    grd2  :  oo∉ordered                    grd3  :  processing=TRUE
  THEN                                     grd4  :  total(oo)<limit
    act1  :  ordered≔ordered∪{oo}          grd5  :  status(oo)=received
    act2  :  ordered_by≔ordered_by∪{oo}  THEN
    act3  :  status(oo)≔received           act1  :  status(oo)≔pending
    act4  :  processing≔TRUE               act2  :  process_successful_call≔FALSE
    act5  :  receive_order_call≔FALSE    END
  END
```

Figure 3: Events *receive_order* and *process_successful* of machine *e* represented in the Rodin platform

Another work is using the example of BPEL processes which should be converted to semantically enriched specifications. All data (stored in process models) must be augmented by references to ontologies [24]. They refer this augmentation to as *ontological lifting* because input business processes must be expressed using richer constructs provided by ontologies. However, from the perspective of web services, systems support only part of the process space representation which is reduced to the patterns of message exchange (choreography) and the control and data flow in the combination of multiple Web services (orchestration) [25].

Others [31] advocate the idea of ensuring the correctness of a workflow by making protocol specifications data-aware through expressing actual data content rather than message names. Workflow validation cannot be complete unless this abstraction is eliminated. They present CTL-FO$^+$ tool, an extension over Computation Tree Logic that includes first-order quantification on state variables in addition to temporal operators, and which is adequate for expressing data-aware constraints.

There is also a variety of research directed towards semantic enriching of Petri net business processes. The authors were proposing to enrich the semantics of Petri nets by combining it with OWL language. Representing Petri nets in combination with OWL is a way to make data computer-interpretable for flexibility, ease of integration and significant level of automation of loosely coupled business processes [3]. In this work, authors are trying to define Petri net models using OWL framework which entails horizontal way of integration. In other words, they are defining semantic metadata for business processes described by Petri nets.

There were also several results on integrating Petri nets and Z. In one of them [37], authors describe a thorough integration definition of Petri nets and Z which results in so called PZ nets for specifying concurrent and distributed systems. In this work, Petri nets are used to define the overall structure, control flow and dynamic properties and Z is applied for specifying tokens, labels and constraints of the system. This result is based on the previous more preliminary work on integrating Petri nets and Z [4]. Another work [16] have used Z to specify certain aspects of restricted hierarchical coloured Petri nets. Namely, the authors have used Z schemas to define the metamodel of a hierarchical colored Petri net and operation for specifying the transitions in a specific colored Petri net.

6 Conclusion and Future Work

Business process management is an evolving area of the enterprise. There are a number of problems in the enterprise where modelling workflows is of major importance. These problems can be related to specifying a new business model or reconfiguring existing workflows to adapt the latter to the changes that may happen. Developing a new business model with dynamically interacting entities and agents or working with the existing system requires skills, knowledge (both technical and non-technical), and relevant modeling tools.

In seeking to address this concern, this paper has described our formal approach of mapping the two levels of abstraction in order to provide a connecting bridge from requirements to design. Section 2 has provided a background information on fundamentals of speech act, normative relationships and the philosophy behind it. It also described our CIM, which is represented as a combination of the MEASUR ontologies and the theory of normative positions limited to the kinds of norms used in MEASUR. The MEASUR ontologies are comprised of entities, agents, relationships between them, and communication acts, thus representing a static information about the model. Our set of norms in turn constrains the behaviour of agents in our ontology by specifying what agent is obliged / permitted / not permitted to perform action under certain conditions.

Continuing on defining formalisms, we have described the Event-B language in Section 3 as a PIM and provided definitions of a consistent B machine. The paper then proceeds with the idea of component-based modeling of Event-B and explains the choice of the language of Event-B.

In order to define a mapping function from normative requirements to Event-B design, Section 4 has described a detailed definition of this mapping by relating different requirements components to their appropriate Event-B entities. With respect to its complexity, the mapping is identified as a nontrivial transformation which generates modular, component-based Event-B models from flat normative requirements. This section included 4.1 with the transformation soundness proofs, in which we have shown that the normative ontologies and Event-B machines are compatible and that Event-B machines behave exactly as prescribed by our norms. In other words, our transformation guarantees that the generated Event-B models will not commit to impossible actions, behavior that is not specified in our requirements.

The contributions described in this paper are substantial and cutting-edge, thereby advancing the state-of-the-art of BPM. We have defined a consistent B machine and operational semantics for Event-B (referred as a platform independent model in our approach). The operational semantics was formally defined by an Event-B specification, models and interpretations and the state of a machine. We have also defined a notion of state transition by virtue of describing generalised substitution and transitional semantics, and we have provided a definition of invariant preservation and model satisfaction. Our approach is one of the few attempts to formally embed semantics to the requirements specification via MDA and provide an operational semantics of Event-B to the behavioral part of the normative specifications.

By virtue of implementing a transformation and generating an Event-B modules we have provided a formal semantics for the normative ontologies with correctness and validity in mind. The generated Event-B specification are further checked for consistency and the respective proof obligations are discharged. These modular specifications can further be used in refining to more detailed implementations.

As a future work we argue that our method can potentially be used to automatically detect faults and inconsistencies in the normative requirements. The concept is illustrated in Fig. 4. According to this diagram, if

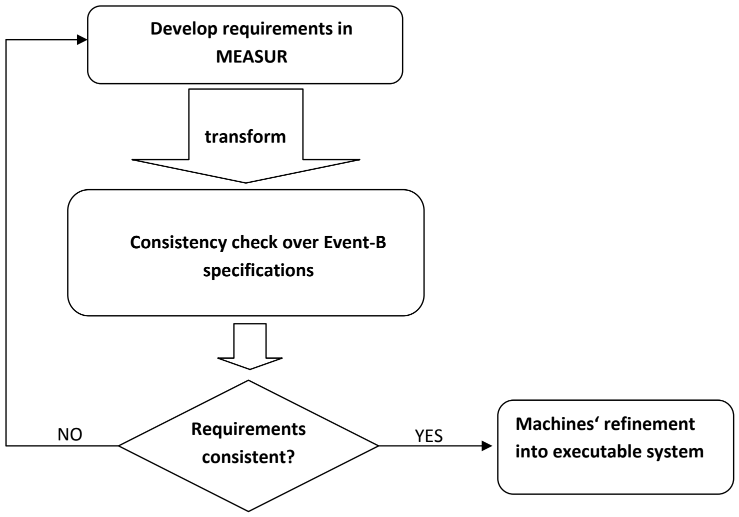

Develop requirements in MEASUR
transform
Consistency check over Event-B specifications
NO
Requirements consistent?
YES
Machines' refinement into executable system

Figure 4: Roundtrip engineering concept for our transformation

the requirements are consistent, then the generated Event-B models from these requirements can be used for further refinement stages in order to obtain an executable system. In the cases of faults in the requirements, we plan to implement "roundtrip engineering". If there are inconsistencies in the specifications, the generated Event-B design models will have these inconsistencies embedded, which can be detected by the Rodin platform. The requirements should be revisited and checked in order to eliminate these faults. This is a subject of our future scientific inquiry.

References

[1] A. Jones, M. Sergot. Formal Specification of Security Requirements using the Theory of Normative Positions. In *Proceedings of the Second European Symposium on Research in Computer Security*, pages 103–121, 1992. doi:10.1007/BFb0013894.

[2] A. Jones, M. Sergot. On the Characterization of Law and Computer Systems: The Normative Systems Perspective. In Eds. John-Jules Meyer and Roel Wieringa *Deontic Logic in Computer Science: Normative System Specification*, pages 275–307, Wiley, 1993.

[3] A. Koschmider, A. Oberweis. Ontology Based Business Process Description. In Michele Missikoff and Antonio De Nicola, eds. *Proceedings of the Open Interoperability Workshop on Enterprise Modelling and Ontologies for Interoperability*, 2005. http://www.ceur-ws.org/Vol-160/paper12.pdf

[4] C.Y. Kan, X. He. High Level Algebraic Petri nets. *Information & Software Technology*, 37(1):23–30, 1995. doi:10.1016/0950-5849(94)00438-X.

[5] E. Dijkstra. Guarded commands, nondeterminacy and formal derivation of program. *Communications of the ACM*, 18(8):453–457, 1975. doi:10.1145/360933.360975.

[6] H.L.R. Finch. *Wittgenstein: The Later Philosophy. An Exposition of the Philosophical Investigations.* Humanities Press, 1977.

[7] I. Poernomo, T. Umarov. A Mapping from Normative Requirements to Event-B to Facilitate Verified Data-Centric Business Process Management. In *Proceedings of 4th IFIP TC 2 Central and East European Conference of Software Engineering Techniques, CEE-SET* 2009. Lecture Notes in Computer Science 7054, *Advances in Software Engineering Techniques*, pages 136 – 149. Springer, 2009. doi:10.1007/978-3-642-28038-2_11.

[8] I. Poernomo, T. Umarov. Implementing a Sound Mapping from Normative Requirements to Event-B Design Blueprints Using MDA. In *Proceedings of the 2012 International Conference on Software Engineering Research & Practice*. CSREA Press, pages 145–151, 2012.

[9] *Information and Knowledge Management in Complex Systems.* Proceedings of The 16th Conference on Organisational Semiotics in Organisations K. Liu, K. Nakata, W. Li, D. Galarreta, eds. (2015) ISBN:978-3-319-16273-7 (Print) 978-3-319-16274-4 (Online). doi:10.1007/978-3-319-16274-4.

[10] I. Pörn. *The Power of Logic.* Blackwell, Oxford, 1970.

[11] I. Pörn. *Action Theory and Social Science: Some Formal Models.* D. Reidel, Dordrecht, 1977.

[12] J.-R. Abrial, C. Métayer, L. Voisin. *Event-B Language*. RODIN Deliverable 3.2, 2005. http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf.

[13] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering* Cambridge University Press, 2010.

[14] J. Searle. *Speech Act: An Essay in the Philosophy of Language.* Cambridge University Press, 1969.

[15] J. Searle. *Expression and Meaning: Studies in the Theory of Speech Acts (essay collection).* Duke University Press, 1979.

[16] J. Vautherin. Parallel Systems Specifications with Coloured Petri nets and Algebraic Specifications. In *Advances in Petri Nets 1987, the 7th European Workshop on Applications and Theory of Petri nets*, pages 293-308, 1987. doi:10.1007/3-540-18086-9_31.

[17] K. Liu. *Semiotics in Information Systems Engineering.* Cambridge University Press, 2000.

[18] K. Liu, R.J. Clarke, P.B. Andersen, R.K. Stamper. *Coordination and communication using signs: Studies in organisational semiotics* Kluwer Academic Publishers, 2002.

[19] K. Liu, R.J. Clarke, P.B. Andersen, R.K. Stamper. *Organizational semiotics: Evolving a science of information systems* Kluwer Academic Publishers, 2002.

[20] L. Lindahl. *Position and Change - A Study in Law and Logic*. D. Reidel Publishing Company, 1977.

[21] L. Lindahl. Stig Kanger's Theory of Rights. *Logic, Methodology and Philosophy of Science* IX, pages 889–911, 1994.

[22] M. Ahtisam, S. Auer, M. Böttcher. From BPEL4WS process model to full OWL-S ontology. In *Proceedings of the Third European Semantic Web Conference*, 2006.

[23] M. Born, F. Dörr, I. Weber. User-friendly semantic annotation in business process modeling. In *Web Information Systems Engineering WISE 2007 Workshops*, Lecture Notes in Computer Science, pages 260–271, Springer, 2007. doi:10.1007/978-3-540-77010-7_25.

[24] M. Hepp, D. Roman. An Ontology Framework for Semantic Business Process Management. In *Proceedings of the 8th International Conference Wirtschaftsinformatik Universitaetsverlag Karlsruhe*, pages 423-440, 2007.

[25] M. Hepp, F. Leymann, J. Domingue, A. Wahler, D. Fensel. Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In *Proceedings of the IEEE ICEBE 2005*, pages 535-540, 2005. doi:10.1109/ICEBE.2005.110.

[26] P. McNamara. *Deontic Logic*. Stanford Encyclopedia of Philosophy, 2006.

[27] P. Navarro, J. Rodriguez. *Deontic Logic and Legal Systems*. Cambridge Introductions to Philosophy and Law, 2014

[28] R.-J. Back. Refinement Calculus, Part II: Parallel and Reactive Programs. In Jaco Willem de Bakker, Willem-Paul de Roever, and Grzegorz Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume 430 of Lecture Notes in Computer Science, pages 67–93. Springer-Verlag, 1990. doi:10.1007/3-540-52559-9_61.

[29] R. Kamun, A. Omarov, T. Umarov. Business Requirements: Normative Approach to Behavior Modeling. In *Proceedings of the 4th International Symposium on Business Modeling and Software Design*, pages 186–195, 2014. doi:10.5220/0005425901860195.

[30] R. Wolfgang. *Petri nets: An introduction.* Springer, 1985.

[31] S. Halle, R. Villemaire, O. Cherkaoui, B. Ghandour. Model Checking Data-aware Workflow Properties with CTL-FO$^+$. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 267–278, 2007. doi:10.1109/EDOC.2007.36.

[32] S. Kanger. Law and Logic. *Theoria*, 38(3):105–132, 1972. doi:10.1111/j.1755-2567.1972.tb00928.x.

[33] S. Kanger. On Realization of Human Rights. In *Action, Logic and Social Theory*, 1985.

[34] T. Andrews et al. *Business Process Execution Language for Web Services (BPEL4WS).* Technical Report, BEA Systems, International Business Machines Corporation, Microsoft, SAP AG, Siebel Systems, 2003. http://xml.coverpages.org/BPELv11-May052003Final.pdf

[35] W.N. Hohfeld. *Fundamental Legal Conceptions As Applied in Judicial Reasoning.* Dartmouth Publishing, 1964.

[36] W. van der Aalst, K. van Hee. *Workflow Management: Models, Methods, and Systems.* The MIT Press, 2002.

[37] X. He. PZ Nets: A Formal Method Integrating Petri nets with Z. *Information & Software Technology*, 43(1):1–18, 2001. doi:10.1016/S0950-5849(00)00134-8.

[38] Y. Gurevich, P. Kutter, M. Odersky, L. Thiele. *Abstract State Machines - Theory and Applications.* volume 1912 of Lecture Notes in Computer Science, Springer, 2000. doi:10.1007/3-540-44518-8.