Arthur H.M. ter Hofstede
Massimo Mecella
Sebastian Sardina
Andrea Marrella (Eds.)

# Knowledge-intensive Business Processes

1st International Workshop, KiBP 2012
Proceedings

June 15, 2012
Rome, Italy

## Preface

Nowadays, Workflow Management Systems (WfMSs) and, more generally, Process Management Systems (PMSs) and Process-aware Information Systems (PAISs), are widely used to support many human organizational activities, ranging from well-understood, relatively stable and structured processes (supply chain management, postal delivery tracking, etc.) to processes that are more complicated, less structured and may exhibit a high degree of variation (healthcare, emergency management, etc.). Every aspect of a business process involves a certain amount of knowledge which may be complex depending on the domain of interest. The adequate representation of this knowledge is determined by the modeling language used. Some processes behave in a way that is well understood, predictable and repeatable: the tasks are clearly delineated and the control flow is straightforward. Recent discussions, however, illustrate the increasing demand for solutions for knowledge-intensive processes, where these characteristics are less applicable.

The actors involved in the conduct of a knowledge-intensive process have to deal with a high degree of uncertainty. Tasks may be hard to perform and the order in which they need to be performed may be highly variable. Modeling knowledge-intensive processes can be complex as it may be hard to capture at design-time what knowledge is available at run-time. In realistic environments, for example, actors lack important knowledge at execution time or this knowledge can become obsolete as the process progresses. Even if each actor (at some point) has perfect knowledge of the world, it may not be certain of its beliefs at later points in time, since tasks by other actors may change the world without those changes being perceived. Typically, a knowledge-intensive process cannot be adequately modeled by classical, state of the art process/workflow modeling approaches. In some respect there is a lack of maturity when it comes to capturing the semantic aspects involved, both in terms of representing them and in terms of reasoning about them. The main focus of the 1st International Workshop on Knowledge-intensive Business Processes (KiBP 2012) was investigating how techniques from different fields, such as Artificial Intelligence (AI), Knowledge Representation (KR), Business Process Management (BPM), Service Oriented Computing (SOC), etc., can be combined with the aim of improving the modeling and the enactment phases of a knowledge-intensive process. The 1st International Workshop on Knowledge-intensive Business Processes (KiBP 2012) was held as part of the program of the 2012 Knowledge Representation & Reasoning International Conference (KR 2012) in Rome, Italy, in June 2012.

The workshop was hosted by the Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti of Sapienza Universitá di Roma, with financial support of the University, through grant 2010-C26A107CN9 TESTMED, and the EU Commission through the projects FP7-258888 Greener Buildings and FP7-257899 Smart Vortex.

This volume contains the 5 papers accepted and presented at the workshop. Each paper was reviewed by three members of the internationally renowned Program Committee. In addition, a further paper was invited for inclusion in the workshop proceedings and for presentation at the workshop. There were two keynote talks, one by Marlon Dumas (Institute of Computer Science, University of Tartu, Estonia) on “Integrated Data and Process Management: Finally?” and the other by Yves Lespérance (Department of Computer Science and Engineering, York University, Canada) on “A Logic-Based Approach to Business Processes Customization” completed the scientific program. We would like to thank all the Program Committee members for their valuable work in selecting the papers, Andrea Marrella for his valuable work as publication and publicity chair of the workshop, and Carola Aiello and the consulting agency Consulta Umbria for the organization of this successful event.

June 15, 2012
Rome, Italy

Arthur H.M. ter Hofstede
Massimo Mecella
Sebastian Sardina

## Organizing Committee

### Program Chairs

| | |
|---|---|
| Arthur H.M. ter Hofstede | Queensland University of Technology |
| Massimo Mecella | Sapienza - University of Rome |
| Sebastian Sardina | RMIT University |

### Proceedings Chair

| | |
|---|---|
| Andrea Marrella | Sapienza - University of Rome |

## Program Committee

| | |
|---|---|
| Marco Aiello | University of Groningen |
| Diego Calvanese | Free University of Bozen-Bolzano |
| Fabio Casati | University of Trento |
| Florian Daniel | University of Trento |
| Massimiliano De Leoni | Eindhoven University of Technology |
| Riccardo De Masellis | Sapienza - University of Rome |
| Claudio Di Ciccio | Sapienza - University of Rome |
| Christoph Dorn | University of California |
| Marlon Dumas | University of Tartu |
| Marie-Christine Fauvet | Joseph Fourier University of Grenoble |
| Paolo Felli | Sapienza - University of Rome |
| Hector Geffner | Pompeu Fabra University of Barcelona |
| Marcello La Rosa | Queensland University of Technology |
| Yves Lespérance | York University |
| Niels Lohmann | University of Rostock |
| Marco Montali | Free University of Bozen-Bolzano |
| Selmin Nurcan | Panthéon - Sorbonne University |
| Manfred Reichert | University of Ulm |
| António Rito Silva | Technical University of Lisbon |
| Alessandro Russo | Sapienza - University of Rome |
| Rainer Schmidt | University of Aalen |
| Pnina Soffer | University of Haifa |
| Roman Vaculín | IBM Research |
| Barbara Weber | University of Innsbruck |
| Mathias Weske | University of Potsdam |
| Petia Wohed | Stockholm University |

### Additional Reviewers

| | |
|---|---|
| Sergey Smirnov | University of Potsdam |

## Table of Contents

# Integrated Data and Process Management: Finally?

Marlon Dumas

University of Tartu, Estonia
marlon.dumas@ut.ee

**Abstract.** Contemporary information systems are generally built on the principle of segregation of data and processes. Data are modeled in terms of entities and relationships while processes are modeled as chains of events and activities. This situation engenders an impedance mismatch between the process layer, the business logic layer and the data layer. We discuss some of the issues that this impedance mismatch raises and analyze how and to what extent these issues are addressed by emerging artifact-centric process management paradigms.

## 1 The Data Versus Process Divide

Data management and process management are both well-trodden fields – but each in its own way. Well-established data analysis and design methods allow data analysts to identify and to capture domain entities and to refine these domain entities down to the level of database schemas in a seamless and largely standardized manner. Concomitantly, database systems and associated middleware enable the development of robust and scalable data-driven applications, while contemporary packaged enterprise systems support hundreds of business activities on top of shared databases.

In a similar vein, well-documented and proven process analysis and design methods allow process analysts to identify and to capture process models at different levels of abstraction, ranging from high-level process models suitable for qualitative analysis and organizational redesign down to the level of executable processes that can be deployed in Business Process Management Systems (BPMS).

But while data management and process management are each well supported by their own body of mature methods and tools, these methods and tools are at best loosely integrated. For example, when it comes to accessing data, BPMS typically rely on request-response interactions with database applications or packaged enterprise systems. Typically, data fetched from these systems are copied into the "working memory" of the BPMS. The data in this working memory are then used to evaluate business rules relevant to the execution of the process, and to orchestrate both manual and automated work. But the burden of synchronizing the working data maintained by the BPMS with the data maintained by the underlying systems is generally left with the developers.

More generally, the "data vs. process" divide leads to an impedance mismatch between the data layer, the business logic layers and the process layer, which in the long run, hinders on the coherence and maintainability of information systems. In particular, the data vs. process divide has the following effects:

– *Process-related and function-related data redundancy*. The BPMS maintains data about the *state* of the process, since these data are needed in order to enable the system to schedule tasks, react to events and to evaluate predicates attached to decision points in the process. On the other hand, data entities manipulated by the process are stored in the database(s) underpinning the applications with which the BPMS interacts. Hence, the state of the entities is stored both by the BPMS and by the underlying applications. In other words, data are managed redundantly at the database layer and at the process layer, thereby adding development and maintenance complexity.

– *Business rules fragmentation and redundancy*. Some business rules are encoded at the level of the business process, others in the business logic layer (e.g. using a business rules engine) and others in the database (in the form of triggers or integrity constraints). Worst, some rules are encoded at different levels depending on the type of rule and the data involved. This fragmentation and redundancy hampers maintainability and potentially leads to inconsistencies.

The effects of this mismatch are perhaps less apparent when a one-to-one mapping exists between the instances of a given process and the entities of a given entity type. This is the case for example of a typical invoice handling process where one process instance (also called a “case”) corresponds exactly to one invoice. In this context, the state of a process instance maps neatly to the state of an entity. Ergo, the data required by the process, for example when evaluating branching conditions, is restricted to the data contained in the associated entity (i.e. the invoice in this example) and possibly to the state of other entities within the *logical horizon* [5] of the said entity – e.g. the Purchase Order (PO) associated to the invoice. Accordingly, collecting the data required for evaluating business rules required by this process is relatively simple, while synchronizing the state of the process instance with the state of its associated entity (at the business logic and data layers) does not pose a major burden.

The impedance mismatch however becomes much more evident when this one-to-one correspondence between processes and entities does not hold. Consider for example a shipment process where a single shipment may contain products for multiple customers, ordered by means of multiple purchase orders (POs) and invoiced by means of multiple invoices – perhaps even multiple POs and multiple invoices per customer involved. Furthermore, consider the case where the products requested in a given PO are not necessarily sent all in a single shipment, but instead may be spread across multiple shipments. In this setting, the effects of a customer canceling a PO are not circumscribed to one single instance of the shipment process. Similarly, the effects of a delayed shipment are not restricted to single PO. Consequently, business rules related for example to cancellation penalties, compensation for delayed deliveries or prioritization of shipments become considerably more difficult to capture, to maintain and to reason about, as exemplified in numerous case studies [1, 9, 8, 3]. Traditional process management approaches quickly hit their limit when dealing with such processes. The outcome of this limitation is that a significant chunk of the “process logic” has to be pushed down to the business logic layer (e.g. in the form of business rules) – which essentially voids the benefits of adopting a structured process management approach supported by a BPMS.

Service-oriented architectures (SOAs) facilitate the inter-connection of applications and application components. Their emergence has greatly facilitated the integration of data-driven and process-driven applications. SOAs have also enabled packaged enterprise software vendors to "open the box" by providing standardized programmatic access to the vast functionality of their systems. But per se, SOAs do not address the problem of data and process integration, since data-centric services and process-centric services are still developed separately using different methods. A case in point is Thomas Erl's service-oriented design method [4], which advocates that process-centric services should be strictly layered on top of data-centric (a.k.a. entity-centric) services. Erl's approach consists of two distinct methods for designing process-centric services and entity-centric services. This same principle permeates in many other service-oriented design methods [7]. Such approaches do not address the issues listed above. Instead, they merely reproduce the data versus process divide by segregating data-centric services and process-centric services.

## 2 The Artifact-Centric Process Management Paradigm

This talk discusses emerging approaches that aim at addressing the shortcomings of the traditional data versus processes divide. In particular, the keynote discusses the emerging artifact-centric process management paradigm [1, 2] and how this paradigm, in conjunction with service-oriented architectures and associated platforms, enable higher levels of integration and higher responsiveness to process change.

Mainstream process modeling notations such as BPMN can be thought as being *activity-centric* in the sense that process models are structured in terms of flows of events and activities. Modularity is achieved by decomposing activities into subprocesses. Data manipulation is captured either by means of global variables defined within the scope of a process or subprocess, or by means of conceptually passive *data objects* that are created, read and/or updated by the events and activities in the process. In contrast, the database applications and/or enterprise systems on top of which these processes execute are usually structured in terms of objects that encapsulate data and/or behavior. This duality engenders the above-mentioned impedance mismatch between the process layer and the business logic and data layers.

In contrast, *artifact-centric* process modeling paradigms aim at conceptually integrating the process layer, the business logic and the data layer. Their key tenet is that business processes should be conceived in terms of collections of *artifacts* that encapsulate data and have an associated lifecycle. Transitions between these states in this lifecycle are triggered by *events* coming from human actors, modules of an enterprise system (possibly exposed as services) and possibly other artifacts, thus implying that artifacts are inter-linked. In this way, the state of the process and the state of the entities are naturally maintained "in sync" and business processes are conceived as network of inter-connected artifacts that may be connected according to N-to-M relations, thus allowing one to seamlessly capture rules spanning across what would traditionally be perceived to be multiple process instances.

The talk also discusses ongoing efforts within the Artifact-Centric Service Interoperation (ACSI) project[-2]. This project aims at combining the artifact-centric process management paradigm with SOAs in order to achieve higher levels of abstraction during business process integration across organizational boundaries. The key principle of

[-2] `http://www.acsi-project.eu/`

the ACSI project is that processes should be conceived as systems of artifacts that are bound to services. The binding between artifacts and services specifies where should the data of the artifact be pushed to, or where it should be pulled from, and when. In the ACSI approach, process developers do not reason in terms of tasks that are mapped to request-response interactions between a process and the underlying systems. Instead, they reason in terms of artifacts, their lifecycles, operations and associated data. Artifact lifecycles are captured based on a meta-model – namely Guard-Stage-Milestone (GSM) – that allows one to capture behavior, data querying and manipulation in a unified framework [6].

Upon this foundation, the ACSI project is building a proof-of-concept platform that supports the definition and execution of artifact-centric business processes. Challenges addressed by ACSI include the problem of reverse-engineering artifact systems from enterprise system logs – for the purpose of legacy systems migration – and the verification of artifact-centric processes, which by nature are infinite-state systems due to the tight integration of processes and data.

**Acknowledgments.** This paper is the result of collective discussions within the ACSI project team. Thanks especially to Rick Hull for numerous discussions on this topic. The ACSI project is funded by the European Commission's FP7 ICT Program.

## References

1. Kamal Bhattacharya, Nathan S. Caswell, Santhosh Kumaran, Anil Nigam, and Frederick Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.
2. David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
3. Marlon Dumas. On the convergence of data and process engineering. In *Proc. of the 15th International Conference on Advances in Databases and Information Systems (ADBIS), Vienna, Austria*, pages 19–26. Springer, September 2011.
4. Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005.
5. P. Feldman and D. Miller. Entity model clustering: Structuring a data model by abstraction. *The Computer Journal*, 29(4):348360, 1986.
6. Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Noi Sukaviriya, and Roman Vaculín. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proceedings of the Fifth ACM International Conference on Distributed Event-Based Systems (DEBS), New York, NY, USA*, pages 51–62. ACM, July 2011.
7. Thomas Kohlborn, Axel Korthaus, Taizan Chan, and Michael Rosemann. Identification and analysis of business and software services - a consolidated approach. *IEEE Transactions on Services Computing*, 2(1):50–64, 2009.
8. Vera Künzle and Manfred Reichert. Philharmonicflows: towards a framework for object-aware process management. *Journal of Software Maintenance*, 23(4):205–244, 2011.
9. Guy Redding, Marlon Dumas, Arthur H. M. ter Hofstede, and Adrian Iordachescu. A flexible, object-centric approach for business process modelling. *Service Oriented Computing and Applications*, 4(3):191–201, 2010.

# A Logic-Based Approach to Business Process Customization

Yves Lespérance

Department of Computer Science and Engineering,
York University, Toronto, Canada
lesperan@cse.yorku.ca

**Abstract.** In this invited lecture, I will present a logic-based approach to modeling and engineering processes that arose from work in AI. The approach is based on a logical framework for modeling dynamic domains called the Situation Calculus. It also uses a language called ConGolog for specifying complex processes on top of the Situation Calculus. By using such a logical framework we can provide clear formal characterizations of problems that arise in the area of business process design and management. Available automated reasoning techniques can also be used to analyze and synthesize processes. After introducing the framework, I will discuss how one can use it to model process customization, where one customizes a generic process to satisfy certain constraints required by a client. I will show how we can allow for uncontrollable actions by the process, and then define a notion of maximally permissive supervisor for such a process, i.e., a supervisor that constrains the process as little as possible, while ensuring that the desired constraints are satisfied. We have shown that such a maximally permissive supervisor always exist and is unique. Finally, I will briefly discuss how one can use the framework to model the problem of process orchestration, where one wants to orchestrate a set of available services to produce a desired process.

# Automatic Detection of Business Process Interference

N.R.T.P. van Beest[1], E. Kaldeli[2], P. Bulanov[2], J.C. Wortmann[1], and A. Lazovik[2]

[1] Department of Business & ICT, Faculty of Economics and Business, University of Groningen
Nettelbosje 2, 9747 AE Groningen, The Netherlands
[2] Distributed Systems Group, Johann Bernoulli Institute, University of Groningen, Nijenborgh 9, 9747 AG, The Netherlands

**Abstract.** Today's organizations are characterized by long-running distributed business processes, which involve different stakeholders and share common resources. One of the main challenges posed in such a highly distributed setting comes from the interference between different processes that are running in parallel. During execution of a business process, a data modification caused by some external process may lead to erroneous and undesirable business outcomes. In order to address this problem, we propose to annotate business processes with dependency scopes, which cover critical sections of the process. Erroneous execution can be prevented by executing intervention processes, which are triggered at runtime. However, for complex processes with a large number of activities and many interactions with the environment, the manual specification of the appropriate critical sections can be particularly time-consuming and error-prone. To overcome this limitation, we present an algorithm for automating the discovery of critical sections. The proposed approach is applied on a real case-study of a BP from the Dutch e-Government.

## 1 Introduction

Modern private and public organizations are moving from traditional, proprietary and locally managed Business Process Management Systems (BPMS) to BPMS where more and more tasks are outsourced to third party providers and resources are shared among different stakeholders. Often, this is realized by the emergent paradigms such as Service Oriented Computing (SOC) and cloud computing. As a result, business processes (BPs) can no longer be considered in isolation, since data can be simultaneously accessed and modified by different external processes. Disregarding the interdependencies with external actors and other processes may lead to inconsistent situations, potentially resulting in undesirable business outcomes. The situation where undesirable business outcomes are caused by data modifications of some other concurrently executing process is known as *process interference* [1, 2]. The problem of process interference is particularly relevant for knowledge-intensive BPs, where shared data are accessed and modified by many processes, involving a large number of stakeholders.

E-Government is a typical area characterized by multiple concurrently executing knowledge-intensive processes. These processes access and modify commonly shared resources such as citizen data, information reported by external contracted parties, etc. In such a context, a "think globally, act locally" approach has to be adopted: each BP instance has to take its own action, independently of other processes, based on how its knowledge about the world evolves during run-time, and how this knowledge affects the next tasks in its workflow. For example, important data used by subsequent tasks may become obsolete, and conditions on which the process relies may not hold anymore. Therefore, a BP has to be continuously informed about changes concerning that data, reason about them, and react accordingly in order to be able to ensure its consistency with the new state of the world.

In the Netherlands, a first attempt has been made to provide a Software as a Service (SaaS) solution for the local e-Government (`www.govunited.nl`). One of the processes that is proposed as a candidate for this initiative concerns the process of the Dutch Law for Societal Support, known as the WMO law. This law is intended to offer support for people with a chronic disease or a disability, by providing facilities (usually by external parties) such as domestic care, transportation, a wheelchair or a home modification. Naturally, several different instances of the WMO process can be executed concurrently, together with other governmental processes, which may access and modify the same data. For example, during the execution of the WMO process, the citizen may move to a different address, the medical status of the citizen may alter, the eligibility criteria may change because of some new directive etc. These changes may pass unnoticed by BPs which rely upon them, and consequently result in unexpected behavior and undesirable business outcomes. The consequences are often noticed only by end customers [3], by erroneous orders or invoices, customer requests that are never handled, etc.

Traditional verification techniques for workflow and data-flow (e.g. [4]) are not sufficient for ensuring the correctness of such BPs, as they assume a closed environment where no other process can use a service that affects the data used by that organization. In addition, most work about resolving process interference refers to failing processes or concerns design-time solutions [5, 6]. Consequently, neither of these solutions is suitable for a highly dynamic SaaS environment. In [2], a run-time mechanism is proposed, where vulnerable parts of the process are monitored in order to manage interferences by employing intervention processes. *Dependency scopes (DS)* are used to specify a *critical section* of the BP, whose correct execution relies on the accuracy of a *volatile* process variable, i.e. a variable that can be changed externally during the execution of the process. If a volatile variable is modified by some exogenous factor during execution of the activities in the respective DS, an *intervention process (IP)* is triggered, with the purpose of resolving the potential execution problems stemming from this change event. However, for complex processes with a large number of activities and many interactions with the environment, the task of manually annotating a BP with DSs becomes difficult, time-consuming, and prone to errors. Thus, critical parts of the BP whose correct execution is dependent on the validity of some volatile variable may be neglected.

In this paper, we extend the initial idea presented in [2], by systematizing the main methodology, and providing an algorithm which automates the task of identifying the critical parts of a BP. To this end, we concretize the proposed approach by describing the semantic extensions to the BP modelling that allow the specification of DSs for resolving runtime process errors. Given a block-style BP specification and some basic information about the services it uses (i.e. the input-output parameters and internal state variables), we show how the parts of the process that are covered by DSs can be automatically inferred. This way, the task of the BP designer can be highly facilitated.

The remainder of this paper is organized as follows. Section 2 describes a possible interference scenario on a real case-study taken from Dutch e-Government, which plays the role of our running example. In Section 3 the basic definitions required for the proposed approach are presented. The algorithm for the automatic identification of critical sections is described in Section 4. Section 5 provides an overview of related work, and the overall conclusions are drawn in Section 6.

## 2 A Process Interference Case-study

In order to illustrate the effects of process interference and the potential ways to overcome them, let us consider a real case-study from the Dutch e-Government regarding the WMO law, as described in [2]. The BP under investigation (referred to as WMO process) concerns the handling of the requests from citizens at one of the 430 municipalities in the Netherlands. In this section, the WMO process is described as used by one of the municipalities. Furthermore, an example is provided, showing the required DSs along with the required IPs.

### 2.1 WMO Process Description

The WMO process (shown in Figure 1) starts with the submission of an application for a provision by a citizen. After receiving the application at the municipality office, a home visit is executed by an officer, in order to gather a detailed understanding of the situation. After the home visit, additional information on the citizen's health may still be required, which can be obtained via a medical advice provided by e.g. a general practitioner. Based on this information, a decision is made by the municipality to determine whether the citizen is eligible to receive the requested provision or not. In case of a negative decision, the citizen has the possibility for appeal. In case of a positive decision, the process continues and the requested provision will be provided. For domestic help, the citizen has the choice between "Personal Budget" and "Care in Kind". In case of a "Personal Budget", the citizen periodically receives a certain amount of money for the granted provision, and in case of "Care In Kind" suppliers who can take care of the provision are contacted. For obtaining a wheelchair, first the detailed requirements are acquired before sending the order to the supplier. The home

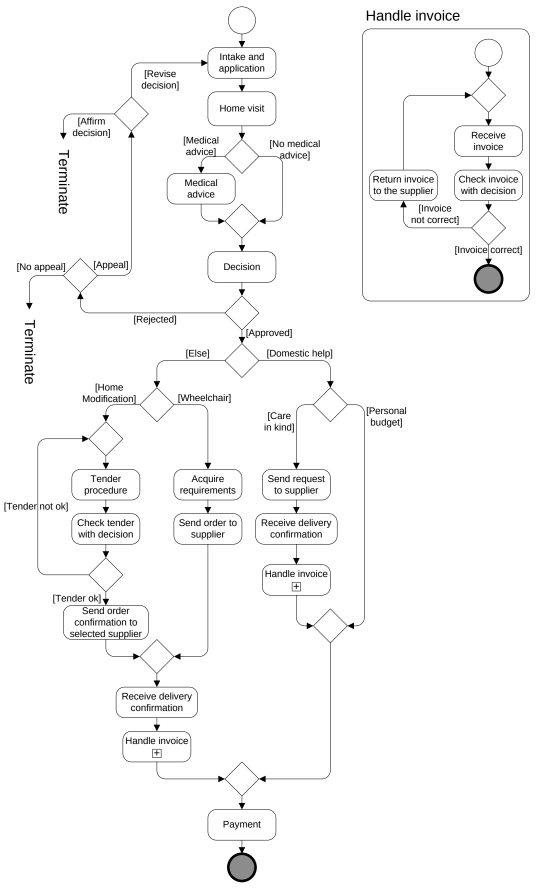

Fig. 1: The WMO process

modification involves a tender procedure to select a supplier that provides the best offer. If the selected tender is approved by the municipality, the order is sent to the selected supplier. After delivery of the provision, an invoice is sent by the supplier to the municipality. Finally, the invoice is checked and paid.

## 2.2 Interference Examples

The request for a wheelchair or a home modification may take up to 6 weeks until the delivery of the provision. These processes depend on the correctness of a number of process variables, like the address of the citizen and the content of the decision. However, these process variables may be changed by another process running in parallel, independently from the WMO process, and are, therefore, volatile. A change in either of these process variables (e.g. address) may have potentially negative consequences for the WMO process, due to its dependencies

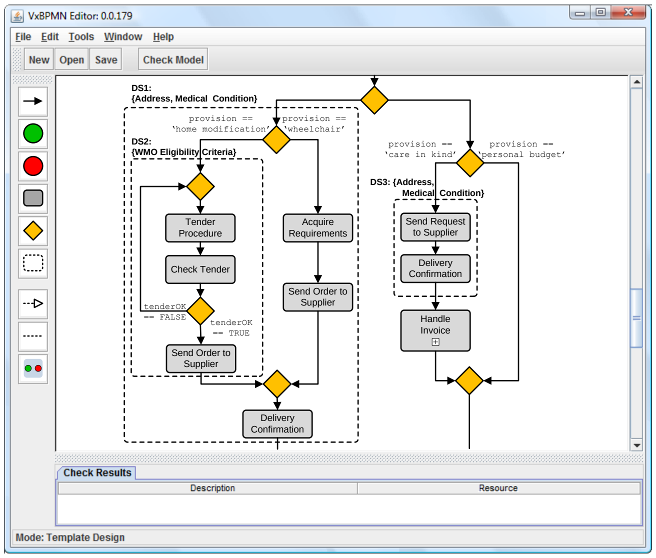

Fig. 2: WMO dependency scopes

on those variables, and lead to erronous outcomes. Such situations are typical examples of process interference.

For example, the requirements of a wheelchair may depend on certain characteristics of the citizen's home. Consequently, an address change after "Acquire requirements" might result in a wheelchair that does not fit the actual requirements. Similarly, if the citizen moves to a nursing home after "Check tender with decision", the home modification is not necessary anymore. However, the supplier is not notified of this address change and the municipality is notified through a different process, which is external to the WMO process. As a result, unless some action is taken to cancel or update the order, the WMO process will proceed with the home modification. In order to guard for changes to the volatile process variables, DSs can be defined, covering those activities for which such a change poses a potential risk of interference. In Figure 2, a part of the process is annotated with DSs using a Process Modeller tool developed for the graphical modeling of BPs. The tool provides a selection of standard control blocks like flow, switch etc., with the extra support of design tools for modeling DSs. For the implementation details see [7].

The activities in DS1 rely on the accuracy of the address. If the address changes, the DS should be triggered, and potentially some recovery activities need to be executed, depending on the state of the BP at that point. For example, if the address change is detected before the order for a wheelchair is sent to the supplier, it is sufficient to execute the IP as shown in Figure 3a. However, if the order is already sent to the supplier, some additional activities are required (Figure 3b). First of all, the current order should be put on hold. After acquiring

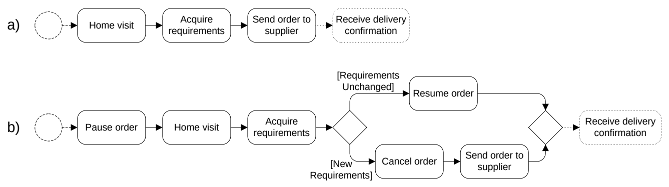

Fig. 3: WMO intervention examples

the requirements again, it is evaluated whether there is a change. If not, the order can be resumed, otherwise the old order should be cancelled and a new order should be sent. The specification of IPs is outside the scope of this paper (for a detailed discussion about the specification of IPs see [2] and [7]).

## 3 Basic Definitions

In this section, we provide the basic definitions regarding the BP representation extended with the support of DSs. First, we define the *Service Repository (SR)*, which is a registry that keeps semantic information about a set of services that are accessible to the client who is executing a specific BP. The SR plays the role of a pool of service descriptions and instances, which are used as the building elements of different process specifications. Service descriptions specify the basic functionalities provided by a service. Service instances refer to specific providers, which offer a service whose functionality conforms to some service description.

The service descriptions specify the operations offered by the respective service type and are represented in terms of simple semantics. Service instances refer to specific providers of a certain service description. The service descriptions can be extracted from standard semantic languages for representing Web Services, such as WSDL-S (`www.w3.org/Submission/WSDL-S`) and OWL-S (`www.w3.org/Submission/OWL-S`). The service descriptions capture the Input-Output behavior of the operations, i.e. the type of the input parameters inputs and of the expected outputs, as well as some information about its internal variables (similar to Locals in OWL-S). No extra semantic information is required to automatically identify the critical sections of a BP.

**Definition 1 (Service Repository (SR)).** *A Service Repository* $SR{=}(SD, SI)$ *is a registry, which keeps a set of* Service Descriptions $SD$*, and a set of* Service Instances $SI$*. A Service Description* $sd \in SD$ *is a tuple* $sd = (sdid, O, SV)$*, where sdid is a unique identifier,* $O$ *is a set of service operations, and* $SV$ *is a list of variables, each ranging over a finite domain. These variables correspond to state variables internal to the service, whose value can be changed by the service operations. Each service operation* $o \in O$ *is a tuple* $o = (id(o), in(o), out(o))$ *where:*

- $id(o)$ *is the identifier of the operation*
- $in(o)$ *is a list of variables that play the role of input parameters to* $o$*, ranging over finite domains*

– *$out(o)$ is a list of variables that play the role of output parameters to $o$, ranging over finite domains*

*A Service Instance $si \in SI$ is a tuple $si = (iid(si), st(si))$:*
– *$st(si)$ is the unique identifier (service type) of the service description $sd \in SD$ this instance complies with*
– *$iid(si)$ is an instance identifier. For each pair of service instances $si_1, si_2 \in SI$ that have the same service type $st(si_1) = st(si_2)$, $iid(si_1) \neq iid(si_2)$.*

The set of state variables involved in the SR may be used by different running process instances, and their value may be changed by any process that has access to the respective setting service operation.

In the followings, the working definition of a Business Process (BP) is provided. Although the WMO process (Figure 1) is represented in BPMN-notation for readability, the core BP representation used in this paper is block-structured [8], and uses the basic BPEL constructs of BPEL, enriched with DSs. As such, the syntax of the BP is block-structured and unambiguously defined, so that the BP can be directly executed by an orchestrator [9], and automatically parsed to identify the parts of the BP that should be covered by a DS. The representation is ultimately a tree structure where a block can have other blocks as children, and for each block its parent can be obtained. All activities included in the BP are references to service instances that exist in the Service Repository.

**Definition 2 (Business Process (BP)).** *Given a Service Repository SR= $(SD, SI)$, a Business Process is a tuple $BP = (PV, E)$, with $E$ being a process element $E = (ACT \mid SEQUENCE \mid FLOW \mid SWITCH \mid REPEAT \mid WHILE \mid DS)$, where:*

– *$PV = PV_i \cup PV_e$ is a set of variables ranging over finite domains.*
  - *$PV_i$ is a set of internal variables, which are declared at the BP level (BP-specific). A subset of $PV_i$ are passed as input parameters to the entire BP, in which case we write $BP(pv_1, \ldots, pv_n)$, where $pv_i \in PV_i$ and $pv_i$ can be initialized with specific values at execution time.*
  - *$PV_e$ is a set of external variables, which refer to state variables declared in the SR. An external variable $v \in PV_e$ is a reference sdid.iid.vid, where sdid is the identifier of a service description $sd = (sdid, O, SV) \in SD$, iid is the identifier of a service instance $si = (iid, sdid) \in SI$, and vid is the identifier of some state variable $v \in SV$.*

– *$ACT$ is a process activity, which represents the invocation of a service operation. For instance, in BPEL it may correspond to an invoke, receive, reply, etc. Every $ACT$ refers to an operation that exists in $SI$. It is a tuple $act = (id(act),\ in(act), out(act))$, where $id(act)$ is a reference sdid.iid.oid, with sdid being an identifier of a service description $sd = (sdid, O, SV) \in SD$, iid the identifier of a service instance $si = (iid, sdid) \in SI$, and oid is the identifier of some operation $o \in O$. The input and output parameters of act refer to the inputs and outputs of the respective oid, i.e. $in(act) = in(oid)$ and $out(act) = out(oid)$. The input (output) parameters of all activities in the BP* *form the sets IP (OP). Input variables can be assigned with constant values or other process variables: $id(act)(ip_1 := v_1, \ldots, ip_n := v_n)$, where $ip_i \in in(act)$, $v_i \in (PV \cup OP)$, or $v_i$ is a value compliant with $ip_i$'s domain. There are also two special types of activities: no-op, which represents an idle activity, and exit, whose execution causes the entire BP to halt.*

- *SEQUENCE refers to a totally ordered set of process elements, which are executed in sequence: $SEQUENCE\{e_1 \ldots e_n\}$, where $e_i$ is a process element.*
- *FLOW represents a set of process elements, which are executed in parallel: $FLOW\{e_1 \ldots e_n\}$, where $e_i$ is a process element.*
- *SWITCH is a set of tuples $\{(c_1, e_1), \ldots, (c_n, e_n)\}$, where $e_i$ is a process element and $c_i$ is a logical condition $C ::= var \circ v$, where $var \in (PV \cup OP)$, $v$ is some constant belonging to var's domain, and $\circ$ is a relational operator ($\circ \in \{=, <, >, \neq, \leq, \geq\}$). All $c_i$ participating in a SWITCH refer to the same variable var and are mutually exclusive.*
- *REPEAT represents a loop structure, and is defined as a tuple $(pe, c\{pe_i\})$, where $c$ is a logical condition as already defined, and $pe, pe_i$ are process elements. $c$ is evaluated just after the end of $pe$, and if it holds then $pe$ is repeated, after the execution of the optional $pe_i$.*
- *DS is a dependency scope as defined in Definition 3.*

## 3.1 Dependency scopes

The DS is based on a **guard-verify** structure to deal with modification events due to factors exogenous to the BP, e.g. due to some other process execution which affects some data on which the BP relies. The critical part of the BP is included in the *guard* block, while the *verify* block specifies the types of events that require intervention. The mechanism of event recording and handling are out of scope of this paper (for a system dealing with process-generated events see e.g. [10]). Whenever such an event occurs, the control flow is transferred to the *verify* block, and the respective goal is activated. Once the resulting IP finishes execution in the updated environment, the control flow of the BP continues from the point following the *guard-verify* structure, unless it is explicitly forced to terminate.

**Definition 3 (Dependency Scope (DS)).** *Given a $SR = (SD, SI)$ and a $BP = (PV_i \cup PV_e, E)$, a dependency scope is a tuple $DS = \langle guard(VV)\{CS\}$, $verify(\{(c_i, IP_i \mid terminate(IP_i))\})\rangle$, where:*

- *$guard(VV)$ indicates the set of volatile variables $VV \subset PV_e$ whose modification triggers the verification of the DS, and CS a process element of BP which is called the* Critical Section. *Whenever during the execution of CS a modification event regarding the value of a $vv \in VV$ is received, the verify part of the DS is triggered, and BP's execution is interrupted.*
- *$verify(\{(c_i, IP_i)\})$ comprises a set of tuples consisting of a logical condition $c_i$ and an intervention process $IP_i$ in compliance with Definition 2 to be pursued if $c_i$ holds. Providing a case condition is optional, with the default interpretation*

*being $c_i = TRUE$. $IP_i$ specifies a BP which ensures the satisfaction of the properties that reflect the state right after the final activity of CS. After the interruption of the BP, some $IP_i$ is executed, and then BP is resumed just after CS (and from any other parallel branches that were interrupted).*

– *terminate(IP) forces the rest of BP's execution to be aborted after completing IP's execution.*

Following Definition 3, the DS specification representing *DS1* of Figure 2 is as follows, where *IPa*, *IPb* and *IPc* refer to the respective intervention processes, which take care of repairing the erroneous execution in each of the cases.

```
<ds>
  <guard>
    <variables>
      <variable name="address" dataType="dt:address"/>
      <variable name="medCond" dataType="dt:medInfo"/>
    </variables>
    <criticalSection>
      <!-- Subprocess covered by DS1 as in Figure 2 -->
    </criticalSection>
  </guard>
  <verify>
    <case condition="address.county!=‘Groningen’">
      <terminate>
        <invoke name="IPa"/>
      </terminate>
    </case>
    <case condition="address.county=‘Groningen’&AND;medCond!=‘deceased’">
      <invoke name="IPb"/>
    </case>
    <case condition="medCond=‘deceased’">
      <terminate>
        <invoke name="IPc"/>
      </terminate>
    </case>
  </verify>
</ds>
```

According to *DS1*, if a modification event regarding the address or the medical condition is received within the scope of the guarded subprocess, different IPs are executed, depending on the state of execution and the kind of modification that has occurred. For example, if the address change indicates that the citizen has moved to another municipality, then *IPa* includes canceling the order (either for a wheelchair or home modification) if one has already been issued, and sending a notification to the city hall. Similarly, IPb takes care of the situation where the customer has moved within the range of the municipality, and IPc in case his medical condition has changed to ‘deceased’. In the following section we describe how the $guard(VV)\{CS\}$ part of a DS description can be derived automatically, by parsing the BP specification.

## 4 Automatic Identification of Critical Sections

The algorithm of automated generation of the parts of a BP covered by a DS is presented in Algorithm 1 below. The algorithm guarantees that the computed

CSs are elements of the BP in compliance with Definition 2. CSs cover all activities that are directly or indirectly dependent on the same set of volatile variables $VV$. That is, they either use a $vv \in VV$ as input or use the output of another activity, which is dependent on $vv$. These activities are referred to as *Dependent Activities* ($DA$). In order to ensure that important change events will not pass untreated, any part of the process in a potential execution path between two activities dependent on the same $VV$ should also be covered by the respective CS. This is necessary to take care of any modification of $vv$ that occurs during the execution of this intermediate part, since the modification may require the cancelation or repetition of some preceding part of the BP which relied on some $vv \in VV$ (e.g. performing a new visit to the new house if the address has changed), and which is used by a succeeding element (e.g. to calculate the characteristics of the requested wheelchair). However, branches in switch or flow constructs that are not on a potential path between two activities dependent on some $vv$, should not be unnecessarily included in the respective CS, in order to avoid unnecessary invocation of intervention processes.

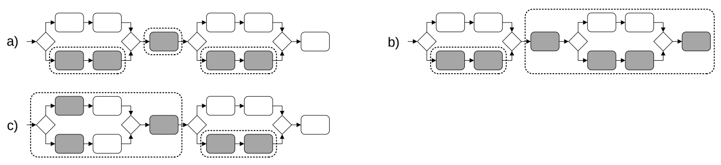

Fig. 4: CS creation examples

In Figure 4, some examples of CSs are provided to illustrate the properties described above. The shaded activities are dependent on $VV$ and should be covered by a CS. The CSs are indicated by a dashed line. In case (a), only the specific branches of the switch-constructs that comprise dependent activities are included in the CS. In situation (b), however, the second switch has to be covered entirely by a CS, because the last activity is dependent on $VV$ as well. Any modification event regarding a $vv \in VV$ that occurs during the upper branch (which is not dependent on $VV$) has still to be dealt with, since the last activity may use a a variable that is a result of some dependent activities before the switch, which produced this result based on the obsolete $vv$. In situation (c), both branches of the first switch contain activities that are not dependent on $VV$. However, as they both are on a path between activities that are dependent on $VV$, the entire switch is covered by a CS.

The main function of Algorithm 1 is *extractScopes*, which takes as an input a BP specification in accordance with Definition 2 and the list of volatile variables $VV$. *extractScopes* returns a list of tuples $\langle VV_i, CS_i\rangle$, which correspond to the *guard* parts of all DSs in the BP. Given a $BP = (PV_i \cup PV_e, E)$, $VV = PV_e$. That is, all state variables that are declared in the SR and used in the BP should be guarded, since their modification may be a source of erroneous results. The

BP is treated as a tree (represented in XML), where the root is the outermost element in the specification, and the leaves are the activities.

The outermost loop in the function *extractScopes* iterates over the list of volatile variables $VV$. For each $vv \in VV$, critical sections are extracted separately. Identical CSs for different variables are merged into a united CS at the end by *mergeScopes*. The first step (line 4) is to find all activities and switch–blocks that depend directly or indirectly on the volatile variable $vv$, by calling the function *getDependentElems*. First (line 18), all activities for which $vv$ is assigned to some of their input parameters directly or by transitivity are added to the dependent elements $DE$. Then (line 24), $DE$ is augmented by adding all switch–blocks whose condition is either on $vv$, or some variable produced by the already considered activities. All elements in $DE$ are arranged in a breadth-first order as they appear in the BP. The next step in *extractScopes* is to iterate through the list $DE$. In the inner loop, for each pair of elements $e_i, e_j$, it is checked whether their minimal common ancestor is of type sequence. If so, then the function $getTempCS$ is called, which returns a set of elements that are candidates for being CSs with respect to the variable $vv$, and lie between $e_i$ and $e_j$. Then, $e_j$ can be removed from $DE$, since subsequent inspections on it are redundant, as the appropriate CSs covering it have already been computed.

Function *getTempCS(*$e_i, e_j, BP$*)* first calls *getPathBtw* to compute the path between $e_i$ and $e_j$ (line 31), which comprises all elements that are part of the sequence between $e_i$ and $e_j$, including the special markers *StartBranchEl* and *EndBranchEl*. These markers indicate the start (splits) and end points (joins) of branching elements. Consequently, a *path* is a list with members of type *Item* (line 44), where an item is either a process element or a $BranchElMarker$. Markers are added in the path only if they concern joins (splits) for which the respective split (join) is not encountered during the traversal of the BP from $e_i$ to $e_j$. This way, the markers divide the path into the appropriate sequences of elements (lines 33 to 39), each of which is a candidate for being a CS.

Function *getPathBtw* uses the auxiliary function *nextItems* (not explained in the algorithm for space reasons), which returns a list consisting of the next element in the sequence path, and some possible *EndBranchEl*, if any are encountered before the next element is fetched. These are added to the path, and the process proceeds by fetching the next items (line 45), until the element in the sequence that contains $e_j$ is reached. In the latter case, *pathInElem* is called, which traverses the path within this last element until $e_j$ is reached. If the element containing $e_j$ is an activity or sequence, this activity ($e_j$) or the subsequence till $e_j$ (line 52) are returned respectively. If the element is a switch or flow, then a *StartBranchEl* marker is added in the list of results, and the branch containing $e_j$ is inspected. *pathInElem* is called recursively on this branch, and all items in the path leading to $e_j$ are collected in $path_j$. Consequently, the computation of the entire path is completed, and returned to $getTempCS$. The path is traversed (line 33), and divided into the appropriate CSs: *currCS* is constructed as a sequence of the elements in *path*, until a marker is met, at which point *currCS* is added to the list of candidate CSs.

**Algorithm 1** Automatic computation of the set of the pairs Guarded=$\{\langle VV_i, CS_i\rangle\}$, consisting of volatile variables and respective elements that constitute the Critical Sections

```
1:  function EXTRACTSCOPES($BP$, $VV$): List[(List[V], E)]
2:    for each $vv \in VV$ do
3:      $guardList = \emptyset$
4:      $DE$ = GETDEPENDENTELEMS($vv, BP$)
5:      for each $e_i \in DE$ do
6:        $tmpCS = \emptyset$
7:        $DE = DE$.remove($e_i$)
8:        for each $e_j \in DE$ do
9:          if type(minCommonAncestor($e_i, e_j$))=sequence then
10:           $tmpCS = tmpCS \cup$ GETTEMPCS($e_i$, $e_j$, $BP$)
11:           $DE = DE$.remove($e_j$)
12:       for $tmpCS_i \in tmpCS$ do
13:         $guardList$.add($\langle \{vv\}, tmpCS_i \rangle$)
14:   MERGESCOPES ($guardList$)

15: function GETDEPENDENTELEMS($vv, BP$): List[Element]
16:   $varList = \{vv\}$
17:   $DE = \emptyset$
18:   for each $a_i \in BP$.getActivities do
19:     for each $ip_i := v \in a_i$.parseInputAssignments do
20:       if $v \in varList$ then
21:         for each $op_i \in out(a_i)$ do
22:           $varList.add(op_i)$
23:         $DE$.add($a_i$); break;
24:   for each $SWITCH_i \in BP$.getSWITCHelements do
25:     $c_i = SWITCH_i$.getFirstCondition
26:     if $c_i.getLeftVariable \in varList$ then
27:       miDE.add($SWITCH_i$);
28:   return $DE$

29: function GETTEMPCS($e_i, e_j, BP$): List[Elem]
30:   $tmpCSList = \emptyset$
31:   $path$ = GETPATHBTW($e_i, e_j, BP$)
32:   $currCS = \varnothing$
33:   for each $item \in path$ do
34:     match type($item$)
35:       case Element:
36:         $currCS$.attachInSeq($item$)
37:       case BranchElMarker:
38:         $tmpCSList$.add($currCS$)
39:         $currCS = \emptyset$
40:   return $tmpCSList$
```

```
41: **function** getPathBtw($e_i, e_j, BP$): List[Item]
42:   $currElem = e_i$
43:   **while** $\neg$ $currElem$.contains($e_j$) **do**
44:     $path$.append($currItems$)
45:     $currItems$ = nextItem($currElem, e_i, BP$)
46:     $currElem = currItems$.getElement
47:     **if** $currItems = \emptyset$ **then return** $\emptyset$
48:   $path$.append(pathInElem($currElem, e_j, BP$))
49:   **return** $path$

50: **function** pathInElem($el, endEl, BP$): List[Item]
51:   **match** type($el$)
52:     **case** activity:
53:       **return** $\{el\}$
54:     **case** sequence:
55:       **return** $el.subsequenceTill(endEl)$
56:     **case** SWITCH $\vee$ flow:
57:       $path_j = \{StartBrEl\}$
58:       $branch_j = el.getBranchWith(endEl)$
59:       **return** $path_j$.append(pathInElem($branch_j, endEl, BP$)
60:   **return** $\emptyset$
```

Once the list of temporary CSs *tmpCS* regarding a volatile variable *vv* is computed as described above, *extractScopes* proceeds with constructing the respective *guardList* consisting of tuples $\langle\{vv\}, tmpCS_i\rangle$ (line 12). After repeating the process described above for each $vv \in VV$, *mergeScopes* is called, in order to clean up the candidate CSs. The following steps are performed in that order:

- If there are two tuples $\langle\{v_1\}, CS_1\rangle$ and $\langle\{v_2\}, CS_2\rangle$, where $CS_1$ and $CS_2$ are identical, then they are replaced by a single tuple $\langle\{v_1, v_2\}, CS_1\rangle$.
- If there are two tuples $\langle\{v_1\}, CS_1\rangle$ and $\langle\{v_2\}, CS_2\rangle$, where $v_1 = v_2$ and $CS_1.descendantOf(CS_2)$, then the former tuple is removed as redundant.
- If a list of tuples on the same volatile variable set $\langle VV, CS_1\rangle, \ldots, \langle VV, CS_n\rangle$ correspond to the branches of a switch, i.e. there is an $e_{switch} = switch\{(CS_1, e_1), \ldots, (CS_n, e_n)\}$, then these are replaced with a single CS, which covers the entire switch–element. A similar process is performed for flow branches.
- If a list of tuples on the same volatile variable set $\langle VV, CS_1\rangle, \ldots, \langle VV, CS_n\rangle$ are interrelated through a sequence relation, i.e. there is a $seq\{CS_1, \ldots, CS_n\}$, then these are replaced with a single CS, which covers the entire sequence.

Algorithm 1 has been applied to the BP specification of the WMO process represented in Figure 1. The algorithm identified three volatile variables, and all five critical sections related to them. The total time for parsing the WMO process specification and computing all CSs is below 100 msec. The discovered CSs can then be projected on the Process Modeller, as presented in Figure 2.

## 5 Related work

Process interference between concurrent BPs occurs frequently in organizations, and some solutions have been provided in literature, e.g. [2, 5, 6]. Although the use of temporal logic for data-flow analysis in business processes can ensure soundness of both the control-flow and the data-flow [4], runtime disruptions due to external data changes are not accounted for. As a result, process interference can not be prevented or resolved by such methods.

However, most existing mechanisms to resolve process interference are either providing a design-time solution, thus requiring that the designer anticipates all potential problems and ways to overcome them in advance, or are based on failing processes [5]. A more elaborate solution for process interference in Service-Oriented Computing is provided by [6], where in addition to failing processes, events like exceptional conditions or unavailable activities are covered. More specifically to cloud computing, an approach for handling faults due to failing processes or services is presented by [11]. In practice, however, process interference does not necessarily cause processes to fail. Often, processes may end up with providing erroneous outcomes as a result of wrong data values, a problem that is acknowledged in [2].

Interference causes processes to provide erroneous outcomes as a result of wrong data values. In most cases, however, wrong data values are interpreted a data integrity problem. Much work has been done with respect to ensuring data integrity in distributed and concurrent systems. Some techniques for checking the integrity of distributed and dynamic data stored on the cloud are discussed in [12, 13], while [14] focus on run-time failures that affect cloud short-lived data. Although the interference problem is related to concurrent data usage, the cause of the problem is beyond data integrity issues. Therefore, we focus on problems that arise at the level of process execution due to the use of outdated data.

## 6 Concluding Remarks

One of the main challenges posed by the emergent distributed setting of modern BP Management Systems comes from the interference between different processes that access common resources. During execution of a business process, a data modification caused by some external factor may lead to erronous results, and should, therefore, be guarded and dealt with. To address this issue, the correct identification of the sections of a business process, whose correct execution depends on some volatile variable, is very important. These sections shoul be guarded upon, so that whenever a modification event is received during their execution, an appropriate intervention process is executed, in order to restore the process to a consistent state. However, the task of manual specification of these critical sections can become cumbersome and prone to errors, especially for processes with a complex structure, using many shared resources. To facilitate this task, we have developed an algorithm, which automatically computes the appropriate critical sections, given a BP specification and some semantics regarding the input-output and the internal state variables of the service operations used by the process. We have shown how this can be applied in a real case-study taken from the Dutch e-government. The results can be presented on a process modelling tool in a graphical way, so as to assist the process designer in the specification of the necessary dependency scopes in order to ensure the delivery of correct results by the process.

## References

1. Xiao, Y., Urban, S.: Process dependencies and process interference rules for analyzing the impact of failure in a service composition environment. In: Business Inf. Systems. Volume 4439 of LNCS. (2007) 67–81
2. van Beest, N.R.T.P., Bulanov, P., Wortmann, J., Lazovik, A.: Resolving business process interference via dynamic reconfiguration. In: Proc. of 8th Int. Conf. on Service Oriented Computing (ICSOC). (2010) 47–60
3. van Beest, N.R.T.P., Szirbik, N.B., Wortmann, J.C.: Assessing the interference in concurrent business processes. In: Proc. of 12th Int. Conf. on Enterprise Information Systems (ICEIS). (2010) 261–270
4. Trčka, N., van der Aalst, W., Sidorova, N.: Data-flow anti-patterns: Discovering data-flow errors in workflows. In: Adv. Inf. Systems Eng. Volume 5565 of LNCS. (2009) 425–439
5. Xiao, Y., Urban, S.: Using data dependencies to support the recovery of concurrent processes in a service composition environment. In: Proc. of the 16th Int. Conf. on Cooperative Inf. Systems. (2008) 139–156
6. Urban, S., Gao, L., Shrestha, R., Courter, A.: The dynamics of process modeling: New directions for the use of events and rules in service-oriented computing. In: The Evolution of Conceptual Modeling. Volume 6520 of LNCS. (2011) 205–224
7. van Beest, N.R.T.P., Kaldeli, E., Bulanov, P., Wortmann, J., Lazovik, A.: Automated runtime repair of business processes. Technical Report 2012-12-2, University of Groningen (2012) www.cs.rug.nl/∼eirini/papers/tech_2012-12-2.pdf.
8. Ouvans, C., Dumas, M., ter Hofstede, A., van der Aalst, W.: From BPMN process models to BPEL web services. In: Int. Conf. on Web Services. (2006) 285–292
9. Kopp, O., Martin, D., Wutke, D., Leymann, F.: On the choice between graph-based and block-structured business process modeling languages. In: Modellierung betrieblicher Informationssysteme (MobIS 2008). Volume 141 of Lecture Notes in Informatics (LNI)., Gesellschaft für Informatik e.V. (GI) (2008) 59–72
10. Rozsnyai, S., Vecera, R., Schiefer, J., Schatten, A.: Event cloud - searching for correlated business events. In: 9th IEEE Int. Conf. on E-Commerce Technology / 4th IEEE Int. Conf. on Enterprise Computing, E-Commerce and E-Services. (2007)
11. Juhnke, E., Dornemann, T., Freisleben, B.: Fault-tolerant BPEL workflow execution via cloud-aware recovery policies. In: 35th EUROMICRO Conference on Softw. Eng. and Adv. Applications (SEAA). (2009) 31 – 38
12. Sravan Kumar, R., Saxena, A.: Data integrity proofs in cloud storage. In: 3rd Int. Conf. on Communication Systems and Networks (COMSNETS). (2011) 1 – 4
13. Hao, Z., Zhong, S., Yu, N.: A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. IEEE Trans. on Knowledge and Data Engineering **23**(9) (2011) 1432–1437
14. Ko, S.Y., Hoque, I., Cho, B., Gupta, I.: Making cloud intermediate data fault-tolerant. In: 1st ACM Symposium on Cloud computing. (2010) 181–192

# Semantically-Governed Data-Aware Processes

Diego Calvanese[1], Giuseppe De Giacomo[2], Domenico Lembo[2], Marco Montali[1], and Ario Santoso[1]

[1] Free University of Bozen-Bolzano, `lastname@inf.unibz.it`
[2] Sapienza Università di Roma, `lastname@dis.uniroma1.it`

**Abstract.** In this paper we consider processes that run over data stored in a relational database. Our setting is that of ontology-based data access (OBDA), where the information in the database is conceptually represented as an ontology and is declaratively mapped to it through queries. We are interested in verifying temporal logic formulas on the evolution of the information at the conceptual level, taking into account the knowledge present in the ontology, which allows for deducing information that is only implicitly available. Specifically, we show how, building on first-order rewritability of queries over the system state that is typical of ontology languages for OBDA, we are able to reformulate the temporal properties into temporal properties expressed over the underlying database. This allows us adopt notable decidability results on verification of evolving databases that have been established recently.

## 1 Introduction

Recent work in business processes, services and databases brought the necessity of considering both data and processes simultaneously while designing the system. This holistic view of considering data and processes together has given rise to a line of research under the name of *artifact-centric business processes* [16, 14, 19, 1] that aims at avoiding the notorious discrepancy of traditional approaches where these aspects are considered separately [7]. Recently, interesting decidability results for verification of temporal properties over such systems have been obtained in the context of so-called Data-centric Dynamic Systems (DCDSs) based on relational technology [12, 6, 4, 5]. In a DCDS, processes operate over the data of the system and evolve it by executing actions that may issue calls to external services. The data returned by such external services is injected into the system, effectively making it infinite state. There has been also some work on a form of DCDS based on ontologies, where the data layer is represented in a rich ontology formalism, and actions perform a form of instance level update of the ontology [3]. The use of an ontology allows for a high-level conceptual view of the data layer that is better suited for a business level treatment of the manipulated information.

Here we introduce Semantically-Governed Data-Aware Processes (SGDAP), in which we merge these two approaches by enhancing a *relational layer* constituted by a DCDS based system, with an ontology, constituting a *semantic layer*. The ontology captures the domain in which the SGDAP is executed, and allows for seeing the data and their manipulation at a conceptual level through an ontology-based data access (OBDA) system [8, 18]. Hence it provides us with a way of semantically governing the underlying DCDS. Specifically, an SGDAP is constituted by two main components: *(i)* an *OBDA system* [8] which includes (the intensional level of) an ontology, a relational database schema, and a mapping between the ontology and the database; *(ii)* a *process component*, which characterizes the evolution of the system in terms of a process specifying preconditions and effects of action execution over the relational layer.

The ontology is represented through a Description Logic (DL) TBox [2], expressed in a lightweight ontology language of the *DL-Lite* family [10], a family of DLs specifically designed for efficiently accessing to large amounts of data. The mapping is defined in terms of a set of assertions, each relating an arbitrary (SQL) query over the relational layer to a set of atoms whose predicates are the concepts and roles of the ontology, and whose arguments are terms built using specific function symbols applied to the answer variables of the SQL query. Such mappings specify how to populate the elements of the ontology from the data in the database, and function symbols are used to construct (abstract) objects (*object terms*) from the concrete values retrieved from the database.

When an SGDAP evolves, each snapshot of the system is characterized by a database instance at the relational layer, and by a corresponding virtual ABox, which together with the TBox provides a conceptual view of the relational instance at the semantic layer. When the system is progressed by the process component, we assume that at every time the current instance can be arbitrarily queried, and can be updated through action executions, possibly involving external service calls to get new values from the environment. Hence the process component relies on three main notions: *actions*, which are the atomic progression steps for the data layer; *external services*, which can be called during the execution of actions; and a *process*, which is essentially a non-deterministic program that uses actions as atomic instructions. During the execution, the snapshots of the relational layer can be virtually mapped as ABoxes in the semantic layer. This enables to: *(i) understand* the evolution of the system at the conceptual level, and *(ii) govern* it at the semantic level, rejecting those actions that, executed at the relational layer, would lead to a new semantic snapshot that is inconsistent with the semantic layer's TBox.

In this work, we are interested in verifying dynamic properties specified in a variant of $\mu$-calculus [15], one of the most powerful temporal logics, expressed over the semantic layer of an SGDAP. We consider properties expressed as $\mu$-calculus formulae whose atoms are queries built over the semantic layer. By relying on techniques for query answering in *DL-Lite* OBDA systems, which exploit FOL rewritability of query answering and of ontology satisfiability, we reformulate the temporal properties expressed over the semantic layer into analogous properties over the relational layer. Given that our systems are in general infinite-state, verification of temporal properties is undecidable. However, we show how we can adapt to our setting recent results on the decidability of verification of DCDSs based on suitable finite-state abstractions [5].

## 2 Preliminaries

In this section we introduce the description logic (DL) *DL-Lite*$_{A,id}$ and describe the ontology-based data access (OBDA) framework.

***DL-Lite*$_{A,id}$** [11, 8] allows for specifying *concepts*, representing sets of objects, *roles*, representing binary relations between objects, and *attributes*, representing binary relations between objects and values. The syntax of concept, role and attribute *expressions* in *DL-Lite*$_{\mathcal{A},id}$ is as follows:

$$B \longrightarrow N \mid \exists R \mid \delta(U) \qquad\qquad R \longrightarrow P \mid P^-$$

Here, $N$, $P$, and $U$ respectively denote a *concept name*, a *role name*, and an *attribute name*, $P^-$ denotes the *inverse of a role*, and $B$ and $R$ respectively denote *basic concepts* and *basic roles*. The concept $\exists R$, also called *unqualified existential restriction*, denotes the *domain* of a role $R$, i.e., the set of objects that $R$ relates to some object. Similarly, the concept $\delta(U)$ denotes the *domain* of an attribute $U$, i.e., the set of objects that $U$ relates to some value. Note that we consider here a simplified version of *DL-Lite*$_{\mathcal{A},id}$ where we distinguish between between objects and values, but do not further deal with different datatypes; similarly, we consider only a simplified version of identification assertions.

A *DL-Lite*$_{\mathcal{A},id}$ *ontology* is a pair $(\mathcal{T}, A)$, where $\mathcal{T}$ is a TBox, i.e., a finite set of *TBox assertions*, and $A$ is an Abox, i.e., a finite set of *ABox assertions*. *DL-Lite*$_{\mathcal{A},id}$ TBox assertions have the following form:

$$\begin{array}{ccc} B_1 \sqsubseteq B_2 & R_1 \sqsubseteq R_2 & U_1 \sqsubseteq U_2 \\ B_1 \sqsubseteq \neg B_2 & R_1 \sqsubseteq \neg R_2 & U_1 \sqsubseteq \neg U_2 \\ (\mathsf{id}\ B\ Z_1, \ldots, Z_n) & (\mathsf{funct}\ R) & (\mathsf{funct}\ U) \end{array}$$

From left to right, assertions of the first row denote *inclusions* between basic concepts, basic roles, and attributes; assertions of the second row denote *disjointness* between basic concepts, basic roles, and attributes; assertions of the last row denote *identification (assertions)* (IdA), and global *functionality* on roles and attributes. In the IdA, each $Z_i$ denotes either an attribute or a basic role. Intuitively, an IdA of the above form asserts that for any two different instances $o$, $o'$ of $B$, there is at least one $Z_i$ such that $o$ and $o'$ differ in the set of their $Z_i$-fillers, that is the set of objects (if $Z_i$ is a role) or values (if $Z_i$ is an attribute) that are related to $o$ by $Z_i$. As usual, in *DL-Lite*$_{\mathcal{A},id}$ TBoxes we impose that roles and attributes occurring in functionality assertions or IdAs cannot be specialized (i.e., they cannot occur in the right-hand side of inclusions).

*DL-Lite*$_{\mathcal{A},id}$ ABox assertions have the form $N(t_1)$, $P(t_1, t_2)$, or $U(t_1, v_1)$, where $t_1$ and $t_2$ denote individual objects and $v_1$ denotes a value.

The semantics of *DL-Lite*$_{\mathcal{A},id}$ is given in [11]. We only recall here that we interpret objects and values over distinct domains, and that for both we adopt the Unique Name Assumption, i.e., different constants denote different objects (or values). The notions of *entailment*, *satisfaction*, and *model* are as usual [11]. We also say that $A$ is *consistent wrt* $\mathcal{T}$ if $(\mathcal{T}, A)$ is satisfiable, i.e., admits at least one model.

Next we introduce queries. As usual (cf. OWL 2), answers to queries are formed by terms denoting individuals appearing in the ABox. The *domain of an ABox* $A$, denoted by $\textsc{adom}(A)$, is the (finite) set of terms appearing in $A$. A *union of conjunctive queries* (UCQ) $q$ over a TBox $\mathcal{T}$ is a FOL formula of the form $\exists \vec{y_1}.conj_1(\vec{x}, \vec{y_1}) \vee \cdots \vee \exists \vec{y_n}.conj_n(\vec{x}, \vec{y_n})$, with free variables $\vec{x}$ and existentially quantified variables $\vec{y_1}, \ldots, \vec{y_n}$. Each $conj_i(\vec{x}, \vec{y_i})$ in $q$ is a conjunction of atoms of the form $N(z)$, $P(z, z')$, $U(z, z')$ where $N$, $P$ and $U$ respectively denote a concept, role and attribute name of $\mathcal{T}$, and $z$, $z'$ are constants in a set $\mathcal{C}$ or variables in $\vec{x}$ or $\vec{y_i}$, for some $i \in \{1, \ldots, n\}$. The *(certain) answers* to $q$ over an ontology $(\mathcal{T}, A)$ is the set $ans(q, \mathcal{T}, A)$ of substitutions[3]

[3] As customary, we can view each substitution simply as a tuple of constants, assuming some ordering of the free variables of $q$.

$\sigma$ of the free variables of $q$ with constants in $\textsc{adom}(A)$ such that $q\sigma$ evaluates to true in every model of $(\mathcal{T}, A)$. If $q$ has no free variables, then it is called *boolean*, and its certain answers are true or false. Computing $ans(q, \mathcal{T}, A)$ of a UCQ $q$ over a *DL-Lite*$_{\mathcal{A},id}$ ontology $(\mathcal{T}, A)$ is in $AC^0$ in the size of $A$ [11]. This is actually a consequence of the fact that *DL-Lite*$_{\mathcal{A},id}$ enjoys the *FOL rewritability* property, which in our setting says that for every UCQ $q$, $ans(q, \mathcal{T}, A)$ can be computed by evaluating the UCQ $\textsc{rew}(q, \mathcal{T})$ over $A$ considered as a database. $\textsc{rew}(q, \mathcal{T})$ is the so-called perfect reformulation of $q$ w.r.t. $\mathcal{T}$ [11]. We also recall that, in *DL-Lite*$_{\mathcal{A},id}$, ontology satisfiability is FOL rewritable. In other words, we can construct a boolean FOL query $\mathsf{q}_{\mathsf{unsat}}(\mathcal{T})$ that evaluates to true over an ABox $A$ iff the ontology $(\mathcal{T}, A)$ is unsatisfiable.

In our framework, we consider an extension of UCQs, called ECQs, which are queries of the query language *EQL-Lite*(UCQ) [9]. Formally, an *ECQ* over a TBox $\mathcal{T}$ is a possibly open *domain independent* formula of the form:

$$Q \longrightarrow [q] \mid \neg Q \mid Q_1 \wedge Q_2 \mid \exists x.Q \mid x = y$$

where $q$ is a UCQ over $\mathcal{T}$ and $[q]$ denotes that $q$ is evaluated under the (minimal) knowledge operator (cf. [9]). To compute the certain answers $\textsc{ans}(Q, \mathcal{T}, A)$ to an ECQ $Q$ over an ontology $(\mathcal{T}, A)$, we can compute the certain answers over $(\mathcal{T}, A)$ of each UCQ embedded in $Q$, and evaluate the first-order part of $Q$ over the relations obtained as the certain answers of the embedded UCQs. Hence, also computing $\textsc{ans}(Q, \mathcal{T}, A)$ of an ECQ $Q$ over a *DL-Lite*$_{\mathcal{A},id}$ ontology $(\mathcal{T}, A)$ is in $AC^0$ in the size of $A$ [9].

**Ontology-Based Data Access (OBDA).** In an OBDA system, a relational database is connected to an ontology that represents the domain of interest by a mapping, which relates database values with values and (abstract) objects in the ontology (c.f. [8]). In particular, we make use of a countably infinite set $\mathcal{V}$ of values and a set $\Lambda$ of function symbols, each with an associated arity. We also define the set $\mathcal{C}$ of constants as the union of $\mathcal{V}$ and the set $\{f(d_1, \ldots, d_n) \mid f \in \Lambda \text{ and } d_1, \ldots, d_n \in \mathcal{V}\}$ of *object terms*.

Formally, an OBDA system is a structure $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$, where: *(i)* $\mathcal{R} = \{R_1, \ldots, R_n\}$ is a database schema, constituted by a finite set of relation schemas; *(ii)* $\mathcal{T}$ is a *DL-Lite*$_{\mathcal{A},id}$ TBox; *(iii)* $\mathcal{M}$ is a set of mapping assertions, each of the form: $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{y}, \vec{t})$, where: (a) $\vec{x}$ is a non-empty set of variables, (b) $\vec{y} \subseteq \vec{x}$, (c) $\vec{t}$ is a set of object terms of the form $f(\vec{z})$, with $f \in \Lambda$ and $\vec{z} \subseteq \vec{x}$, (d) $\Phi(\vec{x})$ is an arbitrary SQL query over $\mathcal{D}$, with $\vec{x}$ as output variables, and (e) $\Psi(\vec{y}, \vec{t})$ is a conjunctive query over $\mathcal{T}$ of arity $n > 0$ without non-distinguished variables, whose atoms are over the variables $\vec{y}$ and the object terms $\vec{t}$.

*Example 1.* As a running example, we consider a simple university information system that stores and manipulates data concerning students and their degree. In particular, we define an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ to capture the conceptual schema of such a domain, how data are concretely maintained in a relational database, and how the two information levels are linked through mappings. The conceptual schema is depicted in Figure 1, and formalized as the following *DL-Lite*$_{\mathcal{A},id}$ TBox $\mathcal{T}$:

$$\begin{array}{rll}
\mathsf{Bachelor} \sqsubseteq \mathsf{Student} & \delta(\mathsf{MNum}) \sqsubseteq \mathsf{Student} & (\mathsf{funct}\ \mathsf{MNum}) \\
\mathsf{Master} \sqsubseteq \mathsf{Student} & \mathsf{Student} \sqsubseteq \delta(\mathsf{MNum}) & (\mathsf{id}\ \mathsf{Student}\ \mathsf{MNum}) \\
\mathsf{Graduated} \sqsubseteq \mathsf{Student} & &
\end{array}$$

The conceptual schema states that Bachelor and Master are subclasses of Student, that some Students could be already Graduated, and that MNum (representing the matriculation number) is

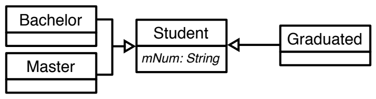

**Fig. 1.** UML conceptual schema for our running example.

an attribute relating individuals of type Student (domain of the attribute) to corresponding Codes (range of the attribute). The conceptual schema also expresses that each Student has *exactly one* matriculation number, and we assume that matriculation numbers can be used to *identify* Students (i.e., each MNum is associated to *at most one* Student). Data related to students are maintained in a concrete underlying data source that obeys the database schema $\mathcal{R}$, constituted by the following relation schemas: *(i)* ENROLLED(id, name, surname, type, endDate) stores information about students that are currently (endDate=NULL) or were enrolled in a bachelor (type="Bachelor") or master (type="Master") course. *(ii)* GRAD(id, mark, type) stores data of former students who have been graduated. *(iii)* TRANSF_M(name, surname) is a temporary relation used to maintain information about master students that have been recently transferred from another university, and must still complete the enrollment process. The interconnection between the database schema $\mathcal{R}$ and the conceptual schema $\mathcal{T}$ is specified through the following set $\mathcal{M}$ of mappings:

$m_1$ : SELECT name, surname, type FROM ENROLLED WHERE type ="Bachelor"
  $\rightsquigarrow$ Bachelor(stu$_1$(name, surname, type))
$m_2$ : SELECT name, surname, type FROM ENROLLED WHERE type ="Master"
  $\rightsquigarrow$ Master(stu$_1$(name, surname, type))
$m_3$ : SELECT name, surname, type, id FROM ENROLLED $\rightsquigarrow$ MNum(stu$_1$(name, surname, type), *val*(id))
$m_4$ : SELECT name, surname FROM TRANSF_M $\rightsquigarrow$ Master(stu$_1$(name, surname, "Master"))
$m_5$ : SELECT e.name, e.surname, e.type FROM ENROLLED e, GRAD g WHERE e.id = g.id
  $\rightsquigarrow$ Graduated(stu$_1$(name, surname, type))

Intuitively, $m_1$ ($m_2$ resp.) maps every id in ENROLLED with type "Bachelor" ("Master") to a bachelor (master) student. Such a student is constructed by "objectifying" the name, surname and course type using variable term stu$_1/3$. In $m_3$, the MNum attribute is instead created using directly the value of id to fill in the target of the attribute. Notice the use of the *val* function symbol for mapping id to the range of MNum. Mapping $m_4$ leads to create further master students by starting from the temporary TRANSF_M table. Since such students are not explicitly associated to course type, but it is intended that they are "Master", objectification is applied to students' name and surname, adding "Master" as a constant in the variable term. Notice that, according to the TBox $\mathcal{T}$, such students have a matriculation number, but its value is not known (and, in fact, no mapping exists to generate their MNum attribute). Finally, $m_5$ generates graduated students by selecting only those students in the ENROLLED table whose matriculation number is also contained in the GRAD table. □

Given a database instance $D$ made up of values in $\mathcal{V}$ and conforming to schema $\mathcal{R}$, and given a mapping $\mathcal{M}$, the *virtual ABox* generated from $D$ by a mapping assertion $m = \Phi(x) \rightsquigarrow \Psi(y,t)$ in $\mathcal{M}$ is $m(D) = \bigcup_{v \in eval(\Phi,D)} \Psi[x/v]$, where $eval(\Phi, D)$ denotes the evaluation of the SQL query $\Phi$ over $D$, and where we consider $\Psi[x/v]$ to be a set of atoms (as opposed to a conjunction). Then, the ABox generated from $D$ by the mapping $\mathcal{M}$ is $\mathcal{M}(D) = \bigcup_{m \in \mathcal{M}} m(D)$. Notice that $\text{ADOM}(\mathcal{M}(D)) \subseteq \mathcal{C}$. As for ABoxes, the active domain $\text{ADOM}(D)$ of a database instance $D$ is the set of values occurring in $D$. Notice that $\text{ADOM}(D) \subseteq \mathcal{V}$. Given an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and a database instance $D$ for $\mathcal{R}$, a *model* for $\mathcal{O}$ wrt $D$ is a model of the ontology $(\mathcal{T}, \mathcal{M}(D))$. We say that $\mathcal{O}$ wrt $D$ is satisfiable if it admits a model wrt $D$.

*Example 2.* Consider a database instance $D = \{\mathsf{ENROLLED}(123, \mathsf{john}, \mathsf{doe}, \mathsf{Bachelor}, \mathsf{NULL})\}$. The corresponding virtual ABox obtained from the application of the mapping $\mathcal{M}$ is $\mathcal{M}(D) = \{\mathsf{Bachelor}(\mathsf{stu}_1(\mathsf{john}, \mathsf{doe}, \mathsf{Bachelor})), \mathsf{MNum}(\mathsf{stu}_1(\mathsf{john}, \mathsf{doe}, \mathsf{Bachelor}), \mathit{val}(123))\}$. □

An UCQ $q$ over an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ is simply an UCQ over $\mathcal{T}$. To compute the certain answers of $q$ over $\mathcal{O}$ wrt a database instance $D$ for $\mathcal{R}$, we follow a three-step approach: *(i)* $q$ is *rewritten* to compile away $\mathcal{T}$, obtaining $q_r = \text{REW}(q, \mathcal{T})$; *(ii)* the mapping $\mathcal{M}$ is used to *unfold* $q_r$ into a query over $\mathcal{R}$, denoted by $\text{UNFOLD}(q_r, \mathcal{M})$, which turns out to be an SQL query [17]; *(iii)* such a query is executed over $D$, obtaining the certain answers. For an ECQ, we can proceed in a similar way, applying the rewriting and unfolding steps to the embedded UCQs. It follows that computing certain answers to UCQs/ECQs in an OBDA system is FOL rewritable. Applying the unfolding step to $\mathsf{q}_{\mathsf{unsat}}(\mathcal{T})$, we obtain also that satisfiability in $\mathcal{O}$ is FOL rewritable.

## 3 Semantically-Governed Data-Aware Processes

A Semantically-Governed Data-Aware Process (SGDAP) $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ is formed by an OBDA System $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ by a process component $\mathcal{P}$, and by an initial database instance $D_0$ that conforms to the relational schema $\mathcal{R}$ in $\mathcal{O}$. Intuitively, the OBDA system keeps all the data of interest, while the process component modifies and evolves such data, starting from the initial database $D_0$.

The process component $\mathcal{P}$ constitutes the progression mechanism for the SGDAP. Formally, $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$, where: *(i)* $\mathcal{F}$ is a finite set of *functions* representing calls to *external services*, which return values; *(ii)* $\mathcal{A}$ is a finite set of *actions*, whose execution progresses the data layer, and may involve external service calls; *(iii)* $\pi$ is a finite set of *condition-action rules* that form the specification of the overall *process*, which tells at any moment which actions can be executed.

An *action* $\alpha \in \mathcal{A}$ has the form $\alpha(p_1, \ldots, p_n) : \{e_1, \ldots, e_m\}$, where: *(i)* $\alpha(p_1, \ldots, p_n)$ is the *signature* of the action, constituted by a name $\alpha$ and a sequence $p_1, \ldots, p_n$ of *input parameters* that need to be substituted with values for the execution of the action, and *(ii)* $\{e_1, \ldots, e_m\}$ is a set of *effect specifications*, whose specified effects are assumed to take place simultaneously. Each $e_i$ has the form $q_i^+ \land Q_i^- \rightsquigarrow E_i$, where: *(a)* $q_i^+ \land Q_i^-$ is a query over $\mathcal{R}$ whose terms are variables $\vec{x}$, action parameters, and constants from $\text{ADOM}(D_0)$. The query $q_i^+$ is a UCQ, and the query $Q_i^-$ is an arbitrary FO formula whose free variables are included in those of $q_i^+$. Intuitively, $q_i^+$ selects the tuples to instantiate the effect, and $Q_i^-$ filters away some of them. *(b)* $E_i$ is the effect, i.e., a set of facts for $\mathcal{R}$, which includes as terms: terms in $\text{ADOM}(D_0)$, input parameters, free variables of $q_i^+$, and in addition Skolem terms formed by applying a function $f \in \mathcal{F}$ to one of the previous kinds of terms. Such Skolem terms involving functions represent external service calls and are interpreted so as to return a value chosen by an external user/environment when executing the action.

The *process* $\pi$ is a finite set of *condition-action rules* $Q \mapsto \alpha$, where $\alpha$ is an action in $\mathcal{A}$ and $Q$ is a FO query over $\mathcal{R}$ whose free variables are exactly the parameters of $\alpha$, and whose other terms can be quantified variables or values in $\text{ADOM}(D_0)$.

*Example 3.* Consider the OBDA system $\mathcal{O}$ defined in Example 1. We now define a process component $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$ over the relational schema $\mathcal{R}$ of $\mathcal{O}$, so as to obtain a full SGDAP.

In particular, $\pi$ is constituted by the following condition-action rules ('_' denotes existentially quantified variables that are not used elsewhere):

- $\mathsf{ENROLLED}(\mathsf{id}, \_, \_, \_, \mathtt{NULL}) \rightsquigarrow \textsc{graduate}(\mathsf{id})$
- $\mathsf{TRANSF\_M}(\mathsf{name}, \mathsf{surname}) \rightsquigarrow \textsc{compl-enr}(\mathsf{name}, \mathsf{surname})$

The first rule extracts a matriculation number id of a currently enrolled student (endDate=NULL) from the ENROLLED relation and graduates the student, whereas the second rule selects a pair name surname in TRANSF_M and use them to complete the enrollment of that student. In order to be effectively executed, the involved actions rely on the following set $\mathcal{F}$ of service calls: *(i)* today() returns the current date; *(ii)* getMark(id, type) returns the final mark received by student id; *(iii)* getID(name, surname, type) returns the matriculation number for the name-surname pair of a student. The two actions GRADUATE and COMPL-ENR are then defined as follows:

$$
\begin{array}{ll}
\textsc{graduate}(\mathsf{id}): \{ & \mathsf{GRAD}(\mathsf{id}_2, \mathsf{m}, \mathsf{t}) \rightsquigarrow \mathsf{GRAD}(\mathsf{id}_2, \mathsf{m}, \mathsf{t}),\\
& \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s}) \rightsquigarrow \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s}),\\
& \mathsf{ENROLLED}(\mathsf{id}_2, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{d}) \wedge \mathsf{id}_2 \neq \mathsf{id} \rightsquigarrow \mathsf{ENROLLED}(\mathsf{id}_2, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{d}),\\
& \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathtt{NULL}) \rightsquigarrow \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{today}()),\\
& \mathsf{ENROLLED}(\mathsf{id}, \_, \_, \mathsf{t}, \mathtt{NULL}) \rightsquigarrow \mathsf{GRAD}(\mathsf{id}, \mathsf{getMark}(\mathsf{id}, \mathsf{t}), \mathsf{t}) \};\\
\textsc{compl-enr}(\mathsf{n}, \mathsf{s}): \{ & \mathsf{GRAD}(\mathsf{id}, \mathsf{m}, \mathsf{t}) \rightsquigarrow \mathsf{GRAD}(\mathsf{id}, \mathsf{m}, \mathsf{t}),\\
& \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}_2, \mathsf{s}_2, \mathsf{t}, \mathsf{d}) \rightsquigarrow \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}_2, \mathsf{s}_2, \mathsf{t}, \mathsf{d}),\\
& \mathsf{TRANSF\_M}(\mathsf{n}_2, \mathsf{s}_2) \wedge (\mathsf{n}_2 \neq \mathsf{n} \vee \mathsf{s}_2 \neq \mathsf{s}) \rightsquigarrow \mathsf{TRANSF\_M}(\mathsf{n}_2, \mathsf{s}_2),\\
& \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s}) \rightsquigarrow \mathsf{ENROLLED}(\mathsf{getID}(\mathsf{n}, \mathsf{s}, \text{"Master"}), \mathsf{n}, \mathsf{s}, \text{"Master"}, \mathsf{NULL}) \}
\end{array}
$$

Given a matriculation number id, action GRADUATE inserts a new tuple for id in GRAD, updating at the same time the enrollment's end date for id in ENROLLED to the current date, while keeping all other entries in TRANSF_M, GRAD and ENROLLED. Given a name and surname, action COMPL-ENR has the effect of moving the corresponding tuple in TRANSF_M to a new tuple in ENROLLED, for which the matriculation number is obtained by interacting with the getID service call; all other entries TRANSF_M, GRAD and ENROLLED are preserved. □

## 4 Semantics of SGDAP

This work focuses on the semantics of SGDAP assuming that *external services behave nondeterministically*, i.e., two calls of a service with the same arguments may return different results during the same run. This captures both services that model a truly nondeterministic process (e.g., human operators), and services that model stateful servers.

Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be a SGDAP where $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$. The semantics of $\mathcal{S}$ is defined in terms of a possibly infinite transition system (TS), which represents all possible computations that the process component can do over the data starting from $D_0$. We start by defining the semantics of *action execution*. Let $\alpha$ be an action in $\mathcal{A}$ of the form $\alpha(\vec{p}) : \{e_1, \ldots, e_n\}$ with effects $e_i = q_i^+ \wedge Q_i^- \rightsquigarrow E_i$, and let $\sigma$ be a substitution of $\vec{p}$ with values in $\mathcal{V}$. The evaluation of the effects of $\alpha$ on a database instance $D$ using a substitution $\sigma$ is captured by the following function:

$$\textsc{do}(D, \alpha, \sigma) = \bigcup_{q_i^+ \wedge Q_i^- \rightsquigarrow E_i \text{in } \alpha} \bigcup_{\theta \in \textsc{ans}((q_i^+ \wedge Q_i^-)\sigma, D)} E_i \sigma \theta$$

which returns a database instance made up of values in $\mathcal{V}$ and Skolem terms representing service calls. We denote with $\textsc{calls}(\textsc{do}(D, \alpha, \sigma))$ such service calls, and with $\textsc{evals}(D, \alpha, \sigma)$ the set of substitutions that replace these service calls with values in $\mathcal{V}$:

$$\textsc{evals}(D, \alpha, \sigma) = \{\theta \mid \theta : \textsc{calls}(\textsc{do}(D, \alpha, \sigma)) \to \mathcal{V} \text{ is a total function}\}.$$

We then say that the database instance $D'$ over $\mathcal{V}$ and conforming to $\mathcal{R}$ is *produced* from $D$ by the application of action $\alpha$ using substitution $\sigma$ if $D' = \textsc{do}(D, \alpha, \sigma)\theta$, where $\theta \in \textsc{evals}(D, \alpha, \sigma)$.

**Relational Layer Transition System (RTS).** Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be a SGDAP with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$. The RTS $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ of $\mathcal{S}$ is formally defined as $\langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$, where $\Sigma$ is a (possibly infinite) set of states, $s_0$ is the initial state, $db$ is a total function from states in $\Sigma$ to database instances made up of values in $\mathcal{V}$ and conforming to $\mathcal{R}$, and $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation. $\Sigma$, $\Rightarrow$ and $db$ are defined by simultaneous induction as the smallest sets such that $s_0 \in \Sigma$, with $db(s_0) = D_0$, and satisfying the following property: Given $s \in \Sigma$, for each condition-action rule $Q(\vec{p}) \mapsto \alpha(\vec{p}) \in \pi$, for each substitution $\sigma$ of $\vec{p}$ such that $\sigma \in \textsc{ans}(Q, D)$, consider every database instance $D'$ produced from $D$ by the application of $\alpha$ using $\sigma$. Then: *(i)* if there exists $s' \in \Sigma$ such that $db(s') = D'$, then $s \Rightarrow s'$; *(ii)* otherwise, if $\mathcal{O}$ is *satisfiable* wrt $D'$, then $s' \in \Sigma$, $s \Rightarrow s'$ and $db(s') = D'$, where $s'$ is a fresh state. We observe that the satisfiability check done in the last step of the RTS construction accounts for *semantic governance*.

**Semantic Layer Transition System (STS).** Given a SGDAP $\mathcal{S}$ with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and with RTS $\Upsilon_{\mathcal{S}}^{\mathrm{R}} = \langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$, the STS $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ of $\mathcal{S}$ is a "virtualization" of the RTS in the semantic layer. In particular, $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ maintains the structure of $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ unaltered, reflecting that the process component is executed over the relational layer, but it associates each state to a virtual ABox obtained from the application of the mapping $\mathcal{M}$ to the database instance associated by $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ to the same state. Formally, $\Upsilon_{\mathcal{S}}^{\mathrm{S}} = \langle \mathcal{T}, \Sigma, s_0, \mathit{abox}, \Rightarrow \rangle$, where *abox* is a total function from $\Sigma$ to ABoxes made up of individual objects in $\mathcal{C}$ and conforming to $\mathcal{T}$, such that for each $s \in \Sigma$ with $db(s) = D$, $\mathit{abox}(s) = \mathcal{M}(D)$.

## 5 Dynamic Constraints Formalism

Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be an SGDAP where $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$. We are interested in the verification of *conceptual temporal properties* over $\mathcal{S}$, i.e., properties that constrain the dynamics of $\mathcal{S}$ understood at the semantic layer. Technically, this means that properties are verified over the SGDAP's STS $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$, combining temporal operators with queries posed over the ontologies obtained by combining the TBox $\mathcal{T}$ with the ABoxes associated to the states of $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$. More specifically, we adopt ECQs [9] to query the ontologies of $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$, and $\mu$-calculus [15] to predicate over the dynamics of $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$.

We use a variant of $\mu$-calculus [15], one of the most powerful temporal logics subsuming LTL, PSL, and CTL* [13], called $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$, whose formulae have the form:

$$\Phi ::= Q \mid Z \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x \in \mathcal{C}_0.\Phi \mid \langle - \rangle \Phi \mid \mu Z.\Phi$$

where $Q$ is an ECQ over $\mathcal{T}$, $\mathcal{C}_0 = \textsc{adom}(\mathcal{M}(D_0))$ is the set of object terms appearing in the initial virtual ABox (obtained by applying the mapping $\mathcal{M}$ over the database instance $D_0$), and $Z$ is a predicate variable. As usual, syntactic monotonicity is enforced to ensure existence of unique fixpoints. Beside the usual FOL abbreviations, we also make use of the following ones: $[-]\Phi = \neg\langle - \rangle(\neg\Phi)$ and $\nu Z.\Phi = \neg\mu Z.\neg\Phi[Z/\neg Z]$. The subscript $C$ in $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ stands for "closed", and attests that ECQs are closed queries. In fact, $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ formulae only support the limited form of quantification $\exists x \in \mathcal{C}_0.\Phi$, which is a convenient, compact notation for $\bigvee_{c \in \text{ADOM}(\mathcal{M}(D_0))} \Phi[x/c]$. We make this assumption for simplicity, but actually, with some care, our result can be extended to a more general form of quantification over time [5].

In order to define the semantics of $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ we resort to transition systems. Let $\Upsilon = \langle \mathcal{T}, \Sigma, s_0, abox, \Rightarrow \rangle$ be an STS. Let $V$ be a predicate and individual variable valuation on $\Upsilon$, i.e., a mapping from the predicate variables $Z$ to subsets of the states $\Sigma$, and from individual variables to constants in $\text{ADOM}(\mathcal{M}(D_0))$. Then, we assign meaning to $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ formulas by associating to $\Upsilon$ and $V$ an *extension function* $(\cdot)_V^{\mathfrak{A}}$, which maps $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ formulas to subsets of $\Sigma$. The extension function $(\cdot)_V^{\mathfrak{A}}$ is defined inductively as:

$$\begin{array}{ll}
(Q)_V^{\mathfrak{A}} & = \{s \in \Sigma \mid \text{ANS}(QV, \mathcal{T}, abox(s)) = \mathsf{true}\} \\
(Z)_V^{\mathfrak{A}} & = V(Z) \subseteq \Sigma \\
(\neg\Phi)_V^{\mathfrak{A}} & = \Sigma - (\Phi)_V^{\mathfrak{A}} \\
(\Phi_1 \wedge \Phi_2)_V^{\mathfrak{A}} & = (\Phi_1)_V^{\mathfrak{A}} \cap (\Phi_2)_V^{\mathfrak{A}} \\
(\exists x \in \mathcal{C}_0.\Phi)_V^{\mathfrak{A}} & = \bigcup\{(\Phi)_{V[x/c]}^{\mathfrak{A}} \mid c \in \text{ADOM}(\mathcal{M}(D_0))\} \\
(\langle - \rangle\Phi)_V^{\mathfrak{A}} & = \{s \in \Sigma \mid \exists s'.\ s \Rightarrow s' \text{ and } s' \in (\Phi)_V^{\mathfrak{A}}\} \\
(\mu Z.\Phi)_V^{\mathfrak{A}} & = \bigcap\{\mathcal{E} \subseteq \Sigma \mid (\Phi)_{v[Z/\mathcal{E}],V}^{\mathfrak{A}} \subseteq \mathcal{E}\}
\end{array}$$

When $\Phi$ is a closed formula, $(\Phi)_V^{\mathfrak{A}}$ does not depend on $V$, and we denote it by $(\Phi)^{\mathfrak{A}}$. We are interested in the *model checking* problem, i.e., verify whether a $\mu\mathcal{L}_C^{EQL}$ *closed formula $\Phi$ holds for the SGDAP $\mathcal{S}$*. This problem is defined as checking whether $s_0 \in (\Phi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$, that is, whether $\Phi$ is true in the initial state $s_0$ of $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$. If it is the case, we write $\Upsilon_{\mathcal{S}}^{\mathrm{S}} \models \Phi$.

*Example 4.* An example of dynamic property in our running example is $\Phi = \mu Z.((\forall s.[\mathsf{Student(s)}] \rightarrow [\mathsf{Graduated(s)}]) \vee [-]Z)$, which says that every evolution of the system leads to a state in which all students present in that state are graduated. □

## 6 Verification of Dynamic Properties over SGDAPs

We now describe how $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ properties can be effectively verified over SGDAPs. Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be an SGDAP where $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$. Let $\Phi$ be a $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ dynamic property specified over the $\mathcal{T}$, and let $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ and $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ respectively be the STS and RTS of $\mathcal{S}$. The main issue to be tackled is that $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ and $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ are in general infinite-state, and their verification undecidable. In [5], some decidability boundaries for the verification of Data-Centric Dynamic Systems (DCDSs) have been extensively studied. DCDSs are tightly related to SGDAPs, with some key differences in the data component: *(i)* the process component is identical in the two frameworks; *(ii)* DCDSs are only equipped with a relational layer, i.e., no ontology nor mapping are specified; *(iii)* while SGDAPs define constraints over the data at the semantic layer, DCDSs are equipped with denial constraints posed directly over the database schema. Given a $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ property $\Phi$, we therefore attack the verification problem $\Upsilon_{\mathcal{S}}^{\mathrm{S}} \models \Phi$ in the following way: (1) We transform $\Phi$ into a corresponding $\mu\mathcal{L}_C$ property $\Phi'$, i.e., a $\mu\mathcal{L}$ property whose atoms are closed FO queries over $\mathcal{R}$, thus reducing $\Upsilon_{\mathcal{S}}^{\mathrm{S}} \models \Phi$ to $\Upsilon_{\mathcal{S}}^{\mathrm{R}} \models \Phi'$. (2) We show, again exploiting FOL rewritability in *DL-Lite*$_{\mathcal{A}}$, that the consistency check used to generate $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ can be rewritten as denial constraints over $\mathcal{R}$. This means that $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ can be generated by a purely relational DCDS. (3) We argue that $\Phi'$ belongs to the dynamic

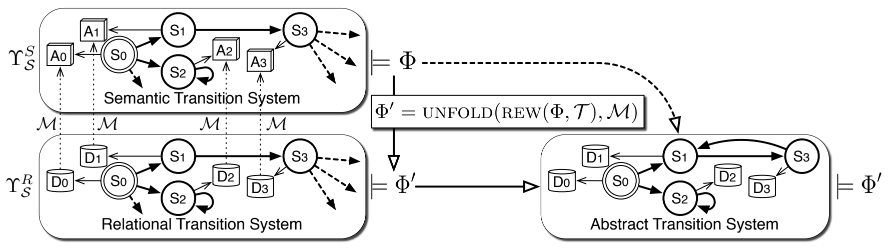
$\Upsilon_S^S$
$\Upsilon_S^R$
Semantic Transition System
Relational Transition System
Abstract Transition System
$\models \Phi$
$\Phi' = \text{UNFOLD}(\text{REW}(\Phi, \mathcal{T}), \mathcal{M})$
$\models \Phi'$
$\mathcal{M}$
A0
A1
A2
A3
D0
D1
D2
D3
S0
S1
S2
S3

**Fig. 2.** Verification of dynamic $\mu\mathcal{L}_C^{\text{EQL}}$ properties over SGDAP

property language investigated in [5] for DCDSs under the nondeterministic semantics. (4) We can therefore reuse the decidability results of [5] to check whether $\Upsilon_S^R \models \Phi'$ can be decided and, in the positive case, we apply the abstraction technique defined in [5] for reducing the verification problem to conventional finite-state model checking. Details are provided below. The idea of the approach is depicted in Figure 2.

**Property Transformation.** In order to transform the property, we separate the treatment of the dynamic part and of the embedded ECQs. Since the dynamics of an SGDAP is completely determined at the relational layer, the dynamic part is maintained unaltered. ECQs are instead manipulated as defined in Section 2. In particular, the rewriting of $\Phi$ wrt the TBox $\mathcal{T}$, denoted by $\Phi_r = \text{REW}(\Phi, \mathcal{T})$, is done by replacing each embedded ECQ with its corresponding rewriting wrt $\mathcal{T}$.

*Example 5.* Consider the $\mu\mathcal{L}_C^{\text{EQL}}$ property $\Phi$ described in Example 4, together with the TBox $\mathcal{T}$ introduced in Example 1. The rewriting of $\Phi$ wrt $\mathcal{T}$ produces $\Phi_r = \text{REW}(\Phi, \mathcal{T})$, which is:

$$\mu Z.(\forall s.[\mathsf{Student(s)} \lor \mathsf{Bachelor(s)} \lor \mathsf{Master(s)} \lor \mathsf{MNum(s,\_)}] \rightarrow [\mathsf{Graduated}(s)]) \lor [-]Z$$

□

Before unfolding the rewritten dynamic property $\Phi_r$ we translate each subformula of the form $\exists x \in \mathcal{C}_0.\Psi$ into the equivalent form $\bigvee_{c \in \text{ADOM}(\mathcal{M}(D_0))} \Psi[x/c]$. This means that when such a form of quantification is used, the initial ABox must be *materialized* in order to compute the initial active domain of the semantic layer. We then extend the UNFOLD() function defined in Section 2 to unfold a $\mu\mathcal{L}_C^{\text{EQL}}$ dynamic property over the semantic layer into a corresponding property over the relational layer. As for the rewriting, the temporal structure is maintained unaltered, reflecting that the dynamics of SGDAPs is determined at the relational layer. For what concerns the ECQs embedded in the property, the interesting case to be discussed is the one of (existential) quantification:

$$\begin{aligned}\text{UNFOLD}(\exists x.\varphi, \mathcal{M}) = {} & \exists x.\text{UNFOLD}(\varphi, \mathcal{M}) \lor \\ & \textstyle\bigvee_{(f/n) \in \text{FS}(\mathcal{M})} \exists x_1, \ldots, x_n.\text{UNFOLD}(\varphi[x/f(x_1, \ldots, x_n)], \mathcal{M})\end{aligned}$$

where $\text{FS}(\mathcal{M})$ is the set of function symbols contained in $\mathcal{M}$. This unfolding reflects that quantification over individuals at the semantic layer must be properly rephrased as a corresponding quantification over those values in the relational layer that could lead to produce such individuals through the application of $\mathcal{M}$. This is done by unfolding $\exists x.\varphi$ into a disjunction of formulae, where: *(i)* the first formula corresponds to $\exists x.\varphi$ itself, and is used to tackle the case in which $x$ appears in the range of an attribute, which is in fact a value; *(ii)* Each of the other formulae is obtained from $\varphi$ by replacing $x$ with one of the possible variable terms produced by $\mathcal{M}$, and quantifying over the existence of values used to construct the corresponding object term.

*Example 6.* Let us consider the $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ property $\Phi_r$ of Example 5, together with the mapping $\mathcal{M}$ defined in Example 1. We get that $\textsc{unfold}(\Phi_r, \mathcal{M})$ corresponds to:

$$\mu Z. \Big(\forall x_1, x_2, x_3.\textsc{aux}_{m_3}(x_1, x_2, x_3, \_) \rightarrow \textsc{aux}_{m_5}(x_1, x_2, x_3)\Big) \vee [-]Z$$

where $\textsc{aux}_{m_3}(\mathsf{name}, \mathsf{surname}, \mathsf{type}, \mathsf{id})$ and $\textsc{aux}_{m_5}(\mathsf{name}, \mathsf{surname}, \mathsf{type})$ represent the auxiliary view predicates of mapping assertions $m_3$ and $m_5$ respectively, whose defining queries are the SQL queries in the left-hand side of the mapping assertion themselves. When unfolding the UCQ $\mathsf{Student}(\mathsf{stu}_1(x_1, x_2, x_3)) \vee \mathsf{Bachelor}(\mathsf{stu}_1(x_1, x_2, x_3)) \vee \mathsf{Master}(\mathsf{stu}_1(x_1, x_2, x_3)) \vee \mathsf{MNum}(\mathsf{stu}_1(x_1, x_2, x_3), \_)$, we notice that the involved mapping assertions are $m_1$, $m_2$, and $m_3$. However, we only consider $m_3$, because the query on its left-hand side contains the ones on the left-hand side of $m_1$ and $m_2$.

**Reduction to Data-Centric Dynamic Systems.** The connection between SGDAPs and DCDSs is straightforward (see [5] for the definition of DCDS). Given a SGDAP $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$, we can construct a corresponding DCDS with nondeterministic services $\mathcal{S}_{\mathrm{REL}} = \langle \mathcal{D}, \mathcal{P} \rangle$, where $\mathcal{D} = \langle \mathcal{V}, \mathcal{R}, \{\mathsf{q}_{\mathsf{unsat}}(\mathcal{T}) \rightarrow \mathsf{false}\}, D_0 \rangle$. Thanks to this encoding, we obtain $\Upsilon_{\mathcal{S}}^{\mathrm{R}} \equiv \Upsilon_{\mathcal{S}_{\mathrm{REL}}}^{\mathrm{DCDS}}$, where $\Upsilon_{\mathcal{S}_{\mathrm{REL}}}^{\mathrm{DCDS}}$ is the RTS constructed for the DCDS $\mathcal{S}_{\mathrm{REL}}$ following the definition in [5].

**Verification.** Leveraging on the parallel between SGDAPs and DCDSs, verification of a $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ property over a SGDAP can be reduced to the verification of a $\mu\mathcal{L}_C$ property over the corresponding DCDS. In fact, $\mu\mathcal{L}_C$ ($\mu$-calculus over closed FOL queries) is contained in the fragments of FO $\mu$-calculus studied for DCDSs in [5], namely $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$. Both $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$ support FOL queries over the DCDS, allowing for controlled forms of FO quantification across states, and therefore they clearly support FO sentences.

Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be a SGDAP with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$, STS $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ and $\Upsilon_{\mathcal{S}}^{\mathrm{R}} = \langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$. We say that $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ is *state-bounded* if there exists a bound $b$ such that for each $s \in \Sigma$, $|\textsc{adom}(db(s))| < b$. Let $\Phi$ be a $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ property, and let $\Phi' = \textsc{unfold}(\textsc{rew}(\Phi, \mathcal{T}), \mathcal{M})$. Since *(i)* $\Upsilon_{\mathcal{S}}^{\mathrm{S}} \models \Phi$ can be reduced to $\Upsilon_{\mathcal{S}}^{\mathrm{R}} \models \Phi'$, *(ii)* $\Phi'$ belongs to $\mu\mathcal{L}_C$ (which is contained in $\mu\mathcal{L}_P$), *(iii)* $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ can be generated by a DCDS with nondeterministic services, we can reuse the decidability results presented in [5]. In particular, we obtain that *$\Upsilon_{\mathcal{S}}^{\mathrm{S}} \models \Phi$ is decidable if $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ is state bounded*. Verification can in this case be reduced to conventional finite-state model checking.

*Example 7.* Consider the SGDAP $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$, where $\mathcal{O}$ is the OBDA system defined in Example 1, $\mathcal{P}$ the process component defined in Example 3. It is easy to see that the resulting RTS $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ is state-bounded. Intuitively, this follows from the facts that the actions of $\mathcal{S}$ either move tuples from the TRANSF_M table to the ENROLLED one, or copy tuples from the ENROLLED table to the GRAD one. Hence, the size of each database instance appearing in $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ is at most twice the size of $D_0$, thus verification of $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ properties over the STS $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ is decidable. □

**Acknowledgements.** This research has been partially supported by the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), funded by the EU under FP7 ICT Call 5, 2009.1.2, grant agreement No. FP7-257593.

## References

1. S. Abiteboul, P. Bourhis, A. Galland, and B. Marinoiu. The AXML artifact model. In *Proc. of TIME 2009*, pages 11–17, 2009.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
3. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, and R. De Masellis. Verification of conjunctive-query based semantic artifacts. In *Proc. of DL 2011*, volume 745 of *CEUR,* `ceur-ws.org`, 2011.
4. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli. Foundations of relational artifacts verification. In *Proc. of BPM 2011*, volume 6896 of *LNCS*, pages 379–395. Springer, 2011.
5. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. CoRR Technical Report arXiv:1203.0024, arXiv.org e-Print archive, 2012. Available at `http://arxiv.org/abs/1203.0024`.
6. F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of deployed artifact systems via data abstraction. In *Proc. of ICSOC 2011*, 2011.
7. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proc. of BPM 2007*, volume 4714 of *LNCS*, pages 288–234. Springer, 2007.
8. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodríguez-Muro, and R. Rosati. Ontologies and databases: The *DL-Lite* approach. In S. Tessaris and E. Franconi, editors, *Semantic Technologies for Informations Systems – 5th Int. Reasoning Web Summer School (RW 2009)*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009.
9. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of IJCAI 2007*, 2007.
10. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.
11. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-based identification constraints in description logics. In *Proc. of KR 2008*, pages 231–241, 2008.
12. P. Cangialosi, G. De Giacomo, R. De Masellis, and R. Rosati. Conjunctive artifact-centric services. In *Proc. of ICSOC 2010*, volume 6470 of *LNCS*, pages 318–333. Springer, 2010.
13. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, Cambridge, MA, USA, 1999.
14. D. Cohn and R. Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Bull. on Data Engineering*, 32(3):3–9, 2009.
15. E. A. Emerson. Automated temporal reasoning about reactive systems. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 41–101. Springer, 1996.
16. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
17. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.
18. M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work in practice. In *Proc. of AMW 2011*, volume 749 of *CEUR,* `ceur-ws.org`, 2011.
19. W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer. Proclets: A framework for lightweight interacting workflow processes. *Int. J. of Cooperative Information Systems*, 10(4):443–481, 2001.

# Knowledge-intensive Processes: An Overview of Contemporary Approaches[⋆]

Claudio Di Ciccio, Andrea Marrella, and Alessandro Russo

Sapienza Università di Roma, Rome, Italy
{cdc,marrella,arusso}@dis.uniroma1.it

**Abstract.** Engineering of knowledge-intensive processes is far from being mastered. Processes are defined knowledge-intensive when people/agents carry them out in a fair degree of "uncertainty", where the uncertainty depends on different factors, such as the high number of tasks to be represented, their unpredictable nature, or their dependency on the scenario. In the worst case, there is no pre-defined view of the knowledge-intensive process, and tasks are mainly discovered as the process unfolds. In this work, starting from three different real scenarios, we present a critical comparative analysis of the existing approaches used for supporting knowledge-intensive processes, and we discuss some recent research techniques that may complement or extend the existing state of the art.

**Keywords:** Knowledge-intensive Processes, Process Management Systems, Health Care, Process Adaptation, Process Mining

## 1 Introduction

Process management systems (PMSs) hold the promise of facilitating the everyday operation of many enterprises and work environments. However, PMSs remain especially useful in a limited range of applications where business processes can be described with relative ease. Current modeling techniques are used to codify processes that are completely predictable: all possible paths along the process are well-understood, and the process participants never need to make a decision about what to do next, since the workflow is completely determined by their data entry or other attributes of the process. This kind of highly-structured work includes mainly production and administrative processes. However, most business functions involve collaborative features and unstructured processes that do not have the same level of predictability as the routine structured work [58].

In [29] processes have been classified on the basis of their "degree of structure". Traditional PMSs perform well with fully *structured processes* and controlled interactions between participants. A major assumption is that such processes, after having been modeled, can be repeatedly instantiated and executed

⋆ This work has been partly supported by the SAPIENZA grant TESTMED and by the EU Commission through the project SmartVortex in a predictable and controlled manner. However, even for structured processes, the combination and sequence of tasks may vary from instance to instance due to changes in the execution context such as user preferences, or modifications in the environment such as exceptions and changes in the business rules. In such cases (*structured processes with ad hoc exceptions*), processes should be adapted accordingly (e.g. by adding, removing or generating an alternative sequence of activities). In general, structured processes can be described by an explicit and accurate model. But in scenarios where processes are to a large extent unclear and/or unstructured, process modeling cannot be completed prior to execution (due to lack of domain knowledge a priori or to the complexity of task combinations). Hence the classical axiom "first model, then execute" – valid for the enactment of structured processes – fails. As processes are executed and knowledge is acquired via experience, it is needed to go back to the process definitions and correct them according to work practices. This is the case of *unstructured processes with predefined fragments*, where processes cannot be anticipated, and thus cannot be studied or modeled as a whole. Instead, what can be done is to identify and study a set of individual activities, and then try to understand the ways in which these activities can precede or follow each other. At the end of the classification lies the category of *unstructured processes*, where it is impossible to define a priori the exact steps to be taken in order to complete an assignment. Since there is no pre-defined view of the process, process steps are discovered as the process scenario unfolds, and might involve decisions not based on some "codified policy", but on the user expertise applied on the scenario at hand.

The class of *knowledge-intensive processes* is transversal with respect to the classification proposed in [29]. In the literature, different definitions have been proposed about what does "knowledge-intensive" mean for a business process. In [24] a process is defined as knowledge intensive if its value can only be created through the fulfillment of the knowledge requirements of the process participants, while Davenport recognizes the knowledge intensity by the diversity and uncertainty of process input and output [11]. In our view, a knowledge-intensive process is characterized by activities that can not be planned easily, may change on the fly and are driven by the contextual scenario that the process is embedded in. The scenario dictates who should be involved and who is the right person to execute a particular step, and the set of users involved may be not formally defined and be discovered as the process scenario unfolds. Collaborative interactions among the users typically is a major part of such processes, and new process steps might have to be defined at run time on the basis of contextual changes. Despite the popularity of commercial PMSs, there is still a lack of maturity in managing such processes, i.e., a lack of a semantic associated to the models or an easy way to reason about that semantic.

In this paper, starting from three different real application scenarios, we present a critical and comparative analysis of the existing approaches used for supporting knowledge-intensive processes, and we discuss some recent research techniques which may complement or extend the existing state of the art. The rest of the paper is organized as follows. Section 2 discusses the role of knowledge-intensive processes in the health-care domain, mainly focusing on how different modeling approaches can contribute to the process representation and execution. Section 3 discusses the use of knowledge-intensive processes for supporting the work in highly dynamic scenarios, by focusing on the challenging aspect of process adaptation. Section 4 traces the evolution of process mining, from the beginnings up to the current open challenge of discovering flexible models for knowledge-intensive partially structured processes, along with the graphical models proposed for presenting them to the user. Finally, Section 5 concludes the paper.

## 2 Modeling Approaches for Healthcare Processes

Healthcare is widely recognized as one of the most promising, yet challenging, domains for the adoption of process-oriented solutions able to support both organizational and clinical processes [10,31,46,30]. Organizational processes, which also include administrative tasks (patient admission/discharge, appointment scheduling, etc.), are typically structured, stable and repetitive, and represent the ideal setting for the application of traditional approaches for process automation and improvement. On the other side, the knowledge-intensive nature and flexibility requirements of medical treatment processes [3,37] pose challenges that existing process management approaches are not able to adequately handle. Although BPM solutions can potentially support these processes, in practice their uptake in healthcare is limited, mainly due to a generally perceived lack of flexibility [30]. Clinical decision making is highly knowledge-driven, as it depends on medical knowledge and evidence, on case- and patient-specific data, and on clinicians' expertise and experience. Patient case management is mainly the result of knowledge work, where clinicians act in response to relevant events and changes in the clinical context on a per-case basis, according to so-called diagnostic-therapeutic cycles based on the interleaving between observation, reasoning and action [31]. Clinical practices can not be captured by process models that require a complete specification of activities and their control/data flow, with the risk of constraining the clinicians and undermining the acceptance of proposed tools.

Despite these characteristics, in the last years the medical community has introduced Clinical Guidelines (CGs), in an attempt to improve care quality and reduce costs. CGs are "systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances"[21] and act as blueprints that guide the care delivery process and provide evidence-based recommendations. Consequently, many research groups have focused on computer-interpretable clinical guidelines (CIGs) and different languages have been proposed [49,42,61], which can be broadly classified as rule-based (e.g., Arden Syntax), logic-based (e.g., PROforma), network-based (e.g., EON) and workflow-based (e.g., Guide). Most of them follow a task-based paradigm where modeling primitives for representing actions, decisions and patient states are linked via scheduling and temporal constraints, often in a rigid flowchart-like structure, and many representation models are supported by systems that allow the definition and enactment of CGs [27]. This rapid evolution in medical informatics has occurred mainly independently of the advances in the BPM community. However, the recent shift in the BPM domain towards process flexibility, adaptation (see Section 3) and evolution [47,30] has led to reconsider the link with CIGs and investigate the benefits coming from the application of process-oriented approaches in the healthcare domain [36]. On the one side, pattern-based analyses of CIG languages have shown that the expressiveness of these models, although specifically developed for the medical domain, is comparable with (or even lower than) the expressiveness of process modeling languages [39]. On the other side, emerging declarative constraint-based approaches [40,32] have been investigated as a possible solution to achieve a high degree of flexibility, taking advantage of loosely specified process models. In this direction, the combination of procedural and declarative models is under investigation, in order to support healthcare processes with different degrees of structuredness.

After more than a decade of research activities, researchers and practitioners agree on three main points: *(i)* clinical procedures, based on semi-structured and unstructured decision making, can not be completely specified in advance nor fully automated; *(ii)* deviations and variations during the care process (as well as uncertainty and changes in the clinical context) represent the rule rather than the exception; *(iii)* process- and activity-centric models can not adequately represent and support clinical case management. One of the main limitations of existing approaches is that they often underestimate the knowledge and data dimension. As patient treatment is knowledge-driven, the focus should be not on automating the decision making process, but rather on supporting the clinician during this process, according to a "system suggests, user controls" approach [62] that makes available the appropriate data and relevant knowledge when needed or required. Any system intended to support CGs should allow for representing and integrating at a semantic level evolving medical knowledge, patient-related data (including conditions, medical history, prescribed treatments and medications, etc.), and the existing (sometimes unpredictable) interactions between patient conditions, treatments and medications. This focus on data and knowledge is producing a shift from a process management approach to a more flexible case management approach, well understood by clinicians (although mostly in the form of paper-based processes) but only partially investigated in the BPM area [60]. Process support requires object-awareness in the form of a full integration of processes with patient data models consisting of object types and object relations [30,5]. Domain-relevant objects (such as medical orders, clinical and lab reports, etc.), their attributes and their possible states need to be explicitly represented, along with their inter-relations, so as to define a rich information model. This data model enables the identification and definition of the activities that rely on the object-related information and act on it, producing changes on attribute values, relations and object states. As a result, a tight integration between data objects and process activities can be achieved. As object-awareness requires a data-driven process modeling and execution approach, based on object behavior and object interactions, process/activity-centric methodologies are being replaced by data-centric models evolving over time [7]. In the context of a CG, patient's clinical situation (referred to as patient state, scenario, or context [49]) is central and represent the shared knowledge that drives the decision making and evolves as a result of performed actions, made decisions and collected data. Conditions defined over patient state, along with temporal constraints, are typically used as entry/exit points for a guideline [61] and as eligibility criteria for specific actions [49]. During the collaboration-based patient management activities, clinicians have to react to internal (e.g., a change in patient's state) and external (e.g., availability of lab test results) events, that can occur in any sequence. Moreover, it is often not possible to predetermine which activities have to be executed and in which order when an event occurs: according to the diagnostic-therapeutic cycles mentioned before, the clinician first assesses and evaluate the situation and then acts or plans the actions to be performed. This suggests an interleaving and overlapping of modeling and execution, where the process is "created at the time it is executed". Any modeling and execution approach for supporting this view has to consider that the clinician should be guided by what *can* be done and not restricted by what *has* to be done [35]. Although the path to be followed can be initially unclear and is gradually determined by clinician decisions, the care process evolves through a series of intermediate goals or milestones to be achieved (e.g., bring a parameter back to a normal level) that can again be expressed as conditions or constraints over patient state.

Given the above scenario, a promising and emerging approach for modeling CGs and supporting their execution and management is the artifact-centric paradigm, which considers data and knowledge as an integral part of business processes [51]. It is based on the concept of business artifacts as an abstraction for business-relevant entities and data that evolve according to a lifecycle and drive the activities in a business setting. Activities are defined in the context of interrelated artifacts and become enabled as the result of triggering events (internal or external) constrained by conditions defined and evaluated over the artifacts. Events and conditions over artifacts can also be used to set specific goals and evaluate the progress towards their achievement. The scheduling of actions is thus event- and data-driven, rather than induced by direct control flow dependencies. Under this perspective, it emerges a clear correspondence between artifact-centric concepts and clinical case management, in particular if considering the Guard-Stage-Milestone (GSM) meta-model [51] as a representative example of the artifact-based paradigm. GSM builds on the concepts of information model and lifecycle model, where the latter includes milestones to be achieved, hierarchically organized stages as clusters of possible activities to be performed to achieve milestones, and guards, timed events and conditions that control the stages and determine milestones' achievement. The patient and his/her state, a diagnostic test, a treatment course can all be considered as artifact types and represented by an information model that evolves according to a lifecycle and captures all relevant data and relations (e.g., as a relational model or domain ontology). CGs could be seen as progressing through a set of stages, where each performed action, made decision or event occurrence is driven by (eligibility criteria mentioned before) and has an impact on patient state, as reflected in the underlying information model. The data-driven nature of the model facilitates the integration between process control knowledge and the patient-related and medical knowledge; in addition, the distinction between data attributes and status attributes can directly support an integrated and explicit representation of both patient and execution states, not provided by all CIG models [61,49]. Although artifact-centric models can open the way for a new generation of flexible and adaptive case management systems in healthcare, further investigation is needed to understand the contribution that these models can bring in solving well-known problems for CIGs; among them: *(i)* how to reconcile the decision-action nature of CGs with a declarative modeling approach than can be used and understood by clinicians and is able to represent the evidence-based knowledge contained in the CGs; *(ii)* how to define an information model that is able to capture all clinically relevant data and takes into account existing standards, models, and ontologies used in Electronic Medical Records (EMRs) for patient and medical data; *(iii)* to what extent clinical events and medical knowledge can be represented and encoded by rules and conditions; *(iv)* how can an artifact-centric model address the problems of guideline acquisition, verification, testing, tracing and evolution, and how to turn or customize abstract models in executable models that take into account additional information, such as resource availability, roles and local services, in a collaborative multi-user environment.

## 3 Process Adaptation in Highly Dynamic Scenarios

A recent open research question in the BPM field concerns how to tackle scenarios characterized by being very dynamic and subject to higher frequency of unexpected contingencies than classical scenarios, e.g., scenarios for emergency management. There, a PMS can be used to coordinate the activities of first responders on the field (e.g., reach a location, evacuate people from collapsed buildings, extinguish a fire, etc.). The use of processes for supporting the work in highly dynamic contexts has become a reality, thanks also to the growing use of mobile devices in everyday life, which offer a simple way for picking up and executing tasks. These kinds of processes are also named *dynamic processes*. A dynamic process usually includes a wide range of knowledge-intensive tasks; as the process proceeds, the sequence of tasks depends so much upon the specifics of the context (for example, which resources are available and what particular options exist at that time), and often it is unpredictable the way in how it unfolds. This is due to the high number of tasks to be represented and to their unpredictable nature, or to a difficulty to model the whole knowledge of the domain of interest at design time. If we refer again to the classification shown in [29], dynamic processes can be classified between structured processes with ad hoc exceptions and unstructured processes with predefined fragments.

Research efforts in this field try to enhance the ability of dynamic processes and their support environments to modify their behavior in order to deal with contextual changes and exceptions that may occur in the operating environment during process enactment and execution. On the one hand, existing PMSs like YAWL [50] provide the support for the handling of expected exceptions. The process schemas are designed in order to cope with potential exceptions, i.e., for each kind of exception that is envisioned to occur, a specific contingency process (a.k.a. exception handler or compensation flow) is defined. On the other hand, adaptive PMSs like ADEPT2 [65] support the handling of unanticipated exceptions, by enabling different kinds of ad-hoc deviations from the pre-modeled process instance at run-time, according to the structural process change patterns defined in [64].

However, traditional approaches that try to anticipate how the work will happen by solving each problem at design time, as well as approaches that allow to manually change the process structure at run time, are often ineffective or not applicable in rapidly evolving contexts. The design-time specification of all possible compensation actions requires an extensive manual effort for the process designer, that has to anticipate all potential problems and ways to overcome them in advance, in an attempt to deal with the unpredictable nature of this kind of processes. Moreover, the designer often lacks the needed knowledge to model all the possible contingencies, or this knowledge can become obsolete as process instances are executed and evolve, by making useless his/her initial effort. In general, for a dynamic process there is not a clear, anticipated correlation between a change in the context and corresponding process changes, since the process may be different every time it runs and the recovery procedure strictly depends on the actual contextual information. For the same reason, it is also difficult to manually define an ad-hoc recovery procedure at run-time, as the correctness of the process execution is highly constrained by the values (or combination of values) of contextual data. Dealing with dynamic processes require that PMSs provide intelligent failure handling mechanisms that, starting from the original process model, are able to adapt process instances without explicitly defining at design time all the handlers/policies to recover from exceptions and without the intervention of domain experts.
Recently, some techniques from the field of artificial intelligence (AI) have been applied to process management, with the purpose of improving the degree of automatic adaptation of dynamic processes. In [23], the authors present a concept for dynamic and automated workflow re-planning that allows recovering from task failures. To handle the situation of a partially executed workflow, a multi-step procedure is proposed that includes the termination of failed activities, the sound suspension of the workflow, the generation of a new complete process definition and the adequate process resumption. In [28], the authors take a much broader view of the problem of adaptive workflow systems, and show that there is a strong mapping between the requirements of such systems and the capabilities offered by AI techniques. In particular, the work describes how planning can be interleaved with process execution and plan refinement, and investigates plan

patching and plan repair as means to enhance flexibility and responsiveness. A new life cycle for workflow management based on the continuous interplay between learning and planning is proposed in [20]. The approach is based on learning business activities as planning operators and feeding them to a planner that generates the process model. The main result is that it is possible to produce fully accurate process models even though the activities (i.e., the operators) may not be accurately described. The approach presented in [45] highlights the improvements that a legacy workflow application can gain by incorporating planning techniques into its day-to-day operation. The use of contingency planning to deal with uncertainty (instead of replanning) increases system flexibility, but it does suffer from a number of problems. Specifically, contingency planning is often highly time-consuming and does not guarantee a correct execution under all possible circumstances. Planning techniques are also used in [22] to define a self-healing approach for handling exceptions in service-based processes and repairing faulty activities with a model-based approach. During the process execution, when an exception occurs, a new repair plan is generated by taking into account constraints posed by the process structure and by applying or deleting actions taken from a given generic repair plan, defined manually at design time.

An interesting approach for dealing with exceptional changes has been proposed in [13,34]. Here, it is presented SmartPM (Smart Process Management), a model and a proof-of-concept PMS featuring a set of techniques providing support for automatic adaptation of processes. In SmartPM, a process model is defined as a set of $n$ task definitions, where each task $t_i$ can be considered as a single step that consumes input data and produces output data. Data are represented through some process variables whose definition depends strictly on the specific process domain of interest. The model allows to define logical constraints based on process variables through a set $F$ of predicates $f_j$. Such predicates can be used to constrain the task assignment (in terms of *task preconditions*), to assess the outcome of a task (in terms of *task effects*) and as guards into the expressions at decision points (e.g., for cycles or conditional statements). Choosing the predicates that are used to describe each activity falls into the general problem of *knowledge representation*. To this end, the environment, services and tasks are grounded in domain theories described in Situation Calculus [48]. Situation Calculus is specifically designed for representing dynamically changing worlds in which all changes are the result of the tasks' execution. Processes are represented as IndiGolog programs. IndiGolog [12] allows for the definition of programs with cycles, concurrency, conditional branching and interrupts that rely on program steps that are actions of some domain theory expressed in Situation Calculus. The dynamic world of SmartPM is modeled as progressing through a series of *situations*. Each situation is the result of various tasks being performed so far. Predicates may be thought of as "properties" of the world whose values may vary across situations. SmartPM provides mechanisms for adapting process schemas that require no pre-defined handlers. Specifically, adaptation in SmartPM can be seen as reducing the gap between the *expected reality*, the (idealized) model of reality that is used by the PMS to reason, and the *physical reality*, the real world with the actual values of conditions and outcomes. The physical reality $\Phi_s$ reflects the concept of "now", i.e., what is happening in the real environment whilst the process is under execution. In general, a task $t_i$ can only be performed in a given physical reality $\Phi_s$ if and only if that reality satisfies the *preconditions* $Pre_i$ of that task. Moreover, each task has also a set of *effects* $Eff_i$ that change the current physical reality $\Phi_s$ into a new physical reality $\Phi_{s+1}$. At execution time, the process can be easily invalidated because of task failures or since the environment may change due to some external event. For this purpose, the concept of *expected reality* $\Psi_s$ is given. A recovery procedure is needed if the two realities are different from each other. An execution monitor is responsible for detecting whether the gap between the expected and physical realities is such that the original process $\delta_0$ cannot progress its execution. In that case, the PMS has to find a recovery process $\delta_h$ that repairs $\delta_0$ and removes the gap between the two kinds of reality. Currently, the adaptation algorithm deployed in SmartPM synthesizes a linear process $\delta_h$ (i.e., a process consisting of a sequence of tasks) and inserts it at a given point of the original process - specifically, that point of the process where the deviation was first noted. This means that such technique is able to automatically recover from exceptions without defining explicitly any recovery policy.

## 4 Mining

*Process Mining* [54], also referred to as *Workflow Mining* [53], is the set of techniques that allow the extraction of process descriptions, stemming from a set of recorded executions. Throughout this Section, we will investigate the techniques adopted, along with the notations used to display the results, i.e., the mined processes. To date, ProM [55] is one of the most used plug-in based software environment for implementing workflow mining techniques. The idea to apply process mining in the context of workflow management systems was introduced in [1]. There, processes were modelled as directed graphs where vertices represented individual activities and edges stood for dependencies between them. Cook and Wolf, at the same time, investigated similar issues in the context of software engineering processes. In [8] they described three methods for process discovery: *(i)* neural network-based, *(ii)* purely algorithmic, *(iii)* adopting a Markovian approach. The authors considered the latter two as the most promising. Although, the results presented in [8] were limited to sequential behavior only. The nowadays mainstream process mining algorithms and management tools model processes with a graphical syntax derived from a subset of Petri Nets, i.e., Workflow Nets (WfN [53]), explicitly designed to represent the control-flow dimension of a workflow. See [41] for a history of Petri nets and an extensive bibliography. From [1] onwards many techniques have been proposed, in order to address specific issues: pure algorithmic (e.g., $\alpha$ algorithm [59] and its evolution $\alpha^{++}$ [67]), heuristic (e.g., [66]), genetic (e.g., [38]). Heuristic and genetic algorithms were introduced to cope with noise, that the pure algorithmic techniques were not able to manage. Whereas algorithmic processes rely on footprints of traces (i.e., tables reporting whether events appeared before or afterwards, if decidable) to determine the workflow net that could have generated them, heuristic approaches build a representation similar to causal nets, taking frequencies of events and sequences into account when constructing the process model, in order to ignore infrequent paths. Genetic process mining adopts an evolutionary approach to the discovery and differs from the other two in that its computation evolves in a non-deterministic way: the final output, indeed, is the result of a simulation of a process of natural selection and evolutionary reproduction of the procedures used to determine the final outcome. A very smart extension to the previous research was achieved by the two-steps algorithm proposed in [52]. Differently from previous works, in which the proposed approaches provide a single process mining step, it splitted the computation in two phases: the first built a Transition System that represents the process behavior and the tasks causal dependencies; the second made use of the state-based "theory of regions" [9,15] to construct a Petri Net bisimilar to the Transition System. The first phase was made "tunable", so that it could be either more strictly adhering or more permissive to the analyzed log traces behavior, i.e., the expert could determine a balance between "overfitting" and "underfitting". Indeed, past execution traces are not the whole universe of possible ones that may run: hence, the extracted process model should be valid for future unpredictable cases, on one hand, nevertheless checking whether the latter actually adhere to the common behavior, on the other hand. This issue reveals to be particularly relevant in the field of knowledge-intensive processes.

To date, the majority of research relating to processes coped with structured business processes. [26] discusses about a particular class of knowledge-intensive processes, named "artful business processes"; they are typically carried out by those people whose work is mental rather than physical (managers, professors, researchers, etc.), the so called "knowledge workers" ([63]). With their skills, experience and knowledge, they are used to perform difficult tasks which require complex, rapid decisions among multiple possible strategies, in order to fulfill specific goals. In contrast to business processes that are formal and standardized, informal processes are not even written down, often, let alone defined formally, and can vary from person to person even when those involved are pursuing the same objective. Knowledge workers create informal processes "on the fly" to cope with many of the situations which arise in their daily work. While informal processes are frequently repeated, because they are not written down, they are not exactly reproducible, even by their originators, nor can they be easily shared. [63] described the "ACTIVE" EU collaborative project, coordinated by British Telecom. Such project addressed the need for greater knowledge worker productivity by providing more effective and efficient tools. Among the main objectives, it aimed at helping users to share and reuse informal processes, even by learning those processes from the user's behavior. Basing on the work of [6] and [56], [19] investigated the challenge of mining these processes out of semi-structured texts, i.e., the email conversations exchanged among knowledge workers, through the interplay of text mining, object matching and process mining techniques. It provided an architectural overview of the application (named MailOfMine) able to fulfill the objective.

The need for flexibility in the definition of some types of process, such as artful business processes, leads to an alternative to the classical "imperative" approach: the "declarative". Rather than using a procedural language for expressing the allowed sequences of activities, it is based on the description of workflows through the usage of constraints: the idea is that every task can be performed, except what does not respect them. [58] showed how the declarative approach can help in obtaining a fair trade-off between flexibility in managing collaborative processes and support in controlling and assisting the enactment of workflows. DecSerFlow [57] and ConDec [43], now under the name of Declare [44], define such constraints as formulations in Linear Temporal Logic. [33] outlines an algorithm for mining Declare processes, integrated in ProM (namely, Declare Miner). The tool is based on the translation of Declare constraints into automata, and works in conjunction with the optimization techniques described in [68]. [4] describes the usage of inductive logic programming techniques to mine models expressed as a $\mathcal{S}$CIFF theory. $\mathcal{S}$CIFF theory is thus translated into the ConDec notation [43]. [2] differs from both [4] and [33] in that it does not directly verify the candidate constraints over the whole set of traces in input. It prepares an ad-hoc knowledge base of its own, instead, which specific queries are further submitted to. The model is determined on the base of the result of such queries. MINERful, proposed in [18], exploits this two-steps technique too, in order to improve the efficiency of the mining procedure. [17] proves the complexity of the algorithm to be polynomial w.r.t. the size of both the alphabet of constraints and the input traces. Differently from [33], [4] and [2], it is independent of the formalism adopted for representing constraints.

Declare provides a graphical model for representing declarative processes, useful to depict the constraints that hold between activities as a graph where nodes are activities and arcs are constraints among them. [25] and [16] presented a different approach to the graphical modelling. The former describes an event-based model, namely DCRGraph, showing the current state of the workflow at run-time, through the listing of tasks that can (either optionally or mandatorily) or can not be executed at the moment. A section describing the mapping of that notation to Büchi Automata is provided as well. The latter provides multiple graphical syntaxes, respectively depicting the process from two viewpoints: *(i) global*, i.e., focused on the representation of constraints between tasks, represented all together in a single graph and *(ii) local* i.e., focused instead on the constraints directly related to one single activity at a time. The first is then divided into a *base* and an *extended* version, in order to respectively depict less or more details about the nature of constraints that hold in the process – following the so called "map metaphor" [14]. The second is also twofold. The *static* view shows the constraints affecting an activity, which is put on the origin of a cartesian-like diagram. There, the implication and the temporal succession are aligned on orthogonal axes. The tasks involved in constraints related to the activity under analysis are put on different coordinates accordingly. In the *dynamic* view, the graph evolves as new tasks are executed. Starting from the initial, the enacted task is chained down to the previous. On the basis of the execution trace, the consequent next tasks are shown below the chain, in compliance with the constraints that hold at the moment.

## 5 Conclusions

In this work, we provided a critical and comparative analysis of the existing approaches used for supporting knowledge-intensive processes, and we showed some recent research techniques that may complement or extend the existing state of the art to this end.

In the health care domain, several challenges still need to be addressed and an interdisciplinary research effort is required. In this direction, the existing gap between the general evidence-based knowledge contained in CGs and the knowledge and information required to apply them to specific patients in local healthcare organizational contexts needs further investigation. Similarly, modeling approaches should allow to capture all "knowledge layers" and their possible interactions, including the procedural knowledge contained in CGs, the declarative knowledge representing domain- or site-specific constraints and properties, and clinicians' basic medical knowledge.

In highly dynamic environments, commercial PMSs are not able to deal with knowledge-intensive processes sufficiently, due to the static and only implicitly defined meta models of those systems. Basically, a dynamic process is largely dependent on the scenario at hand, and the result of process modeling is often a static plan of actions, which is difficult to adapt to changing procedures or to different business goals. In order to devise intelligent failure handling mechanisms for dynamic processes there is the need to define enriched workflow models, possibly with a declarative specification of process tasks, i.e., comprising the specification of input/output artefacts and task preconditions and effects. In general, the use of AI techniques for adapting dynamic processes seems very promising.

In the area of process mining, the declarative model proves to be very effective in allowing flexibility required by knowledge-intensive processes. Although, it has to be verified with people involved in those processes. E.g., the graphical notation proposed in [16] has to be implemented and its readability tested with real actors of those processes. A graphical notation representing the level of severity of a constraint in the process still misses. In the area of declarative workflow mining, it might be useful to determine the tightness of the discovered constraints on the basis of the frequency with which a constraint did not hold in the past. Moreover, a study on the impact of noise in such analysis could be done.

## References

1. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: EDBT'98 (1998)

2. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The sciff framework. ACM Trans. Comput. Log. 9(4) (2008)
3. Ammon, D., Hoffmann, D., Jakob, T., Finkeissen, E., Detschew, V., Wetter, T.: Management of Knowledge-Intensive Healthcare Processes on the Example of General Medical Documentation. In: BPM Workshops (2008)
4. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. T. Petri Nets and Other Models of Concurrency 2, 278–295 (2009)
5. Chiao, C.M., Künzle, V., Reichert, M.: Towards Object-aware Process Support in Healthcare Information Systems. In: eTELEMED 2012 (2012)
6. Cohen, W.W., Carvalho, V.R., Mitchell, T.M.: Learning to classify email into "speech acts". In: EMNLP. pp. 309–316. ACL (2004)
7. Combi, C., Gambini, M., Migliorini, S., Posenato, R.: Modelling temporal, data-centric medical processes. In: ACM SIGHIT IHI 2012 (2012)
8. Cook, J.E., Wolf, A.L.: Discovering models of software processes from event-based data. ACM Trans. Softw. Eng. Methodol. 7(3), 215–249 (1998)
9. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving petri nets from finite transition systems. IEEE Trans. on Computers 47(8), 859 –882 (1998)
10. Dadam, P., Reichert, M., Kuhn, K.: Clinical Workflows - The Killer Application for Process-oriented Information Systems? In: BIS'00 (2000)
11. Davenport, T.H.: Improving knowledge work processes. In: Sloan Management Review, vol. 37 (1996)
12. De Giacomo, G., Lespérance, Y., Levesque, H., Sardina, S.: IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents. In: Multi-Agent Prog.: Languages, Platforms and Applications (2009)
13. de Leoni, M., Marrella, A., Mecella, M., Sardina, S.: SmartPM – Featuring Automatic Adaptation to Unplanned Exceptions. Tech. rep., Sapienza Università di Roma (2011), `http://ojs.uniroma1.it/index.php/DIS_TechnicalReports/article/view/9221/9141`
14. de Leoni, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Visual support for work assignment in process-aware information systems. In: BPM'08 (2008)
15. Desel, J., Reisig, W.: The synthesis problem of petri nets. Acta Informatica 33, 297–315 (1996)
16. Di Ciccio, C., Catarci, T., Mecella, M.: Representing and visualizing mined artful processes in MailOfMine. In: HCI-KDD (2011)
17. Di Ciccio, C., Mecella, M.: MINERful, a mining algorithm for declarative process constraints in MailOfMine. Tech. rep., Sapienza Università di Roma (2012), `http://ojs.uniroma1.it/index.php/DIS_TechnicalReports/issue/view/416`
18. Di Ciccio, C., Mecella, M.: Mining constraints for artful processes. In: BIS'12 (2012)
19. Di Ciccio, C., Mecella, M., Scannapieco, M., Zardetto, D., Catarci, T.: MailOfMine - analyzing mail messages for mining artful collaborative processes. In: SIMPDA'11 (2011)
20. Ferreira, H., Ferreira, D.: An Integrated Life Cycle for Workflow Management Based on Learning and Planning. Int. J. Coop. Inf. Syst. 15 (2006)
21. Field, M.J., Lohr, K.N.: Clinical Practice Guidelines: Directions for a New Program. Institute of Medicine, Washington, DC (1990)
22. Friedrich, G., Fugini, M., Mussi, E., Pernici, B., Tagni, G.: Exception Handling for Repair in Service-Based Processes. IEEE Trans. on Soft. Eng. 36 (2010)
23. Gajewski, M., Meyer, H., Momotko, M., Schuschel, H., Weske, M.: Dynamic Failure Recovery of Generated Workflows. In: DEXA'05 (2005)

24. Gronau, N., Weber, E.: Management of knowledge intensive business processes. In: BPM'04 (2004)
25. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: PLACES'10 (2010)
26. Hill, C., Yates, R., Jones, C., Kogan, S.L.: Beyond predictable workflows: Enhancing productivity in artful business processes. IBM Syst. J. 45(4), 663–682 (2006)
27. Isern, D., Moreno, A.: Computer-based execution of clinical guidelines: a review. Int. J. of Medical Informatics 77(12) (2008)
28. Jarvis, P., Moore, J., Stader, J., Macintosh, A., du Mont, A.C., Chung, P.: Exploiting AI Technologies to Realise Adaptive Workflow Systems. AAAI Workshop on Agent-Based Systems in the Business Context (1999)
29. Kemsley, S.: The Changing Nature of Work: From Structured to Unstructured, from Controlled to Social. In: BPM'11 (2011)
30. Lenz, R., Peleg, M., Reichert, M.: Healthcare Process Support: Achievements, Challenges, Current Research. IJKBO (2012)
31. Lenz, R., Reichert, M.: IT support for healthcare processes - Premises, challenges, perspectives. Data & Know. Eng. 61(1) (2007)
32. Lyng, K.M., Hildebrandt, T.T., Mukkamala, R.R.: From Paper Based Clinical Practice Guidelines to Declarative Workflow Management. In: BPM (2008)
33. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM. pp. 192–199. IEEE (2011)
34. Marrella, A., Mecella, M.: Continuous Planning for Solving Business Process Adaptivity. In: BPMDS'11 (2011)
35. de Man, H.: Case Management: A Review of Modeling Approaches. BPTrends, www.bptrends.com (2009)
36. Mans, R.S., van der Aalst, W.M.P., Russell, N.C., Bakker, P.J.M., Moleman, A.J.: Process-Aware Information System Development for the Healthcare Domain - Consistency, Reliability, and Effectiveness. In: BPM Workshops (2009)
37. Marjanovic, O.: Improving Knowledge-Intensive Health Care Processes beyond Efficiency. In: ICIS'11 (2011)
38. Medeiros, A.K., Weijters, A.J., Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Discov. 14(2), 245–304 (2007)
39. Mulyar, N., van der Aalst, W.M., Peleg, M.: A Pattern-based Analysis of Clinical Computer-interpretable Guideline Modeling Languages. JAMIA 14(6) (2007)
40. Mulyar, N., Pesic, M., Van Der Aalst, W.M.P., Peleg, M.: Declarative and procedural approaches for modelling clinical guidelines: addressing flexibility issues. In: BPM'07 (2007)
41. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541 –580 (1989)
42. Peleg, M.e.a.: Comparing Computer-Interpretable Guideline Models: A Case-Study Approach. JAMIA 10(1) (2003)
43. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: BPM Workshops (2006)
44. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: EDOC. pp. 287–300 (2007)
45. R-Moreno, M.D., Borrajo, D., Cesta, A., Oddi, A.: Integrating planning and scheduling in workflow domains. Expert Syst. with App. 33(2) (2007)
46. Reichert, M.: What BPM technology can do for healthcare process support. In: AIME'11 (2011)
47. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in Process-Aware Information Systems. In: Trans. on Petri Nets and Other Models of Concurrency II (2009)

48. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press (2001)
49. Sonnenberg, F.A., Hagerty, C.G.: Computer-Interpretable Clinical Practice Guidelines. Where are we and where are we going? Yearbook of Medical Inf. 45 (2006)
50. ter Hofstede, A., van der Aalst, W., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2009)
51. Vaculin, R., Hull, R., Heath, T., Cochran, C., Nigam, A., Sukaviriya, P.: Declarative business artifact centric modeling of decision and knowledge intensive business processes. In: EDOC '11 (2011)
52. van der Aalst, W.M.P., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E., Günther, C.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling 9, 87–111 (2010)
53. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
54. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
55. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, E., Weijters, T.: Prom: The process mining toolkit. In: BPM'09 (Demos) (2009)
56. van der Aalst, W.M.P., Nikolov, A.: Mining e-mail messages: Uncovering interaction patterns and processes using e-mail logs. IJIIT 4(3), 27–45 (2008)
57. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM. LNCS, vol. 4184, pp. 1–23. Springer (2006)
58. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Comp. Sc. - R&D 23(2), 99–113 (2009)
59. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. K. D. Eng. 16(9), 1128–1142 (2004)
60. van der Aalst, W.M.P., Weske, M.: Case handling: a new paradigm for business process support. Data & Know. Eng. 53(2) (2005)
61. Wang, D., Peleg, M., Tu, S., Boxwala, A., Greenes, R., Patel, V., Shortliffe, E.: Representation Primitives, Process Models and Patient Data in Computer-Interpretable Clinical Practice Guidelines: A Literature Review of Guideline Representation Models. Int. J. of Medical Informatics 68 (2002)
62. Wang, D., Peleg, M., Tu, S.W., Boxwala, A.A., Ogunyemi, O., Zeng, Q., Greenes, R.A., Patel, V.L., Shortliffe, E.H.: Design and implementation of the GLIF3 guideline execution engine. J. of Biomedical Informatics 37(5) (2004)
63. Warren, P., Kings, N., Thurlow, I., Davies, J., Buerger, T., Simperl, E., Ruiz, C., Gomez-Perez, J.M., Ermolayev, V., Ghani, R., Tilly, M., Bösser, T., Imtiaz, A.: Improving knowledge worker productivity - the Active integrated approach. BT Technology Journal 26(2), 165–176 (2009)
64. Weber, B., Reichert, M., Rinderle-Ma, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-aware Information Systems. Data Knowl. Eng. 66 (2008)
65. Weber, B., Wild, W., Lauer, M., Reichert, M.: Improving Exception Handling by Discovering Change Dependencies in Adaptive Process Management Systems. In: BPI'06 (2006)
66. Weijters, A., van der Aalst, W.: Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering 10, 2003 (2001)
67. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. 15(2), 145–180 (2007)
68. Westergaard, M.: Better algorithms for analyzing and enacting declarative workflow languages using LTL. In: BPM'11 (2011)

# Business Processes Verification with Temporal Answer Set Programming [⋆]

L. Giordano[1], A. Martelli[2], M. Spiotta[1], and D. Theseider Dupré[1]

[1] Dipartimento di Informatica, Università del Piemonte Orientale
[2] Dipartimento di Informatica, Università di Torino

**Abstract.** The paper provides a framework for the specification and verification of business processes, based on a temporal extension of answer set programming (ASP). The framework allows to capture fluent annotations as well as data awareness in a uniform way. It allows for a declarative specification of business process but also for a direct encoding of processes specified in conventional workflow languages. Verification of temporal properties of a business process, including verification of compliance to business rules, can be performed by LTL bounded model checking techniques.

## 1 Introduction

The verification of business process compliance to business rules and regulations has gained a lot of interest in recent years and it has led to the development to a process annotation approach [12, 18, 33, 23], where a business processes is enriched with information relevant for compliance verification, to capture the semantics of atomic tasks execution through preconditions and effects. The treatment of data in business process verification, on the other hand, has attracted growing interest in the last decade, with the definition of *artifact-centric* and *data-centric* process models [27, 5, 9].

In this paper we combine the two perspectives and propose a framework for the specification and verification of business processes which allows to model both annotations and data properties by specifying atomic tasks in a uniform way. The approach is well suited for a declarative specification of the business process, which has been advocated by many authors in the literature [32, 30, 25]. Following [7], the specification of annotation can be done in an action theory by defining the effects and preconditions of atomic tasks. The same approach allows to capture data properties, by modelling data acquisition tasks as actions which nondeterministically assign values to variables (data objects) on given domains, under the restriction that domains are finite.

The use of directional rules for modeling business rules as well as to capture the conditional structure of norms is widely used in the literature [18]. In our approach, besides the specification of action preconditions and direct effects, *causal rules* in an action domain allow to capture dependencies among fluents

⋆ This work has been partially supported by Regione Piemonte, Project ICT4LAW.

(propositions whose truth is affected by actions) and *fluent changes*, as well as dependencies between process data and fluents. Our claim is that both static and dynamic causal laws are useful for the specification of business process annotations and their use allows unintended conclusions to be avoided. Observe that, once the data perspective is included, causal laws can include both conditions on data and annotations. For instance, the rule $age \geq 18 \Rightarrow ofAge$ may establish a link between the business process, whose execution assigns values to the variable $age$, and the compliance rules dealing with persons "of age".

The approach we propose is based on Answer Set Programming (ASP) [11] and, more precisely, on the temporal extension of ASP in [16], combining ASP with the temporal logic DLTL [22], an extension of LTL in which the temporal operators are enriched with program expressions. The action language in [16] allows general DLTL constraints to be included in action domains, which can be profitably used for a declarative specification of the business process advocated in the literature [32, 30, 25]. In addition, the proposed approach also allows for a direct encoding of processes specified in workflow languages, and it can be used in combination with state of the art workflow management systems.

The paper considers several verification tasks including the verification of business process compliance to business rules. Verification is performed through Bounded Model Checking [6] techniques and exploits the approach in [16] for DLTL bounded model checking in ASP, which extends the approach for Bounded LTL Model Checking with Stable Models in [21].

## 2 A Temporal Answer Set Programming language

In this section we recall the temporal ASP language introduced in [16]. The language is based on a temporal extension of Answer Set Programming (ASP) which combines ASP with the temporal logic DLTL [22], an extension of LTL in which temporal operators are enriched with program expressions. In particular, in DLTL the next state modality can be indexed by actions, and the until operator $\mathcal{U}^{\pi}$ can be indexed by a program $\pi$ which, as in PDL, can be any regular expression built from atomic actions using sequence (;), nondeterministic choice (+) and finite iteration ($*$). Satisfiability and validity for DLTL are PSPACE-complete problems [22].

Let $\Sigma = \{a_1, \dots, a_n\}$ be a finite non-empty alphabet of actions. From the until operator, the derived modalities $\langle\pi\rangle$, $[\pi]$, $\bigcirc$ (next), $\mathcal{U}$, $\Diamond$ and $\Box$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^{\pi}\alpha$, $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$, $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, $\Sigma$ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Informally, a formula $[\pi]\alpha$ is true in a world $w$ of a linear temporal model if $\alpha$ holds in all the worlds of the model which are reachable from $w$ through any execution of the program $\pi$. A formula $\langle\pi\rangle\alpha$ is true in a world $w$ of a linear temporal model if there exists a world of the model reachable from $w$ through an execution of the program $\pi$, in which $\alpha$ holds.

A *domain description* $D$ is a pair $(\Pi, \mathcal{C})$, where $\Pi$ is a set of laws describing the effects and executability preconditions of actions (as described below), and $\mathcal{C}$

is a set of *temporal constraints*, i.e., general DLTL formulas. Atomic propositions describing the state of the domain are called *fluents*. Actions may have direct effects, described by action laws, and indirect effects, described by causal laws capturing the causal dependencies among fluents.

Let $\mathcal{L}$ be a first-order language which includes a finite number of constants and variables, but no function symbol. Let $\mathcal{P}$ be the set of predicate symbols, $Var$ the set of variables and $C$ the set of constant symbols. We call *fluents* atomic literals of the form $p(t_1, \ldots, t_n)$, where, for each $i$, $t_i \in Var \cup C$. A *simple fluent literal* $l$ is an atomic literal $p(t_1, \ldots, t_n)$ or its negation $\neg p(t_1, \ldots, t_n)$. We denote by $Lit_S$ the set of all simple fluent literals, and we assume that the fluent $\bot$ representing the inconsistency is included in $Lit_S$. A *temporal fluent literal* has the form $[a]l$ or $\bigcirc l$, where $l \in Lit_S$ and $a$ is an action name (an atomic proposition, possibly containing variables). Given a (simple or temporal) fluent literal $l$, *not* $l$ represents the default negation of $l$. A (simple or temporal) fluent literal possibly preceded by a default negation, will be called an *extended fluent literal*. The laws are formulated as rules of a temporally extended logic programming language having the form

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{1}$$

where the $l_i$'s are simple or temporal fluent literals. As usual in ASP, rules with variables are a shorthand for the set of their ground instances; and we let $\Sigma$ be the set of ground instances of atomic actions in the domain description.

In the following we call a *state* a set of ground fluent literals. A state is said to be *consistent* if it is not the case that both $f$ and $\neg f$ belong to the state, or that $\bot$ belongs to the state. The execution of an action in a state may possibly change the values of fluents in the state through its direct and indirect effects, thus giving rise to a new state. We assume that a law as (1) can be applied in all states while, when prefixed with the **Init**, it only applies to the initial state.

Action laws, causal laws, precondition laws, persistency laws, initial state laws, etc., which are normally used in action theories, can all be defined as instances of (1). *Action laws* describe the effects of atomic tasks. The meaning of an action law $[a]l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$, (where $l_0 \in Lit_S$ and $l_1, \ldots, l_n$ are either simple fluent literals of temporal fluent literals of the form $[a]l$) is that executing action $a$ in a state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold makes the effect $l_0$ to hold (in the state after the action).

*Precondition laws* allow the specification of executability conditions for atomic tasks; they are a special case of action laws with $\bot$ as effect, i.e., they have the form: $[a]\bot \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ meaning that $a$ cannot be executed (has an inconsistent effect) in case $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold.

*Causal laws* define causal dependencies among propositions, which are used to derive indirect effect of actions, called *ramifications* in the literature of reasoning about actions where it is well known that causal dependencies among propositions are not suitably represented by material implication in classical logic. *Static causal laws* have the form: $l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ where the $l_i$'s are fluent literals. Their meaning is: if $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold in a state, then $l_0$ is caused to hold in that state. *Dynamic causal laws* have the form: $\bigcirc l_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n$ where $l_0$ is a fluent literal and the $t_i$'s are either fluent literals or temporal fluent literals of the form $\bigcirc l_i$ (meaning that the fluent literal $l_i$ holds in the next state). Their meaning is: if $t_1, \ldots, t_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold, then $l_0$ is caused to hold in the next state. In particular, in the premise, a combination of the form $\neg f, \bigcirc f$ (or $f, \bigcirc\neg f$) may be used to mean that fluent $f$ *becomes* true (resp., false). The language also includes constraints of the form $\bot \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ where the $l_i$'s are simple or temporal fluent literals.

In this language, default negation in clause bodies allows for the specification of *nondeterministic action laws*, of the form $[a](l_0 \vee \ldots \vee l_k) \leftarrow l_{k+1}, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$, stating that the execution of action $a$ in a state in which $l_{k+1}, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold, makes nondeterministically one of $l_0, \ldots, l_k$ true. In fact, $[a](l_0 \vee \ldots \vee l_k) \leftarrow Body$ can be seen as a shorthand for the rules $[a]l_i \leftarrow Body, not\ [a]l_1, \ldots not\ [a]l_{i-1}, not\ [a]l_{i+1}, \ldots not\ [a]l_k$ $(i = 1, \ldots, k)$.

The laws above can be used to define persistency laws to deal with frame fluents as well as to complete the initial state in all the possible ways compatible with the initial state specification. The semantics of a domain description, is defined by extending the notion of *answer set* [11] to *temporal answer sets*, so to capture the linear structure of temporal models. We refer to [16] for details.

## 3 Declarative specification of business processes: merging annotations with data

A declarative specification of a business process can be given by exploiting the action theory above to define the effects of atomic tasks as well as their executability preconditions. This approach has been followed in different contexts such as in the declarative specification of web services in [26, 5] and in the declarative specification of agent communication protocols in [35, 14]. We show that causal laws have a relevant role in the specification of background knowledge, which is common both to the business process and to the business rules, and that the proposed approach allows for an easy integration of the data perspective.

The declarative specification of business processes has been advocated by many authors [32, 30, 25], as opposed to the more rigid transition based approach. A declarative specification of a process is, generally, more concise than transition based specification as it abstracts away form rigid control-flow details and does not require the order among the actions in the process to be rigidly defined.

The Temporal ASP language in Section 2 is well suited for defining immediate and indirect effects of atomic tasks and their preconditions. Consider, for instance, the business process of an investment firm in [7], where the firm offers financial instruments to an investor. The atomic task *investor identification* has as effect that the investor has been identified, while investor *profiling* has the nondeterministic effect that the investor is recognized as being either *risk_averse* or *risk_seeking*. This can be modeled by the action laws:

$$[investor\_ident(I)]investor\_identified(I)$$
$$[profiling(I)](risk\_averse(I) \vee risk\_seeking(I)) \leftarrow investor\_identified(I)$$

The first action law has empty precondition. The fact that $profiling$ can be executed only when the atomic task *investor identification* has been executed, can be modeled by introducing the precondition law:

$$[profiling(I)]\bot \leftarrow \mathit{not}\ investor\_identified(I))$$

which, literally, states that executing action $profiling$ in a state in which the investor $I$ has not been identified gives an inconsistency. Observe that, in this language, an action is executable unless there is a precondition law for it whose antecedent is not satisfied. Hence, once the investor has been identified, the action $profiling(I)$ becomes executable. However, to guarantee that it will be eventually executed, we can add in $\mathcal{C}$ the DLTL constraint

$$\Box[investor\_ident(I)]\Diamond\langle profiling(I)\rangle\top$$

To force the execution of $profiling$ immediately after *investor identification*, instead, we could add the constraint: $\Box[investor\_ident(I)]\ \langle profiling(I)\rangle\top$.

The presence of DLTL constraints in a domain specification allows for a simple way to constrain activities in a business process. Observe that, as DLTL is an extension of LTL, it is possible to provide an encoding of all ConDec [28] constraints into our action language. The additional expressivity which comes from the presence of program expressions in DLTL, allows for a very compact encoding of certain declarative properties of the domain dealing with finite iterations. For instance, the property "action b must be executed immediately after any even occurrence of action a in a run" can be expressed by the temporal constraint: $\Box[(a;\Sigma^*;a)^*]\langle b\rangle\top)$, where $\Sigma^*$ represents any finite action sequence.

In [7] it has been shown that program expressions can be used to model the control flow of a business process in a rigid way. However, the solution in [7] does not deal with non-structured workflows.

As concerns the data perspective, an atomic task which acquires the value of a data variable (data object) $x$ can be regarded as an action assigning nondeterministically to $x$ one of the values in its domain. Consider, for instance, the atomic task $verify\_status$ which verifies the status of a customer. Assume it has the effect of assigning a value (*gold, silver* or *unknown*) to a variable $status$. The task $verify\_status$ can be regarded as a non deterministic action assigning one of the possible values to the variable status:

$$[verify\_status](\ status(gold) \vee status(silver))$$

In general, we model a data acquisition task as a nondeterministic action. As an example, let us consider an atomic task *get_order* which acquires an order of a product $P$ and an atomic task $select\_shipper(P)$ which selects a shipper among the available shippers, which are compatible with the choice of the product $P$. Let us introduce the notation $1\{[a]R(X) \mid P(X)\}1$ (similar to the notations used in Clingo and in S-models) as a shorthand for the two laws:

$$[a]R(X) \leftarrow\ \mathit{not}\ [a]\neg R(X) \wedge P(X)$$
$$[a]\neg R(X) \leftarrow\ [a]R(Y) \wedge P(X) \wedge P(Y) \wedge X \neq Y$$

meaning that after the execution of action $a$, $R(X)$ holds for a unique value of $X$ among those values satisfying $P(X)$. Let $available\_product(P)$ and $available\_shipper(S)$ be the predicates defining the available products and shippers, and $compatible(P, S)$ be a predicate saying that product $P$ and shipper $S$ are compatible. We can represent the effect of action $get\_order$ by the law:

$1\{[get\_order]product(P) \mid available\_product(P)\}1$

and the effect of action $select\_shipper(P)$ as

$1\{[select\_shipper(P)]shipper(S) \mid available\_shipper(S)\}1$.

The requirement that $P$ and $S$ must be compatible can be enforced introducing the constraint:

$\bot \leftarrow [select\_shipper(P)]shipper(S) \wedge not\ compatible(P, S)$

meaning that it is not the case that the selected shipper $S$ and the product $P$ to be shipped are not compatible.

The above specification of the effects of the task $select\_shipper(P)$ has strong similarities with the specification of a post-condition for a service in [9]. Indeed, in [9], a post-condition of the form $R(\overline{x}) := \psi(\overline{x})$, associated with a service $\sigma$, requires that after the execution of $\sigma$ the argument $\overline{x}$ of $R$ is instantiated with a (unique) tuple $\overline{u}$ such that $\psi(\overline{u})$ holds in the previous state (artifact instance). As a difference with [9], where $\psi(\overline{x})$ is a first-order temporal formula, our temporal language does not allow for explicit quantification: all variables occurring in action and causal laws are intended to be universally quantified in front of the laws. Furthermore, in our approach we cannot deal with infinite domains. As usual in ASP, a finite groundization the set of laws in the domain specification is required. Abstraction techniques as those in [24] can be adopted to abstract infinite or large domains to a finite, small set of abstract values.

## 4 Specification of business rules: causality and commitments

The use of directional implications for modeling business rules as well as for modeling the conditional structure of norms is widely recognized in the literature [18]. In this section we claim that static and dynamic causal laws, proposed in the AI literature about reasoning about actions and change, are also appropriate for modeling business processes.

Consider the domain in examples 2 and 3 in [33], with the rule stating that if an insurance claim is accepted by reviewer A and reviewer B, then it is accepted. Suppose this is represented as the material implication

$$claimAccRevA \wedge claimAccRevB \supset claimAccepted$$

i.e., the clause $\neg claimAccRevA \vee \neg claimAccRevB \vee claimAccepted$. Suppose further, as in [33], that as a result of an action with direct effects, we accept models where such effects hold, that satisfy a background theory including the implication above, and, according to the Possible Models Approach [34], differ minimally from the previous state. Consider a state where $claimAccRevA$ already holds, and an action of acceptance for reviewer B occurs, with direct effect $claimAccRevB$. In order to satisfy the material implication, $claimAccepted$

should become true, or *claimAccRevA* should become false, or both; minimal difference with the previous state only excludes this third alternative, while providing equal status to the first two. If the redundancy in the process means that the assessment of a reviewer has no influence on the other's, then only the first result, where *claimAccepted* becomes true, is intended. The (static) causal rule

$$claimAccepted \leftarrow claimAccRevA, claimAccRevB$$

allows to obtain the first solution, given that its semantics imposes that in all states, if $claimAccRevA \wedge claimAccRevB$ is true (and, in particular, it just became true), then *claimAccepted* holds (and it becomes true as a side effect if the premise just became true).

However, the above implication might not actually be intended, as in case later steps in the process could make the claim not accepted. For example, the process model might specify that if the amount claimed is greater than a threshold, it should go through further approval by a supervisor (with possible effect $\neg claimAccepted$). Unlike [33], we consider the case where this does not mean that $claimAccRevA \wedge claimAccRevB$ should become false, i.e., at least one conjunct (or exactly one, for a minimal change) should become false. Rather, we suggest that here, after reviewers acceptance, *claimAccepted* actually stands for "accepted unless decision is overridden" Dynamic causal laws are suitable to represent this; the side effect of acceptance by the single reviewers becomes:

$$\bigcirc claimAccepted \leftarrow \bigcirc claimAccRevA, \neg claimAccRevB, \bigcirc claimAccRevB$$
$$\bigcirc claimAccepted \leftarrow \neg claimAccRevA, \bigcirc claimAccRevA, \bigcirc claimAccRevB$$

where syntactic sugar can be introduced, as in [8], to succinctly state that the conjunction $claimAccRevA \wedge claimAccRevB$ is *initiated* i.e., it becomes true.

Such rules correctly make *claimAccepted* true after reviewer acceptance, but, if a further step has the effect $\neg claimAccepted$, they do not "fire" because $claimAccRevA \wedge claimAccRevB$ is true, but it is not *becoming* true. Note the difference with the static causal rule which would fire (because $claimAccRevA \wedge claimAccRevB$ is true) and then contradict $\neg claimAccepted$.

A particularly significant case of the pattern above, where a fluent becomes true as an indirect effect of some activity, but may be canceled by further activities, is the one of **obligations**, which arise naturally in compliance rules: several such rules are variants of "if B happens, then A shall happen", or, "if B is (or becomes) true, then A shall become true". Compliance verification for such rules could be performed by verifying a straightforward representation of the rule as a temporal logic formula, e.g., in LTL, the formula $\Box(B \supset \Diamond A)$.

This, however, does not admit the possibility that a later activity cancels the obligation: e.g., if an order for goods is confirmed by the seller, goods have to be shipped; but if the customer cancels the order, the obligation to ship goods is canceled. An explicit representation of obligations is useful to this purpose. In this paper we limit our attention to one type of obligations in the classification in [19]: the case where a given condition should become true at least once, after they have been triggered; i.e., we consider achievement obligations in [19], and we only consider the case where the obligation should be fulfilled after it is triggered.

We then identify obligations with the notion of **commitment** from the social approach to agent communication [30, 20, 10]. A *(base) commitment* $C(i, j, A)$, means that agent $i$ is committed to agent $j$ to bring about $A$, while *conditional commitments* of the form $CC(i, j, B, A)$, mean that agent $i$ is committed to agent $j$ to bring about $A$, if condition $B$ is brought about [35, 14]. In this paper we do not consider agents explicitly, and we concentrate our attention to base commitments $C(A)$ where $A$ is a fluent; $C(A)$ is also a fluent, which can be made true, due to an action law or a dynamic causal law, as a direct or indirect effect of an activity in the process (order confirmation, in the example). The commitment (to ship goods, in the example) can be made false by an action with effect $\neg C(A)$ (the customer cancelling the order). Fulfilling the commitment (shipping goods) also makes the commitment false. Compliance verification, as we shall see in Section 6, amounts then to verifying that commitments, if introduced, are discharged, i.e., they are either fulfilled or explicitly canceled.

We refer to [7] for the treatment of defeasible business rules by means of default negation in ASP.

## 5 Translating business process workflows in ASP

The temporal action language introduced above provides a flexible and declarative specification language for business processes, and in [16] we have provided its translation to standard ASP.

There are, however, cases where the business process is naturally modeled (or it has already been modeled) in a workflow language such as YAWL [31]. In principle, such process models could be translated automatically to the temporal action language, but we have provided a direct translation to ASP for a subset of YAWL including AND- and XOR- splits and joins. The translation is based on an enabling semantics of arcs and tasks: an atomic task can be executed (i.e., the action can occur) when it is enabled. It is enabled when its only incoming arc is enabled, or it is an AND-join and all incoming arcs are enabled, or it is a XOR-join an one incoming arc is enabled. The execution of a task enables the outgoing arcs, and, in case it is a XOR-split, the execution of a subsequent activity based on the enabling of one such arc disables the other arcs.

## 6 Business process verification by bounded model checking

In [16] we have developed Bounded Model Checking techniques for the verification of DLTL constraints. In particular, the approach extends the one developed in [21] for bounded LTL model checking with Stable Models. The approach can be used for checking satisfiability of temporal formulas. To prove the validity of a formula, its negation is checked for satisfiability. In case the formula is not valid, a counterexample is provided.

Several verification tasks can be addressed within the proposed approach. Compliance verification (described in some detail in [7]) amounts to check that all the business rules are satisfied in all the execution of the process. We distinguish among business rules which can be encoded as a temporal formula and business rules whose modeling involves commitments.

As an example of rule which can be encoded as a temporal formula to be verified, consider, in the order-production-delivery process in [24], the rule "Premium customer status shall only be offered after a prior solvency check": it can be verified by checking the validity of the temporal formula

$$\Box(solvency\_check\_done \vee \neg\langle offer\_premium\_status\rangle\top)$$

i.e., by verifying that in all executions of the business process if the action $offer\_premium\_status$ is executable, the fluent $solvency\_check\_done$ holds. As an example of rule modeled through causal laws whose effect is adding a commitment, consider the rule "if the investor signs an order, the firm is obliged to provide him a copy of the contract". It can be encoded by the causal law:

$$C(sent\_contract) \leftarrow order\_signed$$

We require that all the commitments generated are eventually fulfilled, unless they are explicitly cancelled (e.g., in the example, cancelling the order also cancels the obligation to send the contract). Observe that canceling a commitment would not be possible if the commitment to $\alpha$ corresponded directly to the temporal formula $\Diamond\alpha$. A commitment is also discharged when it is fulfilled, i.e., the following causal rule is added for all possible commitments:

$$\bigcirc\neg C(\alpha) \leftarrow C(\alpha) \wedge \bigcirc\alpha$$

Then the verification of rules involving commitments amounts to verifying the validity, for all possible commitments $\mathrm{C}(\alpha)$, of the formula:

$$\Box(\mathrm{C}(\alpha) \rightarrow \Diamond(\neg\mathrm{C}(\alpha)))$$

A verification task considered in [9] is that of verifying properties of a business process, under the assumption that the process satisfies some given business rules. This verification task can also be addressed in our approach: the specification of the business rules is given by adding temporal constraints (and, possibly, causal laws) to the domain specification. The executions of the resulting domain specification are then verified against other temporal properties.

Satisfiability and validity of a DLTL formula over the business process executions are decidable problems. However, given that BMC is not complete in general, an alternative approach to BMC in ASP is proposed in [15] to address the problem of completeness, by exploiting the Büchi automaton construction while searching for a counterexample.

## 7 Conclusions and related work

The paper presents an approach to the verification of the compliance of business processes with norms. The approach is based on a temporal extension of ASP.

The business process, its semantic annotation and the norms are encoded using temporal ASP rules as well as temporal constraints. Causal laws are used for modeling norms, and commitments are introduced for representing obligations. Compliance verification can be performed using the BMC technique developed in [16] for DLTL bounded model checking in ASP, which extends the approach for bounded LTL model checking with Stable Models in [21].

This paper enhances the approach to business processes compliance verification in [7] by taking into consideration the data perspective and by providing a declarative specification of the business process, while in [7] the control flow of a structured business process is modeled in a rigid way by means of a program expression. Also, we have shown that a direct encoding of the process workflow in ASP can be given and exploited for process verification.

Several proposals in the literature introduce annotations on business processes for dealing with compliance verification [12, 18, 33]. In particular, [18] proposes a logical approach to business process compliance based on the idea of annotating the business process. Annotations and normative specifications are provided in the same logical language, namely, the Formal Contract Language (FCL), which combines defeasible logic [3] and deontic logic of violations [17]. Compliance is verified by traversing the graph describing the process and identifying the effects of tasks and the obligations triggered by task execution. Ad hoc algorithms for propagating obligations through the process graph are defined.

The idea of describing the effects of atomic tasks on data through preconditions and effects is already present in [23], where effects and preconditions are sets of atomic formulas, and the background knowledge consists of a theory in clausal form; I-Propagation [33] is exploited for computing annotations. In our approach the domain theory contains directional causal rules rather than general clauses (which allow unintended conclusions to be avoided when reasoning about side effects), and domain annotations are combined with data properties in a uniform approach. In the related paper [33] several verification tasks are defined to verify that the business process control flow interacts correctly with the behaviour of the individual activities.

In [9] a service over an artifact schema is defined as a triple: a precondition, a post-condition and a set of static rules, which define changes on state relations, and are formulas in a first-order temporal logic. State update rules $S(\overline{x}) \leftarrow \phi^+(\overline{x})$ and $\neg S(\overline{x}) \leftarrow \phi^-(\overline{x})$ are essentially specific kind of causal laws whose antecedents $\phi^+$ and $\phi^+$ are evaluated in the artifact instance in which the service is executed and whose consequents are added to the resulting artifact instance. [9] identifies a class of *guarded* artifacts for which verification of properties in a (guarded) first-order extension of LTL is decidable. While our action language does not allow for explicit quantification, it allows for a flexible formulation of action effects and causal laws, which permits (as shown in Section 3) an encoding of post-conditions as in [9].

In [4] compliance checking for BPMN process models is based on the BPMN-Q visual language. Rules are given a declarative representation as BPMN-Q queries, which are translated into temporal formulas for verification.

In [25] the Abductive Logic Programming framework SHIFF [2] is exploited in the declarative specification of business processes as well as in the verification of their properties. In [1] expectations are used for modelling obligations and prohibitions and norms are formalized by abductive integrity constraints.

In [29] Concurrent Transaction Logic (CTR) is used to model and reason about general service choreographies. Service choreographies and contract requirements are represented in CTR. The paper addresses the problem of deciding if there is an execution of the service choreography that complies both with the service policies and the client contract requirements.

Temporal rule patterns for regulatory policies are introduced in [13], where regulatory requirements are formalized as sets of compliance rules in a real-time temporal object logic. The approach is used essentially for event monitoring.

## References

1. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, and G. Sartor. Mapping of Deontic Operators to Abductive Expectations. *NORMAS*, pages 126–136, 2005.
2. Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Trans. Comput. Log.*, 9(4), 2008.
3. G. Antoniou, D. Billington, G. Governatori, and M. J. Maher. Representation results for defeasible logic. *ACM Trans. on Computational Logic*, 2:255–287, 2001.
4. Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic, LNCS 5240. In *BPM*, pages 326–341. Springer, 2008.
5. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *BPM*, pages 288–304, 2007.
6. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
7. D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. L. Pozzato, and D. Theseider Dupré. Verifying business process compliance by reasoning about actions. In *CLIMA XI*, pages 99–116, 2010.
8. M. Denecker, D. Theseider Dupré, and K. Van Belleghem. An inductive definitions approach to ramifications. *Electronic Transactions on Artificial Intelligence*, 2:25–97, 1998.
9. A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.
10. N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *AAMAS03*, pages 520–527.
11. M. Gelfond. Answer Sets. *Handbook of Knowledge Representation, chapter 7, Elsevier*, 2007.
12. A. Ghose and G. Koliadis. Auditing business process compliance. *ICSOC, LNCS 4749*, pages 169–180, 2007.
13. C. Giblin, S. Müller, and B. Pfitzmann. From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. *IBM Reasearch Report*, 2007.
14. L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic*, 5:214–234, 2007.

15. L. Giordano, A. Martelli, and D. Theseider Dupré. Achieving completeness in bounded model checking of action theories in ASP. In *Proc. KR 2012*.
16. L. Giordano, A. Martelli, and D. Theseider Dupré. Reasoning about actions with temporal answer sets. *Theory and Practice of Logic Programming*, 2012.
17. G. Governatori and A. Rotolo. Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. *Australasian Journal of Logic*, 4:193–215, 2006.
18. G. Governatori and S. Sadiq. The journey to business process compliance. *Handbook of Research on BPM, IGI Global*, pages 426–454, 2009.
19. Guido Governatori. Law, logic and business processes. In *Third International Workshop on Requirements Engineering and Law*. IEEE, 2010.
20. F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems, Springer LNAI 2650*, 2003.
21. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.
22. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.
23. J. Hoffmann, I. Weber, and G. Governatori. On compliance checking for clausal constraints in annotated process models. *Information Systems Frontieres*, 2009.
24. D. Knuplesch, L. T. Ly, S. Rinderle-Ma, H. Pfeifer, and P. Dadam. On enabling data-aware compliance checking of business process models. In *Proc. ER 2010, 29th International Conference on Conceptual Modeling*, pages 332–346, 2010.
25. M. Montali, P. Torroni, F. Chesani, P. Mello, M. Alberti, and E. Lamma. Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundam. Inform.*, 102(3-4):325–361, 2010.
26. S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Proc. 11th Int. World Wide Web Conference, WWW2002*, pages 77–88, 2002.
27. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428445, 2003.
28. Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops, LNCS 4103*, pages 169–180. Springer, 2006.
29. D. Roman and M. Kifer. Semantic web service choreography: Contracting and enactment. In *International Semantic Web Conference, LNCS 5318*, pages 550–566, 2008.
30. M. P. Singh. A social semantics for Agent Communication Languages. *Issues in Agent Communication, LNCS(LNAI) 1916*, pages 31–45, 2000.
31. A. M. ter Hofstede, W. M. P. van der Aalst, M. Adamns, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment.* 2010.
32. Wil M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*, 2006.
33. I. Weber, J. Hoffmann, and J. Mendling. Beyond soundness: On the verification of semantic business process models. *Distributed and Parallel Databases (DAPD)*, 2010.
34. M. Winslett. Reasoning about action using a possible models approach. In *Proc. AAAI 88, 7th National Conference on Artificial Intelligence*, pages 89–93, 1988.
35. P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. *AAMAS'02*, pages 527–534, 2002.

# A Knowledge-based Approach to the Configuration of Business Process Model Abstractions

Shamila Mafazi[1], Wolfgang Mayer[2], Georg Grossmann[2], and Markus Stumptner[2]

University of South Australia, Adelaide, SA, 5095, Australia
[1]shamila.mafazi@mymail.unisa.edu.au
[2] {firstname.lastname}@unisa.edu.au

**Abstract.** Methods for abstraction have been proposed to ease comprehension, monitoring, and validation of large processes and their running instances. To date, abstraction mechanisms have focused predominantly on structural aggregation, projection, and ad-hoc transformations.
We propose an approach for configuration of process abstractions tailored to a specific abstraction goal expressed as constraints on the abstraction relation and process transformation operators. Our framework goes beyond simple structural aggregation and leverages domain-specific properties, taxonomies, meronymy, and flow criteria. In this paper we outline the constraint-based framework and its underlying inference procedure. We show that our approach can handle most of the common process analysis use cases.

**Keywords:** business process abstraction, business process management, process configuration

## 1 Introduction

Models of business processes and operational procedures are increasingly being used in modern organizations, and the size and complexity of processes and their models can often be large. Development processes in large technology-focused organizations can easily span more than one thousand process steps [10]. As a result, process models have become difficult to understand and manage, as they may not be specified in full in order to enable flexible executions. However, such flexibility comes at a price: it is no longer easily possible to reason about executions based on a single process model. Although learning methods have been developed to reconstruct process models from execution logs [5], the resulting processes are often very specific and can be difficult to comprehend in full. Therefore, methods for business process abstraction are desired that enable process analysts to tailor large models to their specific analysis task at hand.

Methods for abstraction have been proposed to ease comprehension, monitoring, and validation of large processes and their running instances. To date, abstraction mechanisms have focused predominantly on structural aggregation and projection. Collapsing "similar" entities in a process model into one abstract element and projecting away irrelevant entities are among the most common forms of simplification employed for abstraction. Similarity and relevancy of process entities is often defined ad-hoc using process structure, clustering techniques, and user-specified selection criteria [4]. Clustering techniques, statistical methods, and ad-hoc criteria are commonly used to devise a concise summary representation that reflects certain aspects of the larger process.

Although structural aggregation can lead to considerable simplification of large process models, the resulting model may not show all required elements or aggregate elements together that would be better kept separate. However, these measures fail to take into consideration the purpose of the abstraction for the user.

We propose an approach to computing abstractions of business process models tailored to conducting selected common business process analysis tasks. We address this problem by imposing constraints on the abstraction relations that relate concrete and abstract process models such that the abstract process model induced by the abstraction relation is guaranteed to include the information needed to assess selected properties of the process. Rather than relying on cumbersome explicit specification of relevant process elements, we combine a questionnaire-driven approach to eliciting constraints for common analysis tasks with explicit specification of additional constraints a user may have.

As a result, significance and granularity of an abstract model can be explicitly controlled and adjusted to suit a given task. Furthermore, the granularity need not be uniform across the entire model; different abstraction operators can be applied to different regions of the process model.

Although techniques for parameterizing the granularity of the resulting abstractions have been introduced in order to compensate for current techniques' inability to devise representations that are fit for the user's objective [8], to the best of our knowledge, no explicit means to control abstractions is available to non-experts in formal process analysis.

Our method can be seen as configuration of process models, where configuration applies to the abstraction operators used in devising process rather than the process model itself. In contrast to classic configuration where one chooses between alternative instantiations of given variation points within a parametric process model, our approach takes a detailed process model without explicit variation points and derives simplified variations thereof. Hence, our configuration method controls the operators applied within the abstraction process rather than the underlying process model.

In this paper we make the following contributions:

- a knowledge-based framework for configuring purposeful abstractions;
- a framework for specifying constraints on the abstraction;
- a method to infer the process elements (nodes, data, labels) that need to be retained in a conforming abstraction;
- a method to compute abstractions conforming to the abstraction goal.

The subsequent sections are structured as follows. Our process model and abstraction framework are introduced in section 2, our constraint-based abstraction framework and configuration mechanism are described in sections 3 and 4, respectively. Abstraction operators are modeled in section 5 and our method of synthesizing conforming abstractions is summarized in section 6, followed by discussion of related work in section 7.

## 2 Process Model Abstractions

Different users of a process model are usually interested in observing a process model at different levels of details. This requires creation of different abstract process models from one model. However, not all abstract views of a process are equally desirable, as useful abstractions should be tailored to the user's needs. In this work, we pursue this aspect of process abstraction by constraining abstractions such that certain user-selected properties of the underlying concrete process are maintained in its abstract view.

We adapt the process model of Smirnov et al.[15] for our purposes and furnish the model with explicit representations of data- and domain-specific properties attached to tasks:

**Definition 1 (Process Model).** *A tuple* $(N, F, P, \phi, D_P)$ *is a process model, where* $N$ *is a finite set of nodes partitioned into tasks* $N_t$ *and gateways* $N_g$*,* $F \subseteq N \times N$ *is the flow relation such that* $(N, F)$ *is a connected graph,* $P$ *is a finite set of properties of tasks,* $D_P$ *is a finite set of property values of tasks, and* $\phi : N \times P \mapsto D_P$ *is a function that maps each property of a node to its value. For brevity, we write* $n.p$ *for* $\phi(n, p)$*. Let* $M$ *denote the set of all process models.*

The set of properties $P$ comprises common domain-specific properties, predicate valuations, and information derived from executions of process instances. Common properties include roles, resources, timing information, and used and modified data flow information. Domain-specific predicates are boolean properties expressing facts such as "is on a critical path". Information derived from executions indicate aggregate information, for example execution frequencies or number of running instances of a task.

Given a concrete model $m$ of a business process, an *abstract view* of $m$ is a process model $\hat{m}$ that retains "significant" entities of $m$ and omits insignificant ones. In our framework, *entities* comprise the nodes, flows, and properties associated with nodes in a given model. We write $\Omega_m$ to denote the set of entities in $m$ where $\Omega_m \subseteq N \cup F \cup \{n.p | n \in N, p \in P\}$.

Which entities are considered significant is largely determined by the purpose of the abstract model and hence should be defined flexibly based on the goals of the analyst. We will therefore use an abstract predicate $sign \subseteq \Omega_m \cup \Omega_{\hat{m}}$ to capture the significant entities.

Whereas insignificant entities can be either *eliminated* from in the abstraction or *absorbed* into an abstract entity, the significant elements are to be retained. The correspondence between significant entities of $m$ and their abstract counterpart in $\hat{m}$ is given by an *abstraction relation* $R \subseteq \Omega_m \times \Omega_{\hat{m}}$.

**Definition 2 (Process model abstraction).** *A business process model abstraction is a function* $\alpha : M \mapsto M$ *that transforms a model* $m$ *into a model* $\hat{m} = \alpha(m)$ *with correspondence relation* $R_\alpha$ *such that*

- $\forall \hat{\omega} \in \Omega_{\hat{m}}\ sign(\hat{\omega})$ *is true,*
- $\forall \hat{\omega} \in \Omega_{\hat{m}} \exists \omega \in \Omega_m\ (\omega, \hat{\omega}) \in R_\alpha$*,*
- $\forall \omega \in \Omega_m\ sign(\omega) \rightarrow \exists \hat{\omega} \in \Omega_{\hat{m}} : (\omega, \hat{\omega}) \in R_\alpha$*, and*
- $\alpha$ *preserves local composition of* $m$ *in* $\hat{m}$*.*

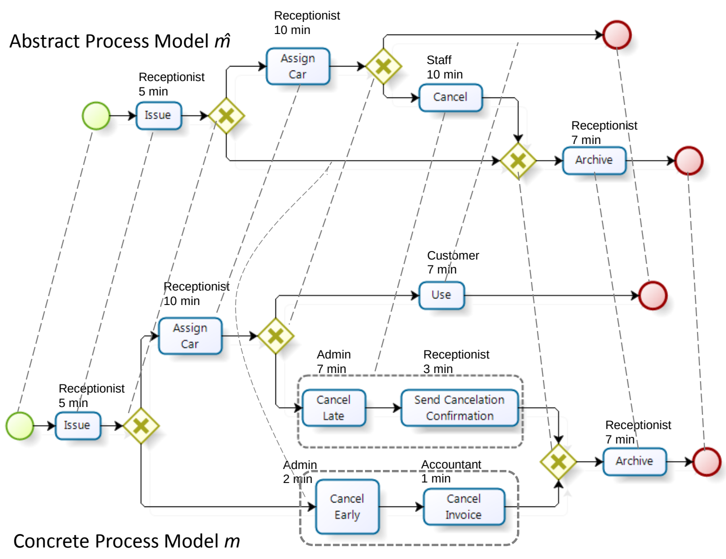

**Fig. 1.** Example Process Model (bottom), Abstract Model (top), and Correspondence Relation

The first three conditions ensure that all retained entities in the abstraction are significant, are justified by the existence of at least one entity in the concrete process, and that all significant concrete entities have a corresponding element in the abstract model. The fourth condition restricts correspondences to meaningful maps that preserve the local structural composition of $m$ in $\hat{m}$. We require that each concrete entity maps to at most one abstract counterpart. Where each abstract property attaches to the abstraction of the concrete nodes belonging to the concrete properties. Also the abstract flow relation reflects the flow in the concrete process model:

- $\forall \omega \in \Omega_m \, \forall \hat{\omega}, \hat{\omega}' \in \Omega_{\hat{m}} \, (\omega, \hat{\omega}) \in R_\alpha \wedge (\omega, \hat{\omega}') \in R_\alpha \rightarrow \hat{\omega} = \hat{\omega}'$,
- $\forall n.p \in \Omega_m \forall \hat{n}.\hat{p} \in \Omega_{\hat{m}} \, (n.p, \hat{n}.\hat{p}) \in R_\alpha \rightarrow (n, \hat{n}) \in R_\alpha$,
- $(\hat{m}, \hat{n}) \in \hat{F} \rightarrow \exists m, n \in N \, (m, n) \in F^* \wedge (m, \hat{m}) \in R_\alpha \wedge (n, \hat{n}) \in R_\alpha$.

Consider the example process models in Figure 1, where the model in the lower half depicts the concrete process and the upper half shows the abstract model. The correspondence relation for tasks is indicated by dashed lines; the correspondences for flows are left implicit. Assuming that all elements performed by role *Receptionist* in $m$ are significant, the abstraction satisfies the condition of Definition 2 as well as the three constraints stated above. For illustration, assume that tasks *Cancel Late* and *Send Cancellation Confirmation* each have a property *Duration*, then the constraints on $R_\alpha$ ensure that property *Duration* of abstract task *Cancel* is an abstraction of only the concrete tasks' property.

## 3 Abstraction Specification

According to Smirnov et al.[15] business process abstraction consists of three aspects: the *why*, the *when* and the *how* aspect. The *why* aspect captures the reasons for building an abstraction of a process model (fragment), the *when* aspect describes the conditions under which an element of a process model needs to be abstracted, and the *how* aspect relates to the concrete transformation mechanism to devise an abstraction. Whereas an extensive body of work covers the *how* aspect, comparatively little work is available to address the remaining aspects.

Our work aims to address the *why* and *when* aspects. We assume that a specification of the information, its granularity, and predicates whose truth values shall be preserved by the abstraction can be elicited, represented formally, and exploited to guide a search procedure to infer suitable abstractions. Let $\Gamma$ be such a specification, formulated over the entities in a given process model $m$. Specifically, we are interested in abstract models $\hat{m} = \alpha(m)$ satisfying $\Gamma$.

By making the abstraction criterion explicit, the *why* aspect of process abstraction is captured, which can be translated into conditions for *when* it is admissible to abstract different entities. We define the significance predicate such that the entities are preserved which are required to ensure that criterion $\Gamma$ is fulfilled on the abstract model. Building on prevalent structural rewriting mechanisms, we provide generic operators on properties and their values in order to automatically eliminate or aggregate entities, and furnish the abstract model with a suitable representation of aggregated information. The application of operators is restricted such that the resulting abstract model retains the significant entities and predicates.

An abstraction criterion may be composed of the following specification primitives:

- $sign(\omega)$ for $\omega \in \Omega_m$;
- $\omega = \omega'$ for $\omega, \omega' \in \Omega_m \cup \Omega_{\hat{m}}$;
- $(n, n') \in F^* \cup \hat{F}^*$, $n, n' \in N_t \cup \hat{N}_t$, $n, n' \in N_g \cup \hat{N}_g$;
- $n.p \oplus c$, where $n$, $p$, and $c$ are a node, a property and a constant drawn from $D_P$, respectively, and $\oplus$ is a relational operator (e.g. $\prec$, $\preceq$, $=$, $\neq$, $\ldots$);
- $(\omega, \hat{\omega}) \in R_\alpha$;
- negation, conjunction, disjunction, universal and existential quantification.

This language is expressive enough to capture many interesting properties, including domain-specific predicates and some aggregate instance information. The starred $F^*$ and $\hat{F}^*$ denote the transitive closure of the flow relation.

For example, one could be interested only in the expensive tasks in the process model in Figure 1, where the value of *Fee* exceeds some threshold \$\$: $\Gamma = x.Fee \geq \$\$ \rightarrow \exists x\, (x, \hat{x}) \in R_\alpha \wedge x.p = \hat{x}.p \wedge (x.p, \hat{x}.p) \in R_\alpha$ for $p \in \{Fee, Label\}$. Capturing this explicitly in $\Gamma$, significance predicate and aggregation operators can be found. The example formula implies that all "expensive" tasks will retain their precise labels and fee information, whereas all other tasks and properties can potentially be abstracted away (subject to maintaining the generic abstraction constraints and well-formedness of the resulting abstract process).

While this example may seem trivial, our approach generalizes to more involved situations. For example, if execution times shall be retained, but labels of some tasks need

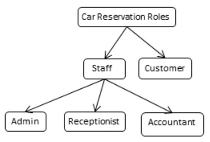

**Fig. 2.** Role Hierarchy

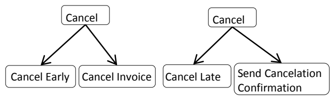

**Fig. 3.** Task Meronymy

not be, our approach allows us to absorb otherwise insignificant tasks into other tasks, but prevents us from eliminating the task entirely, which would result in its contribution to execution time being lost. Similarly, the model abstractions that may be applied in devising an abstraction would be restricted to aggregate the property of sequence of nodes using the *sum* function but not, for example, *max* function. Furthermore, data flow in the model may impose restrictions on significance of non-local process entities.

To facilitate the abstraction of data properties and other non-structural aspects of the business process, we assume that the value domain $D_p$ of each property (including the label of nodes) $p \in P$ forms a (finite-height) (semi-)lattice with partial order $\prec_p$, where $x \prec_p y$ denotes that $x$ is more precise (or has more information) than $y$. We use $\top_p$ to denote the least element of $D_p$, which provides no information. In this case, the property can be omitted.

For example, let us revisit the model in Figure 1. An example of the (semi-)lattice for the *Role* properties is shown in Figure 2. The lattice for roles indicates that roles *Receptionist* and *Admin* are specializations of role *Staff* and are therefore candidates for role abstraction.

For example, one could be interested in distinguishing *Customers* from *Staff* but not the precise staff roles. This could be captured in $\Gamma$ as a constraint on the *Role* property of nodes. As a result, any value $r$ of property *Role* that satisfies $r \prec Staff$ would be abstracted to value *Staff*.

We impose one more constraint on $R_\alpha$: any admissible $R_\alpha$ must satisfy that no information can be gained in the abstract model. That is, $(\omega, \hat{\omega}) \in R_\alpha \rightarrow \omega \preceq \hat{\omega}$ must hold for all property entities $\omega, \hat{\omega}$.

## 4 Abstraction Configuration

Although the method of constraining valid abstractions is powerful, direct exposure of the formal framework to business analysts is rarely feasible in practice. Therefore, we employ knowledge-based configuration mechanisms to elicit appropriate partial abstraction specifications. We use a variant of the questionnaire method of process configuration [6], which interacts with the user in terms of simple domain-specific questions in order to construct the formal domain representation from the user's answers. Different from previous work, our configuration model does not rely on established variation points within the process model, but rather aims to construct a formula that constrains the admissible abstraction relations and operators that can be used to construct it. No explicit library of processes and variation points specific to the process under consideration is needed.

We envision our process abstraction configurator to provide a wizard-like interaction where process analysts may select the information and predicates they wish to retain in the abstraction, and define domain-specific value lattices, aggregation- and structural transformation operators. Underlying our configurator is a catalog of abstraction constraint templates, which can be selected and its parameters instantiated by the user.

**Definition 3 (Configuration Model).** *A configuration model is a triple $(\mathcal{C}, \mathcal{O}, G)$, where $\mathcal{C}$ is a catalog of abstraction aspects, $\mathcal{O}$ is a library of abstraction operators (defined in section 5), and $G$ is a finite set of boolean propositions.*

The catalog contains configuration options and associated abstraction constraints, the library of abstraction operators defines the transformations that can potentially be applied to the process model, and the set of propositions allows one to restrict the set of applicable operators based on choices made for aspects in the catalog. We first describe the catalog and defer discussion of the operators until the next section.

**Definition 4 (Abstraction Aspect Catalog).** *An abstraction aspect catalog is a set of templates $(Q, X, C[X, G])$ where $Q$ is a configuration option, $X$ is a set of parameter variables, and $C[X, G]$ is a formula template parametric in $X$ specifying the abstraction constraints associated with $Q$ in terms of the process model, and abstraction operator constraint in terms of assignments to $G$. Each placeholder variable $x \in X$ can be assigned a predicate or domain value from the process model (subject to resulting in a well-formed formula $C[X, G]$). The configuration criterion $\Gamma$ is simply the conjunction of all constraints $C_i[x_i, G_i]$ of selected $Q_i$ with binding $X_i = x_i$.*

As an example, let the configuration option $Q_1$ be 'Get a process view from all the interactions between two specific roles'. By selecting this configuration option, the parameter variables are set as: $X = \{Role_1, Role_2\}$. The values for the roles are requested and assigned as $Role_1 = Admin$ and $Role_2 = Accountant$. The configuration imposes constraints on the abstraction relation: a task $n$ must be retained in the abstraction if its $Role$ property valuation matches either $Role_1$ or $Role_2$, and there is a flow from $n$ to another task $n'$ that has property $Role$ set to the remaining given role. Formally, the abstraction criterion $\Gamma$ can be expressed as

$$\begin{aligned}\forall n_1, n_2 \in N_t : (n_1, n_2) \in F^* \\ &\wedge ((n_1.Role = Role_1 \wedge n_2.Role = Role_2) \\ &\quad \vee (n_1.Role = Role_1 \wedge n_2.Role = Role_2)) \\ &\qquad \rightarrow (n_1, n_1) \in R_\alpha \wedge (n_2, n_2) \in R_\alpha.\end{aligned}$$

The catalog allows for convenient elicitation of user's requirements based on common abstraction goal patterns. Table 1 shows how 11 of the 14 common use cases for process abstraction presented by Smirnov et al[15] can be expressed in our framework. Most constraints restrict which tasks and properties may be abstracted, and whether insignificant tasks shall be eliminated or aggregated. In the first group of uses cases (1–4), a process view respecting one or more properties of a task, such as *resources and data objects*, is required. For this purpose the properties of all tasks are compared with the user specified property P. Tasks satisfying property P over property A are retained in the abstraction, whereas others are eliminated. In the second use case, tracing a task, the effect of a task in the process model needs to be assessed. For this purpose a process view containing the tasks which are reachable from the interesting task is produced. The constraint ensures that all tasks $x'$ reachable from a given task $x$ are retained in the abstraction. For instance-related use cases (5–7), we currently require a pre-processing stage, where the tasks in the process model are furnished with aggregate property information derived from the instances. For example, an property representing execution frequencies or cumulative case costs could be added. For use case 9, adapt process model for an external partner, the tasks which need to be presented to the external partner are selected. The selected tasks are considered as significant, hence they need to be retained while the rest of the tasks are aggregated. The first constraint ensures that selected tasks are retained in the abstraction, whereas the second constraint ensures that no insignificant tasks are eliminated from the model (although such tasks may be aggregated with other insignificant tasks). In use case 10, a process view respecting the data dependencies of the tasks is required. For this purpose those tasks which make use of the data objects of interest are considered as significant and must be retained in the abstraction while the rest of the tasks are considered as insignificant and can be eliminated from the abstract model. For use case 13, a process view respecting user specified property(s) is required. Different from use cases 1–4, in this process view the insignificant tasks (tasks without interesting property(s)) are aggregated and presented as a composite task in the process view. Hence the constraint prohibiting the elimination of insignificant tasks must be imposed in addition to the constraint capturing use cases 1–4.

Three use cases cannot directly be expressed in our framework. In use case 14, *Retrieve coarse grained activities*, a view over the coarse-grained tasks are required but not a view over the process model. This requires inferring the coarse-grained activities, i.e, abstraction hierarchies and meronymy, from the detailed process model. In contrast, our approach relies on given abstraction hierarchies and meronymy to compute abstractions. In use case 12, the user needs to control the abstraction level gradually while in our approach the process model is abstracted until all the user specified criteria are met. Finally, use case 8 requires to infer possible executions of the process model given a specification of a case instance. Extensions to our framework would be required in order to infer transitions that are potentially enabled or blocked based on guard conditions and values in the given case instance.

## 5 Abstraction Operators

Once abstraction constraints have been set, the concrete process model $m$ can be transformed into a customized process view $\hat{m}$. In our framework, this amounts to constructing an abstraction function $\alpha$ and its induced $R_\alpha$ such that all abstraction constraints are satisfied when applying $\alpha$ on $m$. We employ generic search techniques to compose $\alpha$ from individual model transformation operators selected from a library of abstraction operators.

| |
|---|
| Preserving Relevant Tasks (Use cases 1–4) |
| $Q_1$ : Retain a task if property [A] satisfies [P]<br>$C_1[A,P] = \forall x \in N_t\, [P](x.[A]) \rightarrow (x,x) \in R_\alpha$ |
| Tracing a Task (Use case 11) |
| $Q_2$ : Retain a task if it is reachable from the node [x]<br>$C_2[x] = \forall x' \in N\, (x,x') \in F^* \rightarrow (x',x') \in R_\alpha$ |
| Preserving Relevant Process Instances (Use cases 5–7) |
| $Q_1$ and $Q_2$, based on pre-processed model |
| Adapt Process Model for an External Partner (Use case 9) |
| $Q_3$ : Retain selected tasks in set $T$<br>$\forall x \in T\, (x,x) \in R_\alpha$<br>$Q'_3$ : Aggregate insignificant tasks:<br>$\forall x \in N\, sign(x)$ |
| Trace Data Dependencies (Use case 10) |
| $Q_4$ : Retain a task if it uses data property [P]<br>$C_4[P] = \forall x \in N_t \forall p \in [P]\, HasProperty(x,p) \rightarrow (x,x) \in R_\alpha \wedge (x.p, x.p) \in R_\alpha$ |
| Get Process Quick View Respecting a Property (Use case 13) |
| $Q_1$ and $Q'_3$ |

**Table 1.** Representation of Use Cases in [15]

Abstraction operators are model transformations that rewrite the concrete model's entities into their abstract counterparts. Traditionally, work on business process abstraction focuses predominantly on structural transformations, where rules specify how fragments in a model shall be transformed into an abstract (smaller) fragments in the abstract model. Our work extends this approach to data properties.

Similar to constraints on the abstraction relation, which limit the information retained in the abstraction, the selection of abstraction operators is subject to constraints imposed by the configuration model that ensure abstract data values are given meaningful values consistent with the purpose of the abstraction.

**Definition 5 (Abstraction Operator).** *An abstraction operator is a tuple $(R, S, V, W)$ where $R$, $S$ are fragments of a process model ("patterns") with common variables $V$, and $W$ is a boolean expression over propositions $G$ (in the configuration model) and $V$, governing the application of the operator. If $R$ matches with binding $\sigma$ in a model $m$, and $W$ is satisfiable, a model $m' = m[R\sigma \mapsto S\sigma]$ is obtained by replacing the matched part $R\sigma$ in $m$ with the replacement fragment $S\sigma$. Substitute $S$ may contain data transformation functions that compute the aggregate value for properties in the abstract model. Operators include sum, min, max, avg, for numeric properties, and least upper bound and greatest lower bound operators (if defined) on properties' value lattices.*

Our library of abstraction operators currently comprises:

- Projection operators that eliminate tasks/flows;
- Entity abstraction rules that transform labels and properties of individual tasks. These operators abstract property values according to the corresponding lattices of domain values;

– Structural rewrite rules that transform the process structure and re-arrange tasks and flows;
– Aggregation rules that aggregate values of properties of multiple tasks. Separate rules exist for properties of different type, and different aggregation functions may need to be used for sequence, choice, parallel, and loop constructs.

For space reasons we cannot present the entire collection in detail. Figure 4 contains examples of property-related aggregation for properties of different types (numeric, set-valued, boolean). The bottom part shows the concrete fragments and the top part the abstract counterparts; $X$ and $Y$ represent variables to be matched and $a$,$b$,$c$ represent placeholders for numeric, set-valued, and boolean properties, respectively.

Figure 4a indicates 2 tasks in a block. To aggregate the numeric properties of the two tasks, the operators such as *Max, Min, Avg, Sum* can be employed. Selecting an operator is completely case based. For example, assume a user is interested in tasks with high hand-off times. In this case, the operator *Max* needs to be selected to assign the maximum hand-off time to the composite task *XY*. Likewise, for the set-valued properties, an operator such as *union, aggregate meronymy, abstract label*, based on the configuration option in hand, can be selected. The operators for boolean properties of the tasks, include, *Or, And, Xor*. As an example, assume a user is interested in observing the tasks which are in a critical path, the operator *Or* can be employed which indicates the composite task is whether on a critical path or not. Figure 4b shows an abstraction operator for two tasks in a loop. For the numeric properties of these tasks, based on how many times the loop is executed, the result from the abstracting operator needs to be multiplied or widened to infinity, if an upper bound is not known. Figure 4c shows an abstraction operator for sequential tasks. In this case, where numeric properties typically are aggregated, set-valued properties are merged, and boolean properties are either merged or combined using logic operators to infer the property value associated with the abstract task.

Table 2 gives a list of operators currently defined in our library. The table on the right-hand side of the figure shows examples for formalization of three operators in our framework. Our formalization relies on a set $G$ of propositions defined in the configuration model that is used to govern the application of certain abstraction operators. The elements of this set are determined by the selected configuration options and domain model and consists of propositions of the form $Enable(o, op, p)$, where $o$ is the name of an abstraction operator, $op$ is an aggregation operation, and $p$ is a property. Together with a hierarchy of properties (with specialization ordering $\sqsubseteq$), the propositions are used to control which operators can be applied to certain operations. For example, abstraction operator *SumNumPropSeq* is only applicable if none of the configuration options prohibits its application. Whereas most operators are generic and can be applied to process models from any domain, domain-specific operators can be introduced to account for specific abstractions, such as the meronymy approach presented in [14].

## 6 Abstraction Computation

Conceptually, our abstraction method proceeds as follows. Starting with a given concrete process model $m$ and configuration constraints $\Gamma$, we employ a search procedure

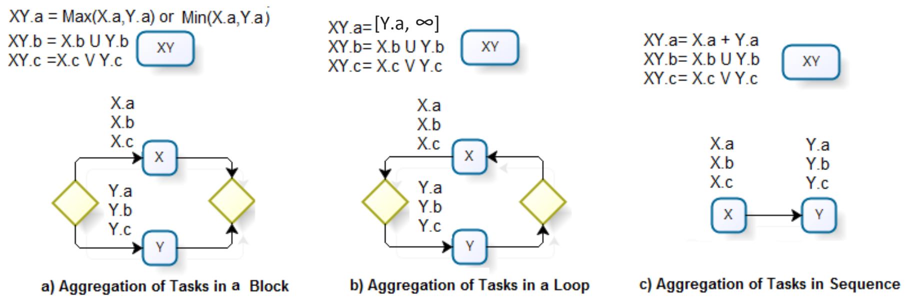

**Fig. 4.** Structural and Property Aggregation Operators

| **Operator** | **Type** |
|---|---|
| Remove Task/Flow | Projection |
| Remove Property | Projection |
| Abstract Label | Entity |
| Abstract Property Value | Entity |
| Aggregate Sequence | Structural |
| Aggregate Concurrent | Structural |
| Aggregate Choice | Structural |
| Aggregate Loop | Structural |
| Aggregate Meronymy | Structural |
| Simplify Gateway | Structural |
| Shift Gateway | Structural |
| Aggregate Value (Seq) | Aggregation |
| Aggregate Value (Concurrent) | Aggregation |
| Aggregate Value (Choice) | Aggregation |
| Aggregate Value (Loop) | Aggregation |

**Logic Representation of the Operators**

**RemoveTask(x):**
$\forall x \in N_t : \neg sign(x) \rightarrow$
$\nexists \hat{x} \in \hat{N}_t : (x, \hat{x}) \in R_\alpha$

**AggregateTaskSeq(x,y):**
$x, y \in N_t \wedge (x, y) \in F \rightarrow$
$\exists \widehat{xy} \in \hat{N}_t : (y, \widehat{xy}) \in R_\alpha$
$\wedge (y, \widehat{xy}) \in R_\alpha$

**SumNumPropSeq(x,y,p):**
$x, y \in N_t \wedge (x, y) \in F \wedge$
$(x, \widehat{xy}) \in R_\alpha \wedge (y, \widehat{xy}) \in R_\alpha$
$\wedge p \sqsubseteq Numeric$
$\wedge \neg Enable(SumNumPropSeq, +, p) \notin G$
$\rightarrow \widehat{xy}.p = x.p + y.p$

**Table 2.** Abstraction Operators

to incrementally build an abstraction. An applicable abstraction operator $r$ is selected and applied to $m$, yielding a transformed model $m'$. If structural aggregation was performed, additional rules to determine the property values of new task(s) are applied. Concurrently, the abstraction function and its correspondence relation are extended to account for the effects of $r$. This process repeats until an abstraction satisfying all constraints in $\Gamma$ has been created and no further rule applications are possible. As a result, we obtain an abstraction function that transforms the given model $m$ in a maximally abstract process model reflecting the relevant tasks and properties. If the intermediate results are recorded, this yields a hierarchy of abstractions of varying granularity. Although not all models in this hierarchy necessarily satisfy all abstraction constraints, navigating the abstraction hierarchy could be useful to "drill-down" in specific areas if needed (comparable to the approach in [12]). Incremental specification and adjustment of abstraction constraints based on initial abstract views remains a direction for future research.

If multiple operators are applicable, this approach may result in multiple possible abstractions. To steer our algorithm towards desirable abstractions, we employ a simple optimization method that aims to minimize both constraint violations and model complexity. When selecting an abstraction operator, we choose the operator that minimizes the sum $viol(\Gamma, \alpha, m) + size(\alpha(m))$, where $viol(\Gamma, \alpha, m)$ denotes the number of constraints in $\Gamma$ that are violated by the current abstraction $\alpha$ when applied to $m$, and $size(\alpha(m))$ measures the number of elements ($|N| + |F|$) in the abstract model $\alpha(m)$. In addition, we maintain a worklist of the current best $k$ abstraction functions. Currently, $k$ is a user-set parameter.

For example, let us revisit the process in Figure 1. Assume that only tasks that are involving role "Receptionist" with *Duration*$>$ 3min are required to be shown in the abstraction.

Based on the given the abstraction constraints, the abstraction criterion $\Gamma$ can be expressed as:
$\forall n \in N_t \wedge n.Role = Receptionist \wedge n.Duration > 3 \rightarrow (n, n) \in R_\alpha$

Considering the criteria, tasks *Use*, *Cancel Early* and *Cancel Invoice* are insignificant, as for example $Use.Role \neq Receptionist$. Aggregating the two tasks *Cancel Early* and *Cancel Invoice* does not result in a significant task either. Hence among others, operator "Remove Task" can be applied to these tasks to eliminate them from the process model. Tasks *Cancel Late* and *Send Cancellation Confirmation* are also insignificant but unlike *Cancel Early* and *Cancel Invoice*, aggregating these two tasks results in a significant task. Hence, operator *Abstract Property Value* can be applied to their role properties to lift the property value to the abstract value *Staff*. Now, operator *Aggregate Meronymy* can be applied (based on the meronymy in Figure 3), combining *Cancel Late* and *Send Cancellation Confirmation* into *Cancel*. The operator *SumNumPropSeq* is applied on the duration properties of the two tasks to add up these properties. Since the abstract task was formed by sequential composition, *Aggregate Value (Seq)* must be applied twice to infer the value for properties *Role* and *Duration* of the abstract task.

At this point, no operators are applicable that satisfy the abstraction constraints. Further simplification of properties and removal of tasks or flows would yield either an ill-formed process model or violate an abstraction constraint.

## 7 Related Work

The research presented in this paper complements the areas of business process model abstraction and process model configuration. Due to emerging various needs, several approaches have been proposed by which the size of a process model can be reduced. However, no single approach provides the same level of configuration ability as ours.

Many approaches for simplifying a given process model based on rewrite rules have been developed [15]. Rewriting approaches based on process fragments, process regions and patterns aim to simplify the structure of large processes by hierarchical aggregation. Various process visualization techniques rely on users selecting interesting tasks and eliminating the remaining tasks from the process model [2]. Pankratius et al. [11] proposed Petri Net based reduction patterns, including place and transition elimination and place and transition join, for abstraction. Liu et al. [9] cluster tasks in a process model, preserving ordering constraints, roles, execution frequencies, and process view for external partners. Since their main abstraction operation is aggregation, the clusters are aggregated into composite nodes. In both of these approaches [11, 9], the authors address the *how* component of the business process abstraction. Since the papers ignore the execution semantic of the process model and treat only tasks, but not the reachability criterion, as the abstraction objects, the process views related to the process instances (use cases [5-7]) cannot be captured by their techniques. Additionally, compared to our approach, their approach is not user interactive. Cardoso et al.[3] proposed reduction rules to synthesize process views respecting ordering constraints and roles. The paper concentrates on the *how* component of the process abstraction while only non-functional property values have been considered. Furthermore, their reduction technique is pattern based. Once a region matches one of their predefined patterns, the region is aggregated into a composite node. Hence, it is not always possible to aggregate an insignificant task, as forming a region for the task that matches the patterns, can be impossible.

Bobrik et al.[1] aggregate information derived from running instances into a summary process model, including completion status, data properties, and timing information. In this paper only the *how* component is discussed. Also the paper does not discuss the property aggregation operations for different types of properties. Polyvyanyy et al. [13] defined abstraction criteria based on slider approach which separate significant from insignificant tasks, which are subsequently aggregated based on structural process patterns. Although the abstraction criteria can be extended to cover more abstraction scenarios, they are limited to those properties which have a quantitative measurement such as cost and execution duration.

Fahland et al.[5] proposed a simplification strategy for Petri nets that is based on unfolding and subsequent transformation and folding regardless of abstraction purposes. Overall most of the process model abstraction approaches focus on only the *how* component, reduce a process model based on predefined patterns, consider only a limited number of properties, and are not user interactive. In contrast, we take other process abstraction components into account, we do not restrict the preservation or aggregation of a task based on its region and the corresponding patterns, we provide an aggregation solution for properties with different types. Finally using a questionnaire, different needs of a user from abstracting a process model are taken into account.

In process model configuration literature, La Rosa et al. [8] introduce a questionnaire approach for system configuration. The questionnaire elicits facts about the desired process variant. Facts are associated with actions that adapt a given generic reference process to suit the users requirements. Gottschalk et al. [7] summarizes similar approaches for EPCs and YAWL, where tasks in the process are either blocked or hidden. In contrast, our approach does not rely on a reference process with variation points. Instead, we constrain the resulting abstraction relation and employ search techniques to compute suitable abstractions for tasks and data entities in the process model.

## 8 Conclusion

We presented a configuration method for generating tailored business process abstractions that satisfy user-selected abstraction criteria. Our method is based on imposing constraints on the abstraction relation, which is computed using a generic search procedure using a library of generic and domain-specific abstraction operators. Elicitation of relevant abstraction constraints is simplified by a questionnaire-based approach that hides much of the formal underpinnings of our method. Our abstraction approach goes beyond simple structural transformation and also considers data properties and flow aspects within the process model in the abstraction.

In this paper we focused on conceptual elaboration of our method. Immediate future work will focus on empirical evaluation of the approach on large business processes, and on incorporating preference orderings into our search and operator selection algorithms. Other avenues for research are incremental elicitation of abstraction constraints in the context of incremental process exploration and integration of process instance-based properties and further reachability-based criteria.

## 9 Acknowledgement

We would like to acknowledge that this research was supported by the Australian Research Council (ARC) under grant DP0988961.

## References

1. Bobrik, R., Reichert, M., Bauer, T.: Parameterizable views for process visualization. Tech. rep., Centre for Telematics and Information Technology, University of Twente (2007)
2. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: Proc. BPM. pp. 88–95. Springer (2007)
3. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web 1(3), 281 – 308 (2004)
4. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: Proc. APCCM. pp. 71–80. Australian Computer Society (2007)
5. Fahland, D., van der Aalst, W.: Simplifying mined process models: An approach based on unfoldings. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) Business Process Management, LNCS, vol. 6896, pp. 362–378. Springer (2011)
6. Gottschalk, F., La Rosa, M.: Process configuration in yawl. In: Hofstede, A.H.M., Aalst, W.M.P., Adams, M., Russell, N. (eds.) Modern Business Process Automation, pp. 313–382. Springer (2010)
7. Gottschalk, F., Wagemakers, T., Jansen-Vullers, M., van der Aalst, W., La Rosa, M.: Configurable process models: Experiences from a municipality case study. In: Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 5565, pp. 486–500. Springer Berlin / Heidelberg (2009)
8. La Rosa, M., van der Aalst, W., Dumas, M., ter Hofstede, A.: Questionnaire-based variability modeling for system configuration. Software and Systems Modeling 8, 251–274 (2009)
9. Liu, D.R., Shen, M.: Workflow modeling for virtual processes: an order-preserving process-view approach. Information Systems 28, 505 – 532 (2003)

10. Mayer, W., Killisperger, P., Stumptner, M., Grossmann, G.: A declarative framework for work process configuration. AI EDAM 25(2), 145–165 (2011)
11. Pankratius, V., Stucky, W.: A formal foundation for workflow composition, workflow view definition, and workflow normalization based on petri nets. In: APCCM. pp. 79–88. APCCM '05, Australian Computer Society (2005)
12. Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: A slider approach. In: EDOC. pp. 325–331 (2008)
13. Polyvyanyy, A., Smirnov, S., Weske, M.: Business process model abstraction. In: Handbook on Business Process Management 1, pp. 149–166. Springer (2010)
14. Smirnov, S., Dijkman, R., Mendling, J., Weske, M.: Meronymy-based aggregation of activities in business process models. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) Conceptual Modeling ER 2010, LNCS, vol. 6412, pp. 1–14. Springer (2010)
15. Smirnov, S., Reijers, H., Weske, M., Nugteren, T.: Business process model abstraction: a definition, catalog, and survey. Distributed and Parallel Databases 30, 63–99 (2012)

# Modular Representation of a Business Process Planner

Shahab Tasharrofi, Eugenia Ternovska

Simon Fraser University, Canada
{sta44,ter}@cs.sfu.ca

**Abstract.** The business process planner relies on external services for particular tasks. The tasks performed by each of the providers or the planner are often NP-complete, e.g. the Traveling Salesman Problem. Therefore, finding a combined solution is a computationally (as well as conceptually) complex task. Such a central planner could be used in business process management in e.g. logistics service provider, manufacturer supply chain management, mid-size businesses relying on external web services and cloud computing. The main challenge is a high level of uncertainty and that each module can be described in a different language. The language is determined by its suitability for the task and the expertise of the local developers. To allow for multiple languages, we approach the problem of finding combined solutions model-theoretically. We describe a knowledge representation formalism for representing such systems and then demonstrate how to use it for representing a business process planner. We prove correctness of our representation, describe general properties of modular systems and ideas for how to automate finding solutions.

## 1 Introduction

Formulating AI tasks as model finding has recently become very promising due to the overwhelming success of SAT (propositional satisfiability) solvers and related technology such as ASP (answer set programming) and SMT (satisfiability modulo theories). In our research direction we focus on a particular kind of model finding which we call *model expansion*. The task of model expansion underlies all search problems where for an instance of a problem, which we represent as a logical structure, one needs to find a certificate (solution) satisfying certain specification. For example, given a graph, we are looking for its 3-colouring in a classic NP-search problem. Such search problems occur broadly in applications; they include planning, scheduling, problems in formal verification (where we are looking for a path to a bug), computational biology, and so on. In addition to being quite common, the task of model expansion is generally simpler (for the same logic) than satisfiability from the computational point of view. Indeed, for a given logic $\mathcal{L}$, we have, in terms of computational complexity,

$$\text{MC}(\mathcal{L}) \leq \text{MX}(\mathcal{L}) \leq \text{Satisfiability}(\mathcal{L}),$$

where MC($\mathcal{L}$) stands for model checking (structure for the entire vocabulary of the formula in logic $\mathcal{L}$ is given), MX($\mathcal{L}$) stands for model expansion (structure interpreting a part of the vocabulary is given) and Satisfiability($\mathcal{L}$) stands for satisfiability task (where we are looking for a structure satisfying the formula). A comparison of the complexity of the three tasks for several logics of practical interest is given in [15].

The next step is to extend the framework to a modular setting. In [21], we started to develop a model-theoretic framework to represent search problems which consist of several modules. In this paper, we develop our ideas further through an example of a Business Process Planner (BPP). This planner generalizes a wide range of practical problems. We envision such a planner used as a part of a multi-tool process management system. The task solved by BPP is extremely complex, and doing it manually requires significant resources. The technology is now ready to automate such computationally complex tasks, and our effort is geared towards making the technology available to less specialized users.

In systems like our planner, a high level of uncertainty is present. In our framework, we can model the following types of uncertainty.

- Each agent can see only the inputs and the outputs of other modules, but not their internals. The modules are viewed as black boxes by the outside world. Modules communicate with each other through common vocabulary symbols.
- Modules can be represented using languages that are not known to other modules. Such languages can even be old and no longer supported, as is common for legacy systems.
- Each module (an agent) can have multiple models (i.e., structures satisfying an axiomatization), each representing a possible plan of an individual module. This is a feature that generates uncertainty in planning. We view each module abstractly as a set of structures satisfying the axioms of the module.

The main challenge is that each module can be represented in a different language, reflecting the local problem's specifics and local expertise. Thus, the only way to formalize such a system is model-theoretic. Our goal is not only to formalize, but to eventually develop a method for finding solutions to complex modular systems like the BPP. This is a computationally complex task. Our inspiration for finding solutions to such systems comes from "combined" solvers for computationally complex tasks such as Satisfiability Modulo Theories (SMT). There, two kinds of propagation work interactively – propositional satisfiability (SAT) and theory propagation. In the case of modular systems, each module will have a so-called oracle that is similar to solvers/propagators used in SMT. If the logic language used by a module has a clear model-theoretic semantics, such an oracle (propagator) is easy to construct, but in the most extreme cases, derivations can be even performed by a human expert. At the level of solving, oracles would interact using a common internal solver language with a clear formal semantics. We believe that a formal model-theoretic approach is the right approach to developing a general algorithm for solving modular systems such as the BPP. This is another important motivation for developing a rigorous model-theoretic framework.

In this paper, we demonstrate how to use ideas of model expansion and modular systems together to naturally represent modular systems such as BPP. We prove correctness of our formalization and explain how finding solutions to such systems can be automated.

## 2 Business Process Planner

A business process planner is an entity which plans a particular task by relying on external services for particular tasks. Often, in business, there are cases when one needs to buy services from other service providers. The planner combines services provided by different companies to minimize the cost of the enterprise. The customer needs to allocate required services to different service providers and to ask them for their potential plans for their share. These plans will then be used to produce the final plan, which can be a computationally complex task. The tasks performed by each of the providers are often NP-complete, e.g. the Traveling Salesman Problem. Therefore, finding a combined solution is a computationally (as well as conceptually) complex task. Such a central planner could be used in business process management in many areas such as:

- **Logistics Service Provider** operates on the global scale, uses contracted carriers, local post, fleet management, driver dispatch, warehouse services, transportation management systems, e-business services as well as local logistics service providers with their own sub-modules.
- **Manufacturer Supply Chain Management** uses a supply chains planner relying on transportation, shipping services, various providers for inventory spaces, etc.. It uses services of third party logistics (3PL) providers, which themselves depend on services provided by smaller local companies.
- **Mid-size Businesses Relying on External Web Services and Cloud Computing** Such businesses often use data analysis services, storing, spreadsheet software (office suite), etc.. The new cloud-based software paradigm satisfies the same need in the domain of software systems.

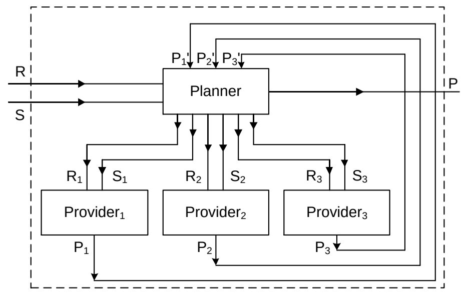

**Fig. 1.** Business Process Planner (BPP).

Figure 1 shows a general representation of a business process planner with three providers. Each of the solid boxes in Figure 1 represents a business entity which, while interested to participate in the process, is not necessarily willing to share the information that has affected their decisions. Therefore, any approach to representing and solving such systems that assumes unlimited access to complete axiomatizations of these entities is impractical.

The business process planner in Figure 1 takes a set $S$ of services and a set $R$ of restrictions (such as service dependencies or deadlines) and generates plan $P$. Each "Provider$_i$" takes a subset of services $S_i$ and their restrictions $R_i$. Provider$_i$ generates a potential plan $P_i$ for subset $S_i$ of services and returns it to "Planner". Planner takes all these partial plans and, if not satisfied with them, reconsiders service allocations or providers. However, if satisfied, it outputs plan $P$ by combining partial plans $P_i$.

## 3 Background: Model Expansion Task

In [17], the authors formalize combinatorial search problems as the task of *model expansion (MX)*, the logical task of expanding a given (mathematical) structure with new relations. Formally, the user axiomatizes their problem in some logic $\mathcal{L}$. This axiomatization relates an instance of the problem (a *finite structure*, i.e., a universe together with some relations and functions), and its solutions (certain *expansions* of that structure with new relations or functions). Logic $\mathcal{L}$ corresponds to a specification/modelling language. It could be an extension of first-order logic, or an ASP language, or a modelling language from the Constraint Programming (CP) community such as ESSENCE [12]. MX task underlies many practical approaches to declarative problem solving.

Recall that a vocabulary is a set of non-logical (predicate and function) symbols. An interpretation for a vocabulary is provided by a *structure*, which consists of a set, called the domain or universe and denoted by $dom(.)$, together with a collection of relations and (total) functions over the universe. A structure can be viewed as an *assignment* to the elements of the vocabulary. An expansion of a structure $\mathcal{A}$ is a structure $\mathcal{B}$ with the same universe, and which has all the relations and functions of $\mathcal{A}$, plus some additional relations or functions. The task of model expansion for an arbitrary logic $\mathcal{L}$ (abbreviated $\mathcal{L}$-MX), is:

**Model Expansion for logic $\mathcal{L}$**
Given: *(1)* An $\mathcal{L}$-formula $\phi$ with vocabulary $\sigma \cup \varepsilon$ and
*(2)* A structure $\mathcal{A}$ for $\sigma$
Find: an expansion of $\mathcal{A}$, to $\sigma \cup \varepsilon$, that satisfies $\phi$.

We call $\sigma$, the vocabulary of $\mathcal{A}$, the *instance* vocabulary, and $\varepsilon := vocab(\phi) \setminus \sigma$ the *expansion* vocabulary[1].

*Example 1.* The following formula $\phi$ in the language of logic programming under answer set semantics constitutes an MX specification for Graph 3-colouring.

$$\begin{array}{l} 1\{R(x), B(x), G(x)\}1 \leftarrow V(x). \\ \perp \leftarrow E(x,y), R(x), R(y). \\ \perp \leftarrow E(x,y), G(x), G(y). \\ \perp \leftarrow E(x,y), B(x), B(y). \end{array}$$

An instance is a structure for vocabulary $\sigma = \{E\}$, i.e., a graph $\mathcal{A} = \mathcal{G} = (V; E)$. The task is to find an interpretation for the symbols of the expansion vocabulary $\varepsilon = \{R, B, G\}$ such that the expansion of $\mathcal{A}$ with these is a model of $\phi$:

[1] By ":=" we mean "is by definition" or "denotes".

$$\underbrace{\overbrace{(V; E^{\mathcal{A}}}^{\mathcal{A}},\ R^{\mathcal{B}}, B^{\mathcal{B}}, G^{\mathcal{B}})}_{\mathcal{B}} \models \phi.$$

The interpretations of $\varepsilon$, for structures $\mathcal{B}$ that satisfy $\phi$, are exactly the proper 3-colourings of $\mathcal{G}$.

Given a specification, we can talk about a set (class) of $\sigma \cup \varepsilon$-structures which satisfy the specification. Alternatively, we can simply talk about a set (class) of $\sigma \cup \varepsilon$-structures as an MX-task, without mentioning a particular specification the structures satisfy.

*Example 2 (BPP as Model Expansion).* In Figure 1, both the planner box and the provider boxes can be viewed as model expansion tasks. For example, the box labeled with "Provider$_1$" can be abstractly viewed as an MX task with instance vocabulary $\sigma = \{S_1, R_1\}$ and expansion vocabulary $\varepsilon = \{P_1\}$. The task is: given some services $S_1$ and some restrictions $R_1$, find a plan $P_1$ to deliver services in $S_1$ such that all restrictions in $R_1$ are satisfied.

Moreover, in Figure 1, the bigger box with dashed borders can also be viewed as an MX task with instance vocabulary $\sigma' = \{S, R\}$ and expansion vocabulary $\varepsilon' = \{P\}$. This task is a compound MX task whose result depends on the internal work of all the providers and the planner.

## 4 Modular Systems

This section presents the main concepts of modular systems.

**Definition 1 (Primitive Module).** *A* primitive module *$M$ is a set (class) of $\sigma_M \cup \varepsilon_M$-structures, where $\sigma_M$ is the instance vocabulary, $\varepsilon_M$ is the expansion vocabulary.*

Each module can be axiomatized in a different logic. However, we can abstract away from the logics and study modular systems entirely model-theoretically.

A modular system is formally described as a set of primitive modules (individual sets of structures) combined using the operations of:

1. Projection($\pi_\tau(M)$) to restrict a module's vocabulary,
2. Composition($M_1 \rhd M_2$) to connect outputs of $M_1$ to $M_2$,
3. Intersection($M_1 \cap M_2$),
4. Union($M_1 \cup M_2$),
5. Feedback($M[R = S]$) which connects output $S$ of $M$ to its inputs $R$.

Formal definitions of these operations were introduces in [21] and are given below.

The initial development of of our algebraic approach was inspired by [14]. In contrast to that work, our contribution was to use a model-theoretic setting, simplify the framework and add a loop operator which increases the expressive power significantly, by one level in the polynomial time hierarchy. Here, we only consider modular systems that do not use the union operator.

**Operations for Combining Modules**

**Definition 2 (Composable, Independent [14]).** *Modules $M_1$ and $M_2$ are* composable *if $\varepsilon_{M_1} \cap \varepsilon_{M_2} = \emptyset$ (no output interference). Module $M_1$ is* independent *from $M_2$ if $\sigma_{M_1} \cap \varepsilon_{M_2} = \emptyset$ (no cyclic module dependencies).*

**Definition 3 (Modular Systems).** *Modular systems are built inductively from constraint modules using projection, composition, union and feedback operators:*

**Base Case** *A primitive module is a modular system.*

**Projection** *For modular system $M$ and $\tau \subseteq \sigma_M \cup \varepsilon_M$, modular system $\pi_\tau(M)$ is defined such that (a) $\sigma_{\pi_\tau(M)} = \sigma_M \cap \tau$, (b) $\varepsilon_{\pi_\tau(M)} = \varepsilon_M \cap \tau$, and (c) $\mathcal{B} \in \pi_\tau(M)$ iff there is a structure $\mathcal{B}' \in M$ with $\mathcal{B}'|_\tau = \mathcal{B}$.*

**Composition** *For composable modular systems $M$ and $M'$ (no output interference) with $M$ independent from $M'$ (no cyclic module dependencies), $M \rhd M'$ is a modular system such that (a) $\sigma_{M \rhd M'} = \sigma_M \cup (\sigma_{M'} \setminus \varepsilon_M)$, (b) $\varepsilon_{M \rhd M'} = \varepsilon_M \cup \varepsilon_{M'}$, and (c) $\mathcal{B} \in (M \rhd M')$ iff $\mathcal{B}|_{vocab(M)} \in M$ and $\mathcal{B}|_{vocab(M')} \in M'$.*

**Union** *For modular systems $M_1$ and $M_2$ with $\sigma_{M_1} \cap \sigma_{M_2} = \sigma_{M_1} \cap \varepsilon_{M_2} = \varepsilon_{M_1} \cap \sigma_{M_2} = \emptyset$, the expression $M_1 \cup M_2$ defines a modular system such that (a) $\sigma_{M_1 \cup M_2} = \sigma_{M_1} \cup \sigma_{M_2}$, (b) $\varepsilon_{M_1 \cup M_2} = \varepsilon_{M_1} \cup \varepsilon_{M_2}$, and (c) $\mathcal{B} \in (M_1 \cup M_2)$ iff $\mathcal{B}|_{vocab(M_1)} \in M_1$ or $\mathcal{B}|_{vocab(M_2)} \in M_2$.*

**Feedback** *For modular system $M$ and $R \in \sigma_M$ and $S \in \varepsilon_M$ being two symbols of similar type (i.e., either both function symbols or both predicate symbols) and of the same arities; expression $M[R = S]$ is a modular system such that (a) $\sigma_{M[R=S]} = \sigma_M \setminus \{R\}$, (b) $\varepsilon_{M[R=S]} = \varepsilon_M \cup \{R\}$, and (c) $\mathcal{B} \in M[R = S]$ iff $\mathcal{B} \in M$ and $R^{\mathcal{B}} = S^{\mathcal{B}}$.*

Further operators for combining modules can be defined as combinations of basic operators above. For instance, [14] introduced $M_1 \blacktriangleright M_2$ (composition with projection operator) as $\pi_{\sigma_{M_1} \cup \varepsilon_{M_2}}(M_1 \rhd M_2)$. Also, $M_1 \cap M_2$ is defined to be equivalent to $M_1 \rhd M_2$ (or $M_2 \rhd M_1$) when $\sigma_{M_1} \cap \varepsilon_{M_2} = \sigma_{M_2} \cap \varepsilon_{M_1} = \varepsilon_{M_1} \cap \varepsilon_{M_2} = \emptyset$.

**Definition 4 (Models/Solutions of Modular Systems).** *For a modular system $M$, a $(\sigma_M \cup \varepsilon_M)$-structure $\mathcal{B}$ is a* model *of $M$ if $\mathcal{B} \in M$.*

Since each modular system is a set of structures, we call the structures in a modular system *models* of that system.

*Example 3 (Stable Model Semantics).* Let $P$ be a normal logic program. We know $S$ is a stable model for $P$ iff $S = Dcl(P^S)$ where $P^S$ is the reduct of $P$ under set $S$ of atoms (a positive program) and $Dcl$ computes the deductive closure of a positive program, i.e., the smallest set of atoms satisfying it. Now, let $M_1(S, P, Q)$ be the module that given a set of atoms $S$ and ASP program $P$ computes the reduct $Q$ of $P$ under $S$. Also, let $M_2(Q, S')$ be a module that, given a positive logic program $Q$, returns the smallest set of atoms $S'$ satisfying $Q$. Now define $M$ as follows:

$$M := \pi_{\{P,S\}}((M_1 \rhd M_2)[S = S']).$$

Then, $M$ represents a module which takes a ground ASP program $P$ and returns all and only its stable models. Figure 2 shows the corresponding diagram of $M$.

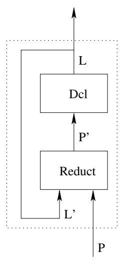

**Fig. 2.** Modular Representation of an ASP Solver.

On a model-theoretic level, this module represents all possible ASP programs and all their solutions, where programs are encoded by structures. While such a module is certainly possible, a more practical use would be where one module corresponds to a particular ASP program such as the one for graph 3-colouring in Example 1. Nevertheless, the Example 3 is useful because it represents a well-known construction and illustrates several concepts associated with modular systems.

*Example 4 (BPP as a Modular System).* Figure 1 can be viewed as a modular representation of the business process planner. There, each primitive module is represented by a box with solid borders and our module of interest is the compound module which is shown by the box with dotted borders. This module is specified by the following formula:

$$BPP := \pi_{\{S,R,P\}}(\text{Planner} \rhd ((\text{Provider}_1 \cap \text{Provider}_2 \cap \text{Provider}_3)[P_1' = P_1][P_2' = P_2][P_3' = P_3])). \quad (1)$$

As in Figure 1, the only vocabulary symbols which are important outside the big box with dashed borders are $S$, $R$ and $P$. There are also three feedbacks from $P_1$ to $P_1'$, $P_2$ to $P_2'$, and $P_3$ to $P_3'$.

## 5 Details of the Business Process Planner

In this section we give a detailed description of one of the many kinds of business process planners, i.e., a logistics service provider on the global scale which hires local carriers and warehouses. So, in Figure 1, "Planner" refers to the global entity and "Provider" refers to local entities.

The logistics provider need a plan to execute the services so that all restrictions are met. Some sample restrictions are: (1) latest delivery time (e.g., Halloween masks should be in stores before Halloween), (2) type of carrying vehicles (perishable products need refrigerator trucks), and (3) level of care needed (glass-works should be carried carefully).

We say that a plan $P$ is good for a set of services $S$ and restrictions $R$ ($Good(P, S, R)$) if $P$ does all services in $S$ and satisfies all restrictions in $R$. For simplicity, here, we only consider time restrictions, i.e., the value of $t(i)$ is the (latest) delivery time for item $i$. There are also functions $s(.)$ and $d(.)$ to indicate the source and the destination of an item. For an item $i$, a plan is a sequence of cities $\langle c_1, \cdots, c_n \rangle$ along with its pickup times $pt(i,j)$ and arrival time $at(i,j)$. So, we have that[2]:

$$\begin{array}{l}
\forall i \in \mathit{Items}\ (P(i) = \langle c_0, \cdots, c_n \rangle \supset \\
\quad c_0 = s(i) \wedge c_n = d(i)), \\
\forall i \in \mathit{Items}\ (P(i) = \langle c_0, \cdots, c_n \rangle \supset at(i,n) \leq t(i)), \\
\forall i \in \mathit{Items}\ (P(i) = \langle c_0, \cdots, c_n \rangle \supset \\
\quad \forall j \in [1,n]\ (\mathit{connected}(c_{j-1}, c_j)), \\
\forall i \in \mathit{Items}\ (P(i) = \langle c_0, \cdots, c_n \rangle \supset \\
\quad \forall j \in [0,n]\ (pt(i,j) \geq at(i,j))), \\
\forall i \in \mathit{Items}\ (P(i) = \langle c_0, \cdots, c_n \rangle \supset \\
\quad \forall j \in [1,n]\ (at(i,j) = pt(i,j-1) + \mathit{time}(c_{j-1}, c_j))).
\end{array}$$

Intuitively, these axioms tell us that a plan for each item should: (1) start at the source and end at the destination, (2) arrive at the destination sooner than their latest delivery time, (3) pass through cities which are connected to each other, (4) respect time constraints, i.e., be picked up at a city after they have arrived at that city, and (5) respect the distance between cities. Certainly, a good plan needs to satisfy all these conditions, but, of course, this does not give us a full axiomatization of the problem. Here, we do not even intend to do that, because we believe that this is enough for the reader to have a good idea on how such full axiomatizatins look like.

Given a definition of a good plan, one can define the intended solutions of a business process planner as below:

**Definition 5 (Intended Solutions).** *Let $BPP$ be a business process planner with access to $n$ providers. Structure $\mathcal{B}$ is an intended solution of $BPP$ if:*

1. *$P^{\mathcal{B}}$ is good for $S^{\mathcal{B}}$ and $R^{\mathcal{B}}$, i.e., $\mathcal{B} \models \mathit{Good}(P, S, R)$,*
2. *All atomic actions $A$ of $P^{\mathcal{B}}$ (here, moving items between different cities) are doable by one of the $n$ providers.*

So, by Definition 5, if some set of services cannot be executed under some restrictions, there should not exist any solution for the whole modular system which interprets $S$ by those services and $R$ by those restrictions.

Now, to ensure that the intended solutions of modular system in Figure 1 coincide with the models of this modular system under our modular semantics, we use the declarative representations below for the modules:

[2] We slightly abuse logic notations here to keep the axiomatization simpler. For example, we use the notation $P(i) = \langle c_0, \cdots, c_n \rangle$ to denote that item $i$ takes a path starting at city $c_0$ and then going to city $c_1$ and so on until it getting to city $c_n$. In practice, such a specification can be realized using two expansion function "$len(.)$" (to show the length of the path of an item) and "$loc(.,.)$" (to show its location). As an example, this is how the first axiom above is rewritten in terms of "$len$" and "$loc$":

$$\forall i \in \mathit{Items}\ (loc(i,0) = s(i) \wedge loc(i, len(i)) = d(i)).$$

Module "Planner" is the set of structures over vocabulary $\sigma = \{R, S, P_1, \cdots, P_n\}$ and $\varepsilon = \{P, S_1, \cdots, S_n, R_1, \cdots, R_n\}$ which satisfies:

$$Good(P, S, R) \Leftrightarrow \bigwedge_{i \in \{1, \cdots, n\}} Good(P_i, S_i, R_i), \tag{2}$$

$$P \text{ is a join of sub-plans } P_i (\text{for } i \in \{1, \cdots, n\}). \tag{3}$$

This module is easily specifiable in extended FO.

Module "Provider$_i$" is the set of structures over vocabulary $\sigma = \{R_i, S_i\}$ and $\varepsilon = \{P_i\}$ which satisfy $Good(P_i, S_i, R_i)$. Each such module "Provider$_i$" can be specified using mixed integer linear programming. Also, in practice, many such modules are realized using special purpose programs (so, no standard language). Our framework enables us to deal with such programs in a unified way.

**Proposition 1 (Correctness).** *Structure $\mathcal{B}$ is in modular system $BPP := \pi_{\{S,R,P\}}(Planner \rhd ((Provider_1 \cap \cdots \cap Provider_n)[P'_1 = P_1] \cdots [P'_n = P_n]))$ (where "Planner" and "Provider$_i$"s are defined as above) iff $\mathcal{B}$ is an intended solution of $BPP$ (according to Definition 5).*

*Proof.* (1) Take $\mathcal{B}$ which satisfies all modules, each $P_i^{\mathcal{B}}$ has to be good for $S_i^{\mathcal{B}}$ and $R_i^{\mathcal{B}}$. Therefore, $P^{\mathcal{B}}$ is good for $S^{\mathcal{B}}$ and $R^{\mathcal{B}}$. Thus, $\mathcal{B}$ is an intended solution of $BPP$. (2) Conversely, take an intended solution $\mathcal{B}$. $P^{\mathcal{B}}$ should be such that $P^{\mathcal{B}}$ is good for $S^{\mathcal{B}}$ and $R^{\mathcal{B}}$. So, set $\mathcal{B}'$ to be an expansion of $\mathcal{B}$ such that $P_i^{\mathcal{B}'}$ is the parts of $P^{\mathcal{B}}$ which are executed by $i$-th provider. Also, $S_i^{\mathcal{B}'}$ is those services that $P_i^{\mathcal{B}'}$ executes and $R_i^{\mathcal{B}'}$ is those restrictions satisfied by $P_i^{\mathcal{B}'}$, e.g., the latest delivery time of item $a$ is the delivery time of $a$ according to $P_i^{\mathcal{B}'}$. Now, $P_i^{\mathcal{B}'}$ is good for $S_i^{\mathcal{B}'}$ and $R_i^{\mathcal{B}'}$. So, $\mathcal{B} \in BPP$.

## 6 The Bigger Picture

**Complexity of the modular framework** In this subsection, we summarize one of our important results about the modular framework from [21]. In order to do so, we first have to introduce the concepts of totality, determinacy, monotonicity, anti-monotonicity, etc. For lack of space, we do this through examples. The exact definitions can be found in [21].

*Example 5 (Reachability).* Consider the following model expansion task with $\sigma = \{S, E, B\}$ and $\varepsilon = \{R\}$:

$$\begin{array}{l} R(v) \leftarrow S(v). \\ R(v) \leftarrow R(u), E(u, v), \ \textbf{not}\ B(u). \end{array} \tag{4}$$

where $S$ represents a set of source vertices of a graph, $E$ represents the edges of the graph, $B$ represents a set of blocked vertices of the graph and $R$ represents a set of vertices which can be reached from a source vertex without passing any blocked vertices.

Through this section, let $M_R$ denote a primitive module which represents the MX task of Example 5. Obviously, $\sigma_{M_R} = \{S, E, B\}$ and $\varepsilon_{M_R} = \{R\}$: Then, we have:

**Totality:** Module $M_R$ is $\{S, E, B\}$-$\{R\}$-total because for every interpretation of $S$, $E$ and $B$, there is an interpretation for $R$ which is a stable model of program 4.

**Determinacy:** Module $M_R$ is $\{S, E, B\}$-$\{R\}$-deterministic because for every interpretation of $S$, $E$ and $B$, there is at most one interpretation for $R$ which satisfies (4).

**Monotonicity:** Module $M_R$ is $\{E\}$-$\{S, B\}$-$\{R\}$-monotone because if we fix the interpretation of symbols $S$ and $B$ and increase the set of edges $E$, then the interpretation of $R$ (reachable vertices) increases.

**Anti-monotonicity:** Module $M_R$ is $\{E\}$-$\{S, B\}$-$\{R\}$-anti-monotone because if we fix the interpretation of $S$ and $E$ and increase the set of blocked vertices ($B$), then, the set $R$ of reachable vertices decreases.

**Polytime Checkability/Solvability:** Module $M_R$ is both polytime checkable (because one can check in polynomial time if a structure $\mathcal{B}$ belongs to $M_R$) and polytime solvable (because, given interpretations to $S$, $E$ and $B$, one can compute the only valid interpretation for $R$ in polynomial time). However, the module $M_C$ which corresponds to the graph 3-coloring (Example 1) is polytime checkable but not polytime solvable (unless P=NP).

Now, we are ready to restate our main theorem from [21]. We should however point out one difference to the readers who are not accustomed to the logical approach to complexity: In theoretical computing science, a problem is a subset of $\{0, 1\}^*$. However, in descriptive complexity, the equivalent definition of a problem being a set of structures is adopted. The following theorem gives a capturing result for complexity class NP:

**Theorem 1 (Capturing NP over Finite Structures).** *Let $\mathcal{K}$ be a problem over the class of finite structures closed under isomorphism. Then, the following are equivalent:*

1. *$\mathcal{K}$ is in NP,*
2. *$\mathcal{K}$ is the models of a modular system where all primitive modules $M$ are $\sigma_M$-$\varepsilon_M$-deterministic, $\sigma_M$-total, $\sigma_M$-$vocab(\mathcal{K})$-$\varepsilon_M$-anti-monotone, and polytime solvable,*
3. *$\mathcal{K}$ is the models of a modular system with polytime checkable primitive modules.*

Note that Theorem 1 shows that when basic modules are restricted to polytime checkable modules, the modular system's expressive power is limited to NP. Without this restriction, the modular framework can represent Turing-complete problems. As an example, one can encode Turing machines as finite structures and have modules that accept a finite structure iff it corresponds to a halting Turing machine.

Theorem 1 shows that the feedback operator causes a jump in expressive power from P to NP (or, more generally, from $\Delta_k^P$ to $\Sigma_{k+1}^P$).

*Example 6 (Stable Model Semantics).* In Example 3, firstly, note that primitive module $M_1$ is $\{S\}$-total and $\{S\}$-$\{P\}$-$\{Q\}$-anti-monotone, and also polytime solvable. Secondly, module $M_2$ is $\{Q\}$-total, $\{Q\}$-$\{\}$-$\{S'\}$-monotone and, again, polytime solvable. However, the module $M := \pi_{\{P,S\}}((M_1 \rhd M_2)[S = S'])$ is neither total nor monotone or anti-monotone. Moreover, $M$ represents the NP-complete problem of finding a stable model for a normal logic program. This shows how, in the modular framework, one can describe a complex modular system in terms of very simple primitive modules.

**Solving modular systems** We would like to find a method for solving complex tasks such as the application in this paper, without limiting to the particular structure of Figure 1, and without committing to a particular language. The language is determined by its suitability for the task and the expertise of the local developers. For example, the planner module is more easily specified as a SAT (propositional satisfiability) problem, while some provider modules are most easily specified using MILP (mixed integer linear programming), and global constraints with CP (constraint programming). A module performing scheduling with exceptions is more easily specified with ASP (answer set programming).

In our research, we focus on the central aspect of this challenging task, namely on *solving the underlying computationally complex task*, for arbitrary modular systems and arbitrary languages suitable for specifying combinatorially hard search/optimization problems. Our approach is model-theoretic. We aim at finding structures satisfying multi-language constraints of the modular system, where the system is viewed as a function of individual modules. *Our main goal is to develop and implement an algorithm that takes a modular system as its input and generates its solutions.* Such a prototype system should treat each primitive module as a black-box (i.e., should not assume access to a complete axiomatization of the module). Not assuming complete knowledge is essential in solving problems like business process planning.

We take our inspiration in how “combined” solvers are constructed in the general field of declarative problem solving. The field consists of many areas such as MILP, CP, ASP, SAT, and each of these areas has many solvers, including powerful “combined” solvers such as SMT, ASP-CP solvers. There are several methods e.g. cutting plain techniques of ILP, the formal interaction between SAT and theory solvers in SMT, etc. used in different communities. We made the fundamental observation [22] that while different on the surface, the techniques are similar when looked at model-theoretically. We proposed that those general principles can be used to develop a new method of solving modular systems as in the example above.

## 7 Related Work

In [21],we continued the line of research initiated in [14]. We introduced MX-based modular systems and extended the previous work in several ways such as adding the feedback (loop) operator, thus drastically increasing the expressive power. The current paper shows one of the important real-world applications of systems with loops. In our modelling of the business process planner, we use the language independence of modular systems in an essential way. This is an essential property because, in practice, providers use domain-specific software which may not belong to a well-studied logic. This property separates the modular framework of [21] from many other languages which support modularity such as modular logic programs [7, 18, 13], and frameworks with multiple languages [19, 10].

An early work on adding modularity to logic programs is [7]. There, the authors derive a semantics for modular logic programs by viewing a logic program as a generalized quantifier. This work is continued by [18] to introduce modular equivalence in normal logic programs under the stable model semantics. That work, in turn, is extended to define modularity for disjunctive programs in [13]. The last two papers focus on introducing modular programming in logic programs and dealing with difficulties that arise there.

Applications such as business process planning need an abstract notion of a module, independent from the languages used. Our MX-based modular framework is well-suited for this purpose. That cannot be said about many other approaches of adding modularity to ASP languages and FO(ID) (such as those described in [2, 1, 6]) because they address different goals.

Modular programming enables ASP languages to be extended by constraints or other external relations. This view is explored in [8, 9, 20, 3, 16]. While this view is advantageous in its own right, we needed an approach that is completely model-theoretic. Also, some practical modelling languages incorporate other modelling languages. For example, X-ASP [19] and ASP-PROLOG [10] extend prolog with ASP. Also ESRA [11], ESSENCE [12] and Zinc [5] are CP languages extended with features from other languages. Such practical modelling languages are further proof that combining different languages is extremely important for practitioners. We take this view to its extreme by looking at modules as only sets of structures and, thus, having no dependency on the language they are described in. The existing practical languages with support for specific languages could not have been applied to our task.

Yet another direction to modularity is the multi-context systems. In [4], the authors introduced non-monotonic bridge rules to the contextual reasoning and originated an interesting and active line of research followed by many others for solving or explaining inconsistencies in non-monotonic multi-context systems. However, we believe that this application cannot be naturally described as a multi-context system because it is impractical to define the concepts of a logic, a knowledge-base and an acceptability relation (these are concepts that are essential to define in multi-context systems) for a domain-specific application which might not use any known logical fragment.

## 8 Conclusion and Future Work

In this paper, we introduced an important range of real-world applications, i.e., business process planning. We discussed several examples of where this general scheme is used. Then we represented this problem as a model expansion task in the modular setting introduced in [21]. We gave a detailed description of the modules involved in describing business process planning in the modular framework and proved the correctness of our representation. Our main challenge is to devise an appropriate mathematical abstraction of "combined" solving. Remaining particular tasks include:

**Algorithm Design and Implementation** We will design and implement an algorithm that given a modular system, computes the models of that modular system iteratively, and then extracts the solutions.

**Reduction in Search Space** We will improve our algorithm by using approximation methods proposed in [21]. These methods correspond to least fixpoint and well-founded model computations (but in modular setting). We will extend our algorithm so that it prunes the search space by propagating information from the approximation process to the solver.

## References

1. M. Balduccini. Modules and signature declarations for a-prolog: Progress report. In *Workshop on Software Engineering for Answer Set Programming (SEA 2007)*, pages 41–55, 2007.
2. Chitta Baral, Juraj Dzifcak, and Hiro Takahashi. Macros, macro calls and use of ensembles in modular answer set programming. In Sandro Etalle and Miroslaw Truszczynski, editors, *Logic Programming*, volume 4079 of *Lecture Notes in Computer Science*, pages 376–390. Springer Berlin / Heidelberg, 2006.
3. S. Baselice, P. Bonatti, and M. Gelfond. Towards an integration of answer set and constraint solving. In Maurizio Gabbrielli and Gopal Gupta, editors, *Logic Programming*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer Berlin / Heidelberg, 2005.
4. Gerhard Brewka and Thomas Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*, pages 385–390. AAAI Press, 2007.
5. Maria de la Banda, Kim Marriott, Reza Rafeh, and Mark Wallace. The modelling language zinc. In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006*, volume 4204 of *Lecture Notes in Computer Science*, pages 700–705. Springer Berlin / Heidelberg, 2006.
6. M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions. *Transactions on Computational Logic*, 9(2):1–51, 2008.
7. Thomas Eiter, Georg Gottlob, and Helmut Veith. Modular logic programming and generalized quantifiers. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Logic Programming And Nonmonotonic Reasoning*, volume 1265 of *Lecture Notes in Computer Science*, pages 289–308. Springer Berlin / Heidelberg, 1997.
8. Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 90–96, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.
9. Islam Elkabani, Enrico Pontelli, and Tran Son. Smodels A – a system for computing answer sets of logic programs with aggregates. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 3662 of *Lecture Notes in Computer Science*, pages 427–431. Springer Berlin / Heidelberg, 2005.
10. O. Elkhatib, E. Pontelli, and T.C. Son. Asp – prolog: A system for reasoning about answer set programs in prolog. In *Proc. of Practical Aspects of Declarative Languages, 6th International Symposium, (PADL 2004)*, volume 3057, pages 148–162, Dallas, TX, USA, 2004.
11. Pierre Flener, Justin Pearson, and Magnus Ågren. Introducing ESRA, a relational language for modelling combinatorial problems. In Maurice Bruynooghe, editor, *Logic Based Program Synthesis and Transformation*, volume 3018 of *Lecture Notes in Computer Science*, pages 214–232. Springer Berlin / Heidelberg, 2004.
12. Alan M. Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. Essence: A constraint language for specifying combinatorial problems. *Constraints*, 13:268–306, September 2008.
13. Tomi Janhunen, Emilia Oikarinen, Hans Tompits, and Stefan Woltran. Modularity aspects of disjunctive stable models. *Journal of Artificial Intelligence Research*, 35:813–857, August 2009.
14. Matti Järvisalo, Emilia Oikarinen, Tomi Janhunen, and Ilkka Niemelä. A module-based framework for multi-language constraint modeling. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 5753 of *Lecture Notes in Computer Science*, pages 155–168. Springer Berlin / Heidelberg, 2009.

15. Antonina Kolokolova, Yongmei Liu, David Mitchell, and Eugenia Ternovska. On the complexity of model expansion. In *Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning*, LPAR'10, pages 447–458, Berlin, Heidelberg, 2010. Springer-Verlag.
16. Veena Mellarkod, Michael Gelfond, and Yuanlin Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53:251–287, 2008.
17. David G. Mitchell and Eugenia Ternovska. A framework for representing and solving np search problems. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 1*, pages 430–435. AAAI Press, 2005.
18. Emilia Oikarinen and Tomi Janhunen. Modular equivalence for normal logic programs. In *Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva del Garda, Italy*, pages 412–416, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.
19. T. Swift and D. S. Warren. *The XSB System*, 2009.
20. L. Tari, C. Baral, and S. Anwar. A language for modular answer set programming: Application to ACC tournament scheduling. In *Proc. of Answer Set Programming: Advances in Theory and Implementation*, CEUR-WS, pages 277–292, 2005.
21. S. Tasharrofi and E. Ternovska. A semantic account for modularity in multi-language modelling of search problems. In *FroCoS 2011*.
22. S. Tasharrofi, X. Wu, and E. Ternovska. Solving modular model expansion tasks. In *WLP/INAP 2011*.

# Author Index