1st International Workshop on Model-Driven Engineering For Business Process Management

Milano, Italy, September 1st 2008
Preliminary Proceedings

Cesare Pautasso
Jana Koehler (Eds.)

Prof. Cesare Pautasso
Faculty of Computer Science
University of Lugano
Via Buffi 13, 6900 Lugano, Switzerland
E-mail: `c.pautasso at ieee.org`

Dr. Jana Koehler
IBM Research
Zurich Research Laboratory
Säumerstrasse 4, 8803 Rüschlikon, Switzerland
E-mail: `koe at zurich.ibm.com`

Model-Driven Engineering for Business Process Management 2008

BPM08 Workshop, MDE4BPM08
Milano, Italy, September 1st 2008
Preliminary Proceedings

http://www.inf.unisi.ch/mde4bpm08/

Preface

The 1st Workshop on Model-Driven Engineering for Business Process Management (MDE4BPM08) was held in conjunction with the 6th International Conference on Business Process Management (BPM 2008) on September 1st, 2008, at Politecnico di Milano, Italy.

An improved alignment of the IT infrastructure of an enterprise with its business needs and requirements is a trend that has dominated and driven innovations in information technology over the past couple of years. Model-driven development techniques are used to translate business needs directly into IT solutions. A special area of model-driven engineering, often denoted as business-driven development, focuses on generating SOA-based IT solutions from business process models.

The MDE4BPM workshop collects research contributions about the application of Model-Driven Engineering to Business Process Management. It focuses on research problems that arise when the model-driven engineering and development methodology is applied to automate the whole lifecycle of business process modeling artifacts. As it can be seen from the workshop program, one of the most relevant problems concerns the automatic mapping of high level business process models down to executable IT-level workflows. Also, the opposite problem is relevant: so that the impact of changes of IT-level workflows can be analyzed back in the context of the original business-level models. The workshop also includes two tool demonstrations, showing the potential of the research results presented to make a concrete impact.

We would like to thank the authors for their submissions, and our Program Committee for their hard work in reviewing the papers. For this first edition of MDE4BPM, we received 11 submissions, out of which we were able to accept 5 full papers and one invited paper. Finally, we would like to thank the BPM conference organizers (in particular the workshop chairs: Massimo Mecella and Jian Yang) for their support without which this event would not have happened.

September 2008

Cesare Pautasso & Jana Koehler
MDE4BPM08 Program Chairs

Organization

Program Chairs

Program Committee

Table of Contents

Business to IT Transformations Revisited

Sebastian Stein*[1], Stefan Kühne*[2], and Konstantin Ivanov[1]

[1] IDS Scheer AG
Altenkesseler Str. 17, 66115 Saarbrücken, Germany
{sebastian.stein|konstantin.ivanov}@ids-scheer.com,
http://www.ids-scheer.com/soa/
[2] Business Information Systems, University of Leipzig
Johannisgasse 26, 04103 Leipzig, Germany
kuehne@informatik.uni-leipzig.de,
http://bis.informatik.uni-leipzig.de/
* Both authors contributed equally to this paper.

Summary. Bridging the gap between business and IT is a significant challenge companies face today. One approach is using automated model transformations to derive IT implementations from business process models. In spite of extensive research on such model transformations, the problem is still not solved from a practical point of view. In this paper, we provide a classification of existing literature on the topic and relate these transformation approaches to real-world requirements. We provide a conceptual framework for business to IT transformations that identifies open directions for future research.

1 Introduction

The semantic gap [1, 2] between business process modelling languages (BPML) (like BPMN [3] and EPC [4]) and executable orchestration languages (like BPEL[1]) and its closure by automatic transformations has been a subject of research for a couple of years. Manual approaches are considered as error-prone, time-consuming, and cost-intensive. Developing an automated transformation to bridge the gap is a complex task, because business process models are usually created by business experts with only limited IT background, whereas executable process models must be formally correct so that they can be compiled and run on computer systems.

We[2] evaluate the current state of this research and show what is needed to go forward. First, we classify existing literature (see Sect. 2) according to different criteria. Based on the literature review, we show that all reviewed works only cover a subset of the different criteria. To overcome this problem, we provide a conceptual framework (see Sect. 3), which provides the building blocks of business to IT transformations from a practical point of view.

[1] http://www-128.ibm.com/developerworks/library/ws-bpel/
[2] This research was funded by the German federal ministry of education and research within the public research project OrViA – http://www.orvia.de/

2 Literature Review

2.1 Requirements of Business to IT Transformations

To clarify the concepts and issues related to business to IT transformations, we assume a general transformation process (depicted in Fig. 1) that conforms to our experience in this area. The involved artefacts are outlined as boxes of different sizes representing information sets. They are arranged according to their level of abstraction and the point of their use.

At first, a business analyst describes a business process at an abstract (i. e. business-oriented) level according to functional and control-flow related aspects in a BPML. The abstract business process is enriched with additional information concerning data-flows, service interactions, and other perspectives. These activities are supported by a modelling tool, which for instance highlights syntax errors and inconsistencies or retrieves applicable entities. Then, the refined business process model is translated into an abstract orchestration model. Before executing the orchestration, the model is extended by an IT developer adding technology-related aspects such as service bindings or exception handling.

The artefacts created evolve over time. The IT developer may change the refined orchestration. At the same time, the business analyst may modify the business process, because of changing business requirements, incremental development cycles or inappropriate test results. The changed business process model again is translated in an abstract orchestration model. Since the transformation result conflicts with the evolved refined orchestration, the models must be synchronised.

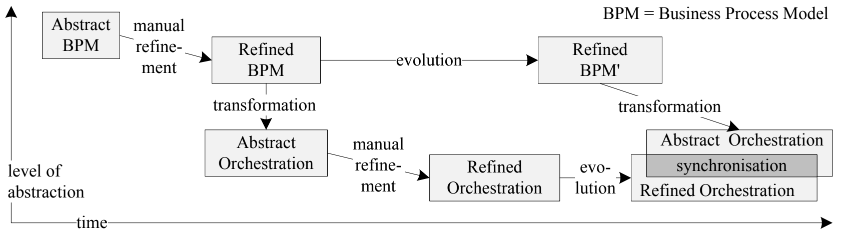


Fig. 1. Artefacts in a Business to IT Transformation Process

The outlined transformation process consists of several variation points and issues. We define variation point as a case where a decision between rather equivalent alternatives for instantiating the transformation process in a specific environment must be taken. We define an issue as an inherent difficulty for which no obvious solution exists.

Variation Point: Business Process Modelling Language The modelling language defines modelling concepts used to describe certain aspects of business processes and the level of abstraction at which these aspects are considered. EPC and BPMN provide accepted notations for this purpose. EPC [4] is a widespread notation used in the SAP reference model and integrated in SAP's Netweaver platform. BPMN [3] is an OMG standard combining ideas of existing modelling languages like UML activity diagrams, UML EDOC business processes, IDEF and others. BPMN provides more technical-oriented concepts, but is also considered as an adequate notation for business experts. Another relevant notation often used in context of object-oriented design is UML activity diagrams.

Issue: Complexity Reduction Strategy Business processes are manifold entities. An adequate description of them for the purpose of transformation introduces a significant cognitive complexity. Different mechanisms may be used for complexity reduction, e. g. introduction of new language constructs, introduction of domain-specific patterns or multi-perspective modelling.

Variation Point: SOA Implementation Technology During a process automation project a decision must be taken which technology to use. The solution space ranges from workflow related languages (like XPDL or YAWL) and web service oriented languages (like BPEL) to domain-specific execution languages used in business domains such as e-government and e-commerce. Besides, there are many proprietary languages offered by different middleware vendors.

Issue: Transformation Power The degree of how the gap between business and IT can be bridged through model transformations depends on various aspects like the process perspectives used, the granularity of entities, the tolerated extent of ambiguity, the incompleteness and inconsistency of input models, the readability of output models, and others.

With respect to granularity and amount of data of input and output models, Mens et al. [5] distinguish horizontal and vertical transformations. A horizontal transformation translates input and output models on the same level of abstraction, whereas a vertical transformation refines the input model to a more detailed level. In the context of business to IT transformations, a horizontal transformation uses a business process model annotated with technical details. In case of a vertical transformation, only an orchestration stub is created. In both cases, the transformation value is rather limited. Vertical transformations try to bridge the gap between business and IT by adding additional information, which is rather difficult to accomplish and has impacts in synchronisation of co-evolved artefacts.

The semantics of BPMLs is often not precisely defined. For instance, since 1994 there is an ongoing debate on the semantics of EPCs [6] reaching from informal semantics [7] to local semantics (e. g. free-choice semantics [8], boolean semantics [6]) and non-local semantics [9]. Similar observations can be made for BPMN and UML activity diagrams [2]. A transformation approach has to handle this issue, e. g. by restricting the input language to an unambiguous subset.

Variation Point: Representation Schema Concerning readability, Mendling et al. [10] classify modelling languages according to two different representation schemes. BPMLs like EPC and BPMN are mostly graph-structured, i. e. the control-flow is specified by arcs representing the execution logic between nodes. In opposite, block-oriented languages define the control flow by nested elements representing sequences, concurrency, alternatives, and loops. BPEL provides concepts for both paradigms in a redundant manner [11]. Consequently, transformation approaches between graph-structured BPMLs and BPEL have a certain freedom of choice.

Issue: Adaptability and Extensibility of the Transformation Customisation of the execution environment or the modelling method may cause adaptation or extensions of the transformation as well. Declarative implementations expressed in relational or graph-based transformation languages tend to provide more adaptable and understandable transformation rules than operational implementations expressed in imperative transformation languages. Functional and template-based approaches may be used for both paradigms.

Issue: Iterative Development Support The co-evolution of process models at different levels of abstraction causes divergent artefacts. As outlined in Fig. 1, conflicting models overlap to some extent. Changed abstract orchestration elements resulting from changed business process models have to be synchronised with changed elements of the refined orchestration. A comprehensive transformation approach defines mechanisms for model synchronisation, which is also known as reconciliation [2].

2.2 Approaches

Considering these issues and variation points, existing transformation approaches can be classified into two main classes. One class of existing work focuses on algorithms for control flow transformation. A second class focuses on more comprehensive support. In the following, we divided the second class further into approaches based on domain-specific language extensions and approaches based on frameworks. For clarity reasons, we restrict the survey to works transforming business process models into the orchestration language BPEL.

Control Flow Centred Approaches Basic considerations about transformations of business process models into executable once (and vice versa) are described by Hauser and Koehler [12]. They use compiler theory techniques to transform process graphs (a subset of UML activity diagrams) into BPEL. Their approach translates sequential process graphs (without OR-splits and OR-joins) into goto-programs, which are further processed by a goto-elimination algorithm. Concurrent parts are translated by identification of sequential *single entry single exit* (SESE) fragments. In a similar approach, van der Aalst and Lassen [13] abstract from concrete modelling languages (like EPC, BPMN or UML activity diagrams) and use workflow nets instead. Similar to [12], they identify and translate structured SESE fragments.

A pragmatic transformation approach from EPC to BPEL is proposed by Ziemann and Mendling [14], who restrict their approach to acyclic EPCs. Depending on connected data elements, EPC functions are mapped to basic BPEL activities (like invoke, receive or reply). The EPC is translated to graph-structured BPEL constructs on the basis of split and join patterns. The approach was implemented as XML transformation working on the EPML format. Kopp et al.

[15] describe a similar approach. The authors map acyclic extended Nautilus EPCs, which allow in contrast to standard EPCs [16] multiple events between functions, to graph-structured BPEL. EPC events are interpreted as transition conditions of BPEL links.

In contrast to graph-structured transformations, the approach described by Specht et al. [17] relies on the workflow patterns by van der Aalst et al. [18] to transform structured EPCs to block-structured BPEL. Inner EPC events are interpreted as business-oriented documentation and are ignored in the transformation process. The algorithm searches for corresponding split and join patterns and transforms such pairs to BPEL switch and flow constructs.

Similar transformation approaches exist for other graph-based BPMLs. White [19] proposes a mapping between BPMN and BPEL on the base of templates and relations between syntax elements. Yet, this description is considered as informal and incomplete [10, 20]. The BPMN specification [3] does not provide significant improvements with respect to this. A more formalised approach is given by Ouyang et al. [20]. For each BPMN activity so-called precondition sets are identified, which are mapped to event condition actions in BPEL (onEvent handlers). The authors emphasise that their approach can be improved according to compactness and comprehensibility aspects. Following this intention, Ouyang et al. [21] combine this approach with structure identification mechanism based on components [13].

Approaches Based on Domain-Specific Language Extensions Several approaches rely on UML activity diagrams extended by self-defined or standardised UML profiles. Mantell [22] proposes a BPEL-specific profile to model and transform UML activity diagrams. Activities are marked with stereotypes such as ≪process≫, ≪receive≫ or ≪invoke≫. Bordbar and Staikopoulos [23] use a specific UML profile, too. Activity diagrams modelled by business experts are refined and enriched with stereotypes such as ≪CallOperationAction≫ or ≪datastore≫. The enriched process models are transformed to BPEL for which the authors outline a few transformation rules. Skogan et al. [24] take existing services in terms of WSDL imports into account. WSDL descriptions are transformed to UML class diagrams, which are referenced by UML activity diagrams. The transformation to BPEL code relies on technical oriented information. For example, activities are stereotyped with ≪WebServiceCall≫ or ≪DataTransformation≫. Heckel and Voigt [25] define a BPEL-specific UML profile for component, class, and activity diagrams. They call for synchronism between the UML source and the BPEL target, which allows iterative incremental development. For this purpose, a declarative transformation approach using graph transformation techniques is proposed. Another BPEL generation approach is given by Yu et al. [26]. This approach differs from other UML profile based approaches by relying on a standard UML profile (EDOC CCA[3]) and using OMG's standard transformation technique QVT.

[3] http://www.omg.org/technology/documents/formal/edoc.htm

Following the intention of EPC transformation approaches like [14, 15, 17], Schmelzle [27] transforms acyclic EPCs to graph-structured BPEL processes. He explicitly differentiates between business oriented and implementation oriented levels of abstractions. The author declines using default values for aspects not expressed in the business process models but required on the implementation level. Instead, an intermediate level is introduced, where additional information can be supplemented using annotations.

Framework-Based Approaches Another class of approaches proposes extensible frameworks, which have to be adapted to the specific requirements of an application domain. Allweyer [28] applies model-driven architecture (MDA) concepts to the area of business process management and outlines an EPC framework based on high-level process patterns. The conceptual framework requires the specification of validation rules for source models, templates for target models, and transformation rules defining the execution logic of templates. The proposed approach should be applicable for recurring business related as well as implementation related problems. The outlined approach is presented as a control flow centred non-iterative one-step refinement, whereas extensions according to a multi-perspective and multi-step approach are possible.

Considering multi language support as an essential feature in design and implementation phases of process-driven SOAs, Zdun and Dustdar [29] propose a language engineering framework based on model-driven development concepts. The framework allows the recursive specification of modelling patterns based on pattern primitives. For example, to decompose the control flow perspective the concepts Macroflow, Macroflowsteps, Microflow, and Microflowsteps are introduced. The authors refer to code generation capabilities of the framework but without any detailed explanation.

Instead of a pattern based approach, Roser et al. [30] propose a language oriented framework. It allows the extension of standard workflow models with pre-defined domain concepts, such as application, service or offer. Additionally, to support different expectations, experiences, and skills of business experts and IT developers, language concepts can be visualised differently using customised concrete syntaxes. For the purpose of adaptability and reuse the framework aims at the separation of domain specific and domain independent concerns. It is therefore structured into domain specific language adapters and code generation templates and domain independent process transformation and rephrasing components.

2.3 Evaluation

We consider the variation points and issues discussed in Sect. 2.1 to evaluate the transformation approaches. Because of the different intentions, we discuss control flow centred and holistic approaches separately.

Control flow centred transformation approaches focus on the process perspective, which according to [31] is the most essential dimension of process modelling languages and engines. The mapping of the functional perspective is basically restricted to specialisation refinements. Abstract concepts (e. g. EPC functions) are replaced by more concrete concepts with reduced extension and increased intension (e. g. BPEL receive, reply or invoke activities). Other refinement strategies (see [32, 33]) such as functional or aspectual decomposition, choice of representation, choice of algorithm, concretisation, elaboration, and realisation are only partly used. We therefore regard these approaches as horizontal transformations (see Table 1(a)). Control flow centred approaches deal with different process representation schemes (graph-oriented, block-oriented) and can therefore be classified according to the four transformation strategies for business process models described in [10]: Element-Preservation (EP), Element-Minimization (EM), Structure-Identification (SI) and Structure-Maximization (SM). Since most approaches are formally described, the underlying programming paradigm used to implement the approach can be determined. The spectrum ranges from imperative and functional implementations to relational, graph-based, and template-based approaches [5].

Table 1. Evaluation

(a) Control flow centred approaches HT/VT = Horizontal/Vertical transformation, EP/EM = Element Preservation/Minimization, SI/SM = Structure Identification/Maximization	Input/Output		Abstr.		Strategy				Paradigm					Descr.	
	Source	Target	HT	VT	EP	EM	SI	SM	Imper.	Funct.	Relat.	Graph	Templ.	Formal	Semif.
[14] Ziemann/Mendling 2005	EPK	BPEL	×		×				×					×	
[20] Ouyang et al. 2006a	BPMN	BPEL	×		×					×				×	
[15] Kopp et al. 2006	N-eEPK	BPEL	×			×			×					×	
[17] Specht et al. 2005	EPK	BPEL	×				×						×		×
[19] White 2004	BPMN	BPEL	×				×				×		×		×
[12] Hauser/Koehler 2004	UML AD	BPEL	×					×	×					×	
[13] Aalst/Lassen 2005	WFN	BPEL	×					×	×					×	
[21] Ouyang et al. 2006b	BPMN	BPEL	×					×	×	×				×	

(b) Domain specific and framework based approaches SA/AU = (Semi) automated, NLC= New language concepts, MM= Multiperspective modelling, ER= Extension required, ID=Iterative development	Input/Output		Abstr.		Auto.		Compl.				Usab.	
	Source	target	HT	VT	SA	AU	NLC	Patterns	MM	Adaptab.	ER	ID
[22] Mantell 2003	UML profile	BPEL	×			×	+	−	+	±	−	−
[23] Bordbar/Staikopoulos 2004	UML profile	BPEL	×		×		+	−	+	±	−	−
[24] Skogan et al. 2004	UML profile	BPEL	×			×	+	−	+	±	−	−
[25] Heckel/Voigt 2004	UML profile	BPEL	×			×	+	−	+	±	−	+
[26] Yu et al. 2007	UML CCA	BPEL	×			×	+	−	+	±	−	−
[27] Schmelzle 2007	annot. EPC	BPEL		×	×		+	−	−	−	−	−
[28] Allweyer 2007	annot. EPC	BPEL		×	×		−	+	+	+	+	−
[29] Zdun/Dustdar 2006	DSL	BPEL		×	n/a		−	+	+	+	+	−
[30] Roser et al.	DSL	BPEL		×		×	+	−	−	+	+	−

Table 1(b) summarises domain specific and framework based approaches. Most of them extend control flow centred process transformations to holistic approaches in terms of usability and code generation capabilities. The described UML-based transformations use the UML language extension mechanisms to introduce new technical concepts. The level of abstraction between the UML source and the BPEL target are rather close to one another. In many cases this is due to the motivation of introducing a modelling notation for BPEL ([25, see]). Other approaches aim at complexity reduction in business process modelling by providing a design method, which is aligned to business needs. They increase the level of abstraction by introducing business related concepts expressed as new language concepts or patterns, which enable the re-use of recurring problem solutions. The cognitive complexity is also reduced by multi perspective modelling. Depending on the considered transformation input, some approaches offer a completely automated transformation process. Adaptability is mainly addressed by framework based approaches. A practical usage of them even requires extensions. In general, the need for incremental development is basically not considered in all reviewed works. Only one approach (the horizontal UML to BPEL transformation given in [25]) addresses this issue by taking advantage of bi-directionality unambiguous pair grammar.

3 Business to IT Transformation Framework

3.1 Axiom

Most of the approaches discussed in Sect. 2 follow a horizontal transformation strategy. Source models are translated into another format without further refinements. A horizontal transformation starting with an abstract business process model results in an abstract orchestration model requiring significant refinement efforts to make it executable. An alternative is to start from a business process model augmented with technical details. This strategy is addressed by several research approaches and can be found in commercial products, e. g. Intalio BPMS[4] and Lombardi Blueprint[5].

The horizontal strategy is adequate for providing a modelling language that is to some extend independent of the orchestration language because minor changes in the execution platform may be reflected in the transformation definition. According to our industrial experience and industrial case studies (e. g. [34, 35, 36]) a rather tight relationship between business process model and its implementation causes problems in heterogeneous IT landscapes with different middlewares from SAP, Oracle, IBM, and Microsoft at the same time. Furthermore, the requirement that the source model contains all necessary information for complete code generation forces the business analyst to think in terms of executable business processes.

Taking those arguments into account, we formulate the following axiom, which forms the basic assumption for the business to IT transformation framework: *Business process models (e. g. BPMN, EPC) must be platform independent and a platform specific IT implementation (e. g. BPEL) must be derived through a vertical transformation strategy.*

[4] http://www.intalio.com/
[5] http://www.lombardisoftware.com/

3.2 Axiom's Consequences and Requirements from the Field

If the axiom described in the previous subsection is accepted as basic assumption for business to IT transformations, *a business process model shall not contain any platform specific details*, e. g. references to WSDL artefacts or SOAP descriptions. Instead, the business process model should represent the execution related perspectives on an abstract business level in terms of refineable proxy elements. From a single business process model several orchestration models can be derived following different implementation paradigms. This clear separation allows having a customised view on the business process – one view for business experts and one view for implementation experts like software engineers.

Besides those methodical consequences, there are also additional requirements from the field. According to our experience it is *feasible to restrict the expressiveness of the source model language* to a subset, which can be unambiguously transformed to the executable model language. Obviously, this restriction should at least allow well-structured input models including cycles. Instead of supporting exotic control flow transformations, the business analyst should be guided according to clear modelling guidelines by validation functionality provided by the modelling tool.

Due to the abandoned full code generation property the *generated executable model should be comprehensible for human users* for further refinements. In case of BPEL, only block elements should be used ignoring BPEL's flow character. This is also demanded by [10, 13].

From our point of view, without addressing the evolutionary aspect of business process and orchestration models the value of a business to IT transformation approach is only limited. Following the axiom a transformation approach must provide *change detection, change visualisation, and merge functionalities* of concurrent changes in the corresponding models.

3.3 Framework

Based on the requirements stated in the previous section and the axiom's consequences, we can now formulate a general framework for business to IT transformations (see Fig. 2). The framework refers to different modelling perspectives – process, data, and interaction perspective – and thus reduces cognitive complexity at design time. Furthermore, Fig. 2 exemplifies some entities on two different levels of abstraction as well as their relationships, which are operationalised by a vertical transformation. Manual refinements in iterative development cycles are explicitly addressed by perspective-specific merge functionalities.

The business to IT transformation framework does not designate a certain way of how the transformation itself is implemented. Ideally, to enhance adaptability and usability the transformations of the different perspectives are defined in a modular, declarative, and fully automated manner. Instead, the framework highlights that the intended transformation support does not just consist of a control flow transformation, but that also data and interaction related aspects must be transformed. For example, a business process usually has a notion of

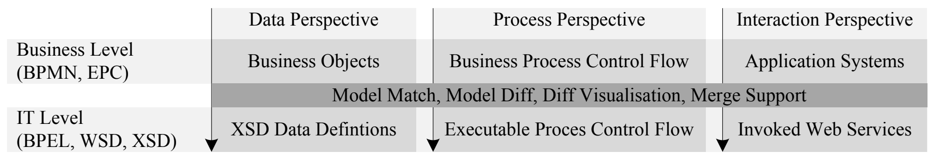

Data Perspective
Process Perspective
Interaction Perspective
Business Level (BPMN, EPC)
Business Objects
Business Process Control Flow
Application Systems
Model Match, Model Diff, Diff Visualisation, Merge Support
IT Level (BPEL, WSD, XSD)
XSD Data Defintions
Executable Proces Control Flow
Invoked Web Services

Fig. 2. Business to IT transformation framework

data like business objects consumed and produced by business functions. Those business objects must be present in the generated BPEL model as well, e. g. as input and output variables.

The framework also does not propose a particular way to implement the merge of different model versions. At least, it should consist of some kind of model matching, change detection, change visualisation, and merge support. Implementation details, e. g. the type of merge (2-way, 3-way, 4-way), type of match (identity-based, heuristic) or time of synchronisation, depend on the requirements of the specific usage scenario.

4 Summary

Business to IT transformations are an important approach to bridge the gap between business and IT. In this paper, we first investigated previous work done on this topic. We classified the existing literature according to different criteria like horizontal and vertical model transformations, transformation algorithms, transformation frameworks, support for incremental development, etc. The literature review was done for works dealing with BPMN, EPC or UML activity diagrams as business process modelling language. In all cases, the reviewed works transform those source models to BPEL. The literature review showed that the problem of business to IT transformation is not solved, because no single existing transformation or transformation framework supports all important criteria.

We therefore derived a new business to IT transformation framework to fill this gap. This framework is based on the assumptions that the business process model should stay untouched in case the implementation technology changes. We motivated this assumption based on existing case studies. To evaluate and demonstrate the feasibility of our framework, we will implement a new EPC to BPEL transformation in a future research activity.

References

1. Dehnert, J., van der Aalst, W.: Bridging gap between business models and workflow specifications. Intl J of Cooperative Information Systems **13**(3) (2004) 289–332
2. Koehler, J., Hauser, R., Sendall, S., Wahler, M.: Declarative techniques for model-driven business process integration. IBM Systems Journal **44**(1) (2005) 47–65
3. OMG: Business process modeling notation (bpmn) specification. Technical report, Object Management Group (OMG) (February 2006)

4. Scheer, A.W., Thomas, O., Adam, O.: Process modelling using event-driven process chains. In Dumas, M., van der Aalst, W., ter Hofstede, A., eds.: Process-Aware Information Systems. Wiley, Hoboken, New Jersey, USA (2005) 119–146
5. Mens, T., Czarnecki, K., Van Gorp, P.: A taxonomy of model transformations. In Bezivin, J., Heckel, R., eds.: Language Engineering for Model-Driven Software Development. Number 04101 in Dagstuhl Seminar Proceedings, Germany (2005)
6. Wehler, J.: Boolean and free-choice semantics of event-driven process chains. In Nüttgens, M., Rump, F.J., Gadatsch, A., eds.: EPK 2007 – Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Volume 303 of CEUR Workshop Proc. (2007)
7. Nüttgens, M., Rump, F.J.: Syntax und semantik ereignisgesteuerter prozessketten (epk). In: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen (Promise), Potsdam, Germany (2002)
8. van der Aalst, W.: Formalization and verification of event-driven process chains. Information and Software Technology **41**(10) (1999) 639–650
9. Kindler, E.: On the semantics of epcs: Resolving the vicious circle. Data Knowl. Eng. **56**(1) (2006) 23–40
10. Mendling, J., Lassen, K.B., Zdun, U.: Transformation strategies between block-oriented and graph-oriented process modelling languages. In Lehner, F., Nösekabel, H., Kleinschmidt, P., eds.: Multikonferenz Wirtschaftsinformatik (MKWI). Volume 2., Passau, Germany, GITO-Verlag Berlin (2006) 297–312
11. Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A.: Pattern-based analysis of bpel4ws. Technical Report FIT-TR-2002-04, Queensland University of Technology, Brisbane, Australia (2002)
12. Hauser, R., Koehler, J.: Compiling process graphs into executable code. In Karsai, G., Visser, E., eds.: 3rd Intl Conf on Generative Programming and Component Engineering (GPCE). Volume 3286 of LNCS., Vancouver, Canada (2004) 317–336
13. van der Aalst, W., Lassen, K.B.: Translating workflow nets to bpel4ws. Technical Report BPM-05-16, BPM Center, Eindhoven, Netherlands (2005)
14. Ziemann, J., Mendling, J.: Epc-based modelling of bpel processes: a pragmatic transformation approach. In: 7th Intl Conf on Modern Information Technology in the Innovation Processes of the Industrial Enterprises (MITIP), Italy (2005)
15. Kopp, O., Unger, T., Leymann, F.: Nautilus event-driven process chains: Syntax, semantics, and their mapping to bpel. In Nüttgens, M., Rump, F.J., Mendling, J., eds.: EPK 2006 – Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Volume 224 of CEUR Workshop Proc. (2006) 85–104
16. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische prozessmodellierung auf der grundlage ereignisgesteuerter prozessketten (epk). Technical Report Heft 89, Universität des Saarlandes, Saarbrücken, Germany (1992)
17. Specht, T., Drawehn, J., Thränert, M., Kühne, S.: Modeling Cooperative Business Processes and Transformation to a Service Oriented Architecture. In: 7th Intl IEEE Conf on E-Commerce Technology, IEEE CS (2005) 249–256
18. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(3) (July 2003) 5–51
19. White, S.A.: Using bpmn to model a bpel process. Technical report, BPTrends (March 2005) http://www.bptrends.com/.
20. Ouyang, C., Dumas, M., Breutel, S., ter Hofstede, A.: Translating standard process models to bpel. In: 18th Intl Conf on Advanced Information Systems Engineering (CAiSE), Luxembourg (2006)
21. Ouyang, C., van der Aalst, W., Dumas, M., ter Hofstede, A.: Translating bpmn to bpel. Technical Report BPM-06-02, BPM Center, Netherlands (2006)

22. Mantell, K.: From uml to bpel: Model driven architecture in a web services world. Technical report, IBM developerWorks (September 2003)
23. Bordbar, B., Staikopoulos, A.: On behavioural model transformation in web services. In Wang, S., Tanaka, K., Zhou, S., Ling, T.W., Guan, J., Yang, D.Q., Grandi, F., Mangina, E., Song, I.Y., Mayr, H., eds.: eCOMO – E-Business Processes and Infrastructure. Volume 3289 of LNCS. (2004)
24. Skogan, D., Grønmo, R., Solheim, I.: Web service composition in uml. In: 8th IEEE Intl Enterprise Distributed Object Computing Conf (EDOC). (2004)
25. Heckel, R., Voigt, H.: Model-based development of executable business processes for web services. In: Lectures on Concurrency and Petri Nets. Volume 3098 of LNCS. (2004) 559–584
26. Yu, X., Zhang, Y., Zhang, T., Wang, L., Zhao, J., Zheng, G., Li, X.: Towards a model driven approach to automatic bpel generation. In Akehurst, D.H., Vogel, R., Paige, R.F., eds.: 3rd European Conf on Model Driven Architecture-Foundations and Applications (ECMDA). Volume 4530 of LNCS., Haifa, Israel (2007) 204–218
27. Schmelzle, O.: Transformation von annotierten geschäftsprozessen nach bpel. Master's thesis, Gottfried Wilhelm Leibniz Universität Hannover, Germany (2007)
28. Allweyer, T.: Erzeugung detaillierter und ausführbarer geschäftsprozessmodelle durch modell-zu-modell-transformationen. In Nüttgens, M., Rump, F.J., Gadatsch, A., eds.: EPK 2007 – Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Volume 303 of CEUR Workshop Proc. (2007)
29. Zdun, U., Dustdar, S.: Model-driven and pattern-based integration of process-driven soa models. In Leymann, F., Reisig, W., Thatte, S.R., van der Aalst, W., eds.: The Role of Business Processes in Service Oriented Architectures. Number 06291 in Dagstuhl Seminar Proceedings, Germany (2006)
30. Roser, S., Lautenbacher, F., Bauer, B.: Generation of workflow code from dsms. In Sprinkle, J., Gray, J., Rossi, M., Tolvanen, J.P., eds.: 7th OOPSLA Workshop on Domain-Specific Modeling (DSM). Number TR-38 in Computer Science and Information System Reports, Jyväskylä, Finland (2007) 149–159
31. Kiepuszewski, B., ter Hofstede, A., van der Aalst, W.: Fundamentals of control flow in workflows. Acta Informatica **39**(3) (2003) 143–209
32. Czarnecki, K., Eisenecker, U.W.: Generative Programming : Methods, Tools, and Applications. Addison-Wesley, Boston et. al. (2000)
33. Greenfield, J., Short, K.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley Publishing, Inc. (2004)
34. Stein, S., Kühne, S., Drawehn, J., Feja, S., Rotzoll, W.: Evaluation of orvia framework for model-driven soa implementations: An industrial case study. In: 6th Intl Conf on Business Process Management (BPM), Milan, Italy (September 2–4 2008)
35. Allweyer, T.: Vom fachlichen modell zum ausführbaren workflow: Am beispiel von aris und intalio bpms. Technical report, University of Applied Sciences Kaiserslautern, Germany (February 2008) http://kurze-prozesse.de/blog/wp-content/uploads/2008/02/Vom-fachlichen-Modell-zum-Workflow.pdf.
36. Moertl, H., Groß, H.J., Herrmann, M.: Prototypische implementierung eines bestellanforderungsprozesses nach den prinzipien einer service-orientierten architektur (soa) bei der daimler ag. Technical report, Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V. (BITKOM), Berlin, Germany (2007) http://www.bitkom-hochschulwettbewerb.org/.

A Model-Driven Approach to Implementing Coordination Protocols in BPEL

Oliver Kopp[1], Branimir Wetzstein[1], Ralph Mietzner[1], Stefan Pottinger[2], Dimka Karastoyanova[1], and Frank Leymann[1]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Germany
[2] IPL Information Processing Ltd, United Kingdom

Abstract. WS-Coordination defines a framework for establishing protocols for coordinating the outcome agreement within distributed applications. The framework is extensible and allows support for multiple coordination protocols. To facilitate the realization of new coordination protocols we present a model-driven approach for the generation of BPEL processes used as implementation of coordination protocols. We show how coordination protocols can be modeled in domain-specific graph-based diagrams and how to transform such graphs into abstract BPEL process models representing the behavior of the coordinator and the participants in the protocol.

1 Introduction

Web services are the most recent middleware technology for application integration within and across enterprises [1]. Through the use of standards like SOAP, WSDL, UDDI, and the Web Services Business Process Execution Language (BPEL, [2]) the Web service technology enables interoperable service interactions in heterogeneous environments. Coordination is an important mechanism used in distributed computations with multiple participants that must jointly agree on the outcome of the computation. A well-known example for the use of coordination are distributed transactions using atomic commitment protocols to agree on the success or failure of a transaction [3]. The aspect of coordination in the domain of Web services is addressed by WS-Coordination [4]: it defines an extensible framework for coordinating the outcome of a set of Web services contributing to a distributed computation using a generalized notion of a coordinator and the so-called coordination protocols. In the context of WS-Coordination, coordination protocols describe the messages exchanged between the coordinator and the participants of a distributed computation and thus realize a one-to-many coordination. Two types of protocols (aka coordination types) have already been defined to cover "traditional" atomic transactions (WS-AtomicTransaction [5]) and long-running business transactions (WS-BusinessActivity [6]). However, the use of WS-Coordination is not restricted to transaction processing systems only. Other types of coordination protocols have also been defined for distributed computations such as protocols describing auctions [7], protocols for split BPEL loops and split BPEL scopes [8] and protocols for externalizing the coordination of BPEL scopes as a whole [9].

Coordination protocols can be quite complex. The coordinator has to deal with a variable number of participants. Each participant is in a well-defined state that potentially differs from the state of another participant at the same time. The implementation of a coordination protocol is difficult and error-prone. To simplify and accelerate the implementation, and eliminate errors, in this paper we propose a model-driven architecture (MDA)approach: The protocol is first modeled as a state-based graph, which we call coordination protocol graph (CPG). A CPG captures the different states and state changes based on the messages exchanged between coordinator and participant. The graph diagram is the domain specific language (DSL) we use for specifying coordination protocols. It contains only those elements which are needed for coordination protocol modeling and is therefore well suited for protocol designers. In MDA terms a coordination protocol graph specifies a Platform Independent Model (PIM) [10]. The CPG is independent of any hardware or programming platform.

We have decided to represent the Platform Specific Model (PSM) in terms of BPEL since, in general, coordination protocols define a sequence of steps and messages to be exchanged between participants in a coordinated interaction, timing issues, and how exceptional situations must be tackled. In that respect, modeling coordination protocols is similar to modeling business processes. In this work we generate abstract BPEL processes for both the coordinator and the participant roles in coordination protocols. These BPEL process models capture the essential parts of the message exchange between the parties and the resulting protocol state changes. The generated code reduces the need for tedious and error-prone programming concerning the communication between the coordinator and participants in the protocol. Additional protocol logic, which cannot be captured in the CPG, has to be manually added by the programmer.

The rest of the paper is organized as follows: Section 2 gives an overview of BPEL and WS-Coordination. In Section 3 we present the syntax and semantics of the coordination protocol graph (CPG). After depicting our model-driven approach in Section 4, we describe the generation of the BPEL process models in Section 5. We finalize with the discussion of related work, conclusion and future work.

2 Background

WS-Coordination [4] defines an extensible framework for coordinating interactions between Web services. Coordinated interactions are called (coordinated) activities in the context of WS-Coordination. The framework enables participants to reach agreement on the outcome of distributed activities using a coordinator and an extensible set of coordination protocols. The framework defines three services a coordinator has to provide: activation service, registration service, and protocol services. When an application, in the role of an initiator, wants to start a coordinated activity, it requests a coordination context from an activation service.

The coordination context contains an activity identifier, the coordination type (e.g. atomic transaction) as requested by the initiator, and the endpoint reference of the registration service. When the initiator distributes work, it sends the coordination context with the application message to the participant. Before starting work, the participant registers at the registration service of the coordinator. At some later point the protocol service, which coordinates the outcome according to the specific protocol of the coordination type, is started.

While the logic of the activation and registration service are fixed, the framework allows the definition of arbitrary coordination types as well as their implemenation by means of different protocol services. In the following when referring to "coordinator" and "participant", we mean the protocol service implementations at the coordinator and participant, respectively.

The *Web Services Business Process Execution Language* (BPEL) is an orchestration language for Web services. A BPEL process is a composition of Web services, which are accessed through partner links referencing their WSDL port types. The process is itself exposed as a Web service.

The BPEL process model comprises two types of activities: basic activities cope with invoking other Web services (invoke), providing operations to other Web services (e.g. receive and reply), timing issues and fault handling; structured activities nest other activities and deal with parallel (flow) and sequential execution (sequence), conditional behavior and event processing. Process data is stored in variables, while the assign activity is used for data manipulation. Activities can be enclosed in scopes to denote sets of activities that are to be dealt with as a unit of work. Scopes can be modeled to ensure all-or-nothing behavior, support data scoping, exception handling, compensation, and sophisticated event handling. Instance management is done using correlation sets. Correlation sets define which fields in incoming messages are to be used as identifiers to route the messages to one of possibly several running instances of the same process model.

BPEL processes can be either abstract or executable. An executable BPEL process provides a process model definition with enough information to be interpreted by a BPEL process engine. An abstract BPEL process hides some of the information needed for execution and is associated with a process profile defining restrictions and the indented usage of the abstract process. The profile used in our approach is the abstract process profile for templates. It allows marking sections of the process model as "opaque" using opaque tokens. It is thus explicitly specified which sections of the process model have to be later replaced by concrete activities, expressions etc. to make the process executable.

3 Modeling Coordination Protocols

There is no standard notation for modeling coordination protocols. The specifications in this area use either a proprietary or a generic diagram type (e.g. UML sequence diagram), or a combination of these. For modeling coordination protocols we have adopted the diagram type from the WS-AtomicTransaction (WS-AT) and WS-BusinessActivity (WS-BA) specifications. This diagram type can be seen as a domain specific language for modeling coordination protocols. WS-BA contains two protocols: *WS-BA with Participant Completion*, where the participant signals when it has completed its work and *WS-BA with Coordinator Completion*, where the coordinator notifies the participant when it has to complete his work. Figure 1 shows the WS-BA with Participant Completion protocol as an example, which we will also use in the rest of the paper for illustration of mapping concepts.

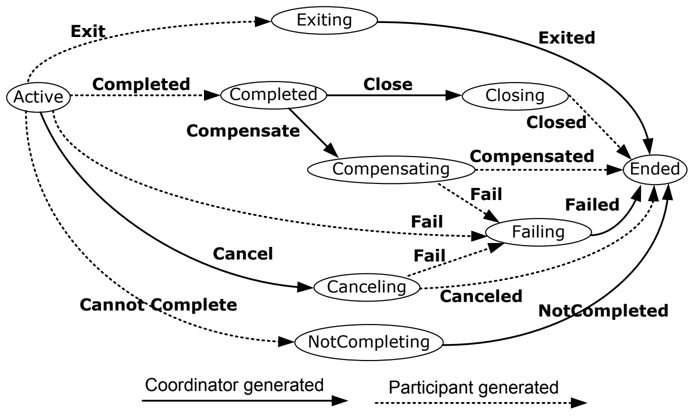


Fig. 1. WS-BA with Participant Completion Protocol [6]

The diagram defines a state-based graph, which we name coordination protocol graph (CPG). A CPG is a directed graph with labeled edges and labeled nodes. The nodes denote the states of the coordination protocol between a coordinator and a participant. The node labels describe the semantics of the states. The edges depict the messages exchanged by the protocol parties; the edge labels describe the semantics of the message. Since messages can be sent by a participant and by a coordinator, the set of all edges is divided into two disjoint sets: edges denoting coordinator messages (solid lines) and edges denoting participant messages (dashed lines). Each CPG has exactly one node with no incoming edges (source) and at least one node without outgoing edges (sink). No two coordinator edges or participant edges with the same label may leave the same node, because this would lead to non-determinism. A CPG does not contain cycles. The conclusion section includes a discussion about cycles in a CPG and the possibilities to support cyclic CPGs. At a certain point in time each participant can be in a different state. For example, one participant can be in the state "Failing" while another is in the state "Closing". Since coordinator usually interacts with more than one participant, the coordinator has to hold the state of each state machine.

Outgoing edges of a CPG denote messages which may be sent and each state denotes the possible state of a participant. The sender of the message (participant or coordinator) transitions to the next state when sending a message. The recipient of the message transitions to the next state when receiving the message. For the period of time when the message is transported, the coordinator and participant thus are in different states. In addition to the obvious behavior of state changes there are three special cases: (i) ignoring same messages which are sent more than once, (ii) precedence of participant messages over coordinator messages, (iii) invalid messages.

If the message leading to a new state is received more than once, it is simply ignored. For example, if the coordinator being in state "Exiting" receives the message "Exit" again, that message is ignored. This case can arise, when messages are resent because it is suspected that the first message hasn't been transmitted successfully.

If a state has both outgoing participant and coordinator messages, then it can happen that the coordinator sends a protocol message and enters the corresponding new state, but later receives a protocol message from the participant which is consistent with the former state. This can happen when both the coordinator and the participant send their messages at about the same time, which leads to different views on the protocol state on coordinator and participant side. In that case the participant messages have precedence over coordinator messages. In Figure 1 the state is "Active" at the beginning of the protocol. Let us assume the coordinator sends "Cancel" to the participant and sets the state to "Canceling". At the same time, however, the participant sends the message "Completed" and changes his state to "Completed". When the coordinator receives the message "Completed" while being in state "Canceling" for the participant, he has to revert to the former state "Active", accept the notification "Completed" and change the state to "Completed". The participant on the other side just discards the coordinator message "Cancel".

Finally, if in a state other messages than the allowed ones are received, a fault message should be generated and sent to the sender of the invalid message. The protocol execution is aborted.

It is important to note, that a CPG captures only the possible interactions and state changes between the coordinator and participant. A CPG does not capture the reason of these state changes. For example, if a participant is in the state "Completed" it can receive either a "Close" or a "Compensate" message from the coordinator. Which of the two messages is sent, is part of the protocol logic. For example, if another participant has failed and all-or-noting semantics is needed a "Compensate" message would be sent. Because not all of the protocol logic is captured by the graph, it has to be additionally implemented after the generation of the BPEL process.

The CPG and its semantics are derived from WS-BA and WS-AT protocols. In summary, the CPG graph captures the exchanged messages between a coordinator and a participant, and the resulting state changes, however not the cause of the state changes.

4 Model-Driven Implementation Approach

For the implementation of coordination protocols we adopt a model-driven approach. Our goal is to model the coordination protocol using a domain-specific language suitable for coordination protocol designers, and then generate BPEL code which implements the coordination protocol.

The DSL, in our case the CPG, is used for creating a platform-independent model (PIM) of the coordination protocol. The PIM can be transformed to platform-specific models (PSM) for different kind of platforms. In this paper we use BPEL and the Web service platform, in particular WS-Coordination as the coordination framework.

As the CPG does not contain enough information to be executed, the additional information has to be added to the PSM after generation. We thus do not achieve 100% BPEL code generation, but still avoid much of tedious and error-prone programming. The use of a model-driven approach ensures higher productivity in development and better quality of the implemented code.

Using BPEL for implementation of coordination protocols has several benefits when compared to a 3GL programming language such as Java. BPEL enables programming on a higher abstraction level which makes the code generation easier. BPEL has native WSDL support needed for interoperability and native support for concurrency, backward and forward recovery. As BPEL supports graph-based models, coordination protocol graphs can be more easily and naturally transformed into BPEL. A BPEL engine persistently stores all events related to process execution in an audit log and thus automatically supports reliable recording of coordination protocol execution out of the box. The audit log enables checking the execution of coordination for compliance with the protocol. A BPEL engine typically also provides a monitoring tool, which enables observing the execution of coordination protocols in real-time. Finally, as the state of a BPEL process instance is persistently saved after each state transition, the coordination protocol can be stopped and resumed at any time using a monitoring tool.

The approach is shown in Figure 2. In the first step, the CPG is created using a corresponding graphical CPG modeling tool. The CPG models the interaction between the coordinator and the participant in a platform-independent way.

In the next step, the CPG is transformed into two abstract BPEL processes, one for the coordinator and one for the participants. Therefore, the abstract BPEL processes and corresponding WSDL definitions are generated. If the WSDL definitions already exist, as for example in the case of the WS-AT and WS-BA specification, then the CPG has to be correspondingly marked. One has to specify the names of the WSDL port types for both the participant and coordinator process, the WSDL message and operation names, which correspond to the labels of the state transitions in the CPG, etc. That ensures that the generated BPEL processes and WSDL descriptions are compliant to the already existing probably standardized ones. The two corresponding WSDL interface descriptions of the processes can be completely generated. Using standardized WSDL interfaces ensures that the coordinator process can be used to coordinate arbitrary protocol participants apart from the generated BPEL-based participants. This is also the

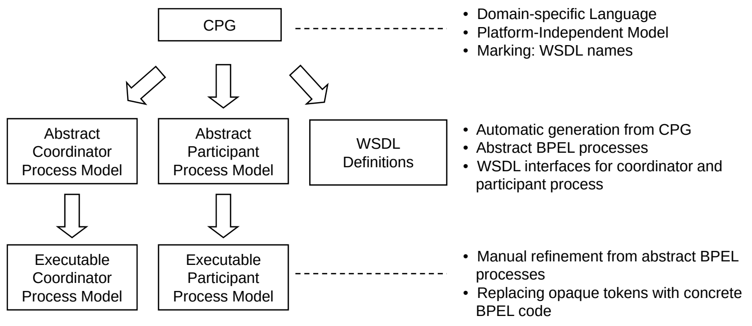

CPG
Domain-specific Language
Platform-Independent Model
Marking: WSDL names
Abstract Coordinator Process Model
Abstract Participant Process Model
WSDL Definitions
Automatic generation from CPG
Abstract BPEL processes
WSDL interfaces for coordinator and participant process
Executable Coordinator Process Model
Executable Participant Process Model
Manual refinement from abstract BPEL processes
Replacing opaque tokens with concrete BPEL code

Fig. 2. Model-Driven Implementation Approach

case for the generated participants, which can be used with another protocol-compliant coordinator. Thus, our approach supports heterogeneous environments.

As discussed in the previous section, the generated process models cannot be executable, because the CPG does not capture the whole protocol logic. The locations in the process model where missing logic has to be added are "marked" in the generated BPEL code using opaque tokens, as defined in the abstract process profile for templates [2]. These opaque tokens show to the developer where additional logic has to be added to make the process executable. The abstract BPEL process profile for observable behavior [2] cannot be used, since it does not allow the addition of interaction activities with existing partner links when replacing opaque activities. However, that is needed in certain cases: For example, in the coordinator process after the interaction activity receiving a "Fail" from one participant, one might want to add interaction activities (BPEL invoke) which send "Cancel" notifications to other participants.

As already described in Section 2, WS-Coordination defines three services a coordinator has to provide: activation, registration, and protocol services. While protocol services can be additionally defined in separate specifications such as WS-BA, the implementation of the activation and registration services stays the same. The activation and registration service of the coordinator can thus be fully generated. Both services in addition to the protocol service are implemented by the coordinator process model (see Section 5).

After generation, the abstract process models are refined manually by a developer who replaces the opaque tokens by the missing coordination protocol logic. The resulting executable BPEL process models can finally be deployed on a BPEL engine.

5 Generating BPEL Process Models

In the following we describe in detail how CPG graphs are transformed to abstract BPEL process models. We generate two abstract BPEL process models, one for the coordinator and one for the participants.

We have chosen different approaches for the generation of the two process models. For the participant process model, we keep the graphical structure of the CPG in the BPEL process model by mapping the CPG graph directly to a BPEL flow. The BPEL flow activity together with BPEL links enables graph-based workflow modeling. The generated BPEL process structure closely resembles the CPG structure and thus increases the readability of the process. The generated BPEL constructs are described in detail in [11].

For the coordinator process model the participant approach is not feasible, since the coordinator holds a different state for each participant. The coordinator cannot leave the scope “Active” until all registered participants have been handled for that scope. In the meantime, however, several participants could have declared that they want to exit the protocol by sending the message “Exit” to the coordinator. In that case the coordinator should immediately send the notification “Exited” to the participant. However, this is not possible, since the coordinator is in the scope “Active” and waits for other participants to complete their work. When the coordinator finally leaves the scope “Active”, a new participant could register for the protocol. Since the scope “Active” has already finished, the new participant cannot be handled. Therefore, we define global event handlers for each message that can be received by the coordinator. That means, we implement a state-machine by specifying rules of the form: if received message x, then perform some logic which handles that message x.

Figure 3 illustrates the pattern for the implementation of the coordinator scope. An instance of the coordinator process model is started when a new WS-Coordination activity is created. This is done by an application by sending a “CreateCoordinationContext” message to the coordinator endpoint which replies with a “CreatedCoordinationContext” message to indicate successful creation of the context.

Having received such a message the coordinator process is now ready to accept registration messages from participants that wish to participate in the coordination, and to react on messages sent by participants that have already registered. The coordinator leaves this state if the application determines that the coordination should end and sends a corresponding message.

The abstract BPEL process template for the coordinator is generated as follows: In order to manage the participants for the activity an array is generated that holds the status of all participants of the activity as well as the endpoint references of the participants. The endpoint references are obtained during registration and are needed to send coordination messages to the right endpoints.

Regarding the control flow, at first a process instance creating receive activity is added that is triggered by WS-Coordination “CreateCoordinationContext” messages. The user can then replace the following opaque activity by inserting arbitrary BPEL activities that handle the message. Afterwards the confirmation for the successful creation of the coordination context is sent. The control-flow now enters the scope that handles the coordination protocol specific messages as well as the registration of participants for the activity. Both types of messages are received and handled via event handlers.

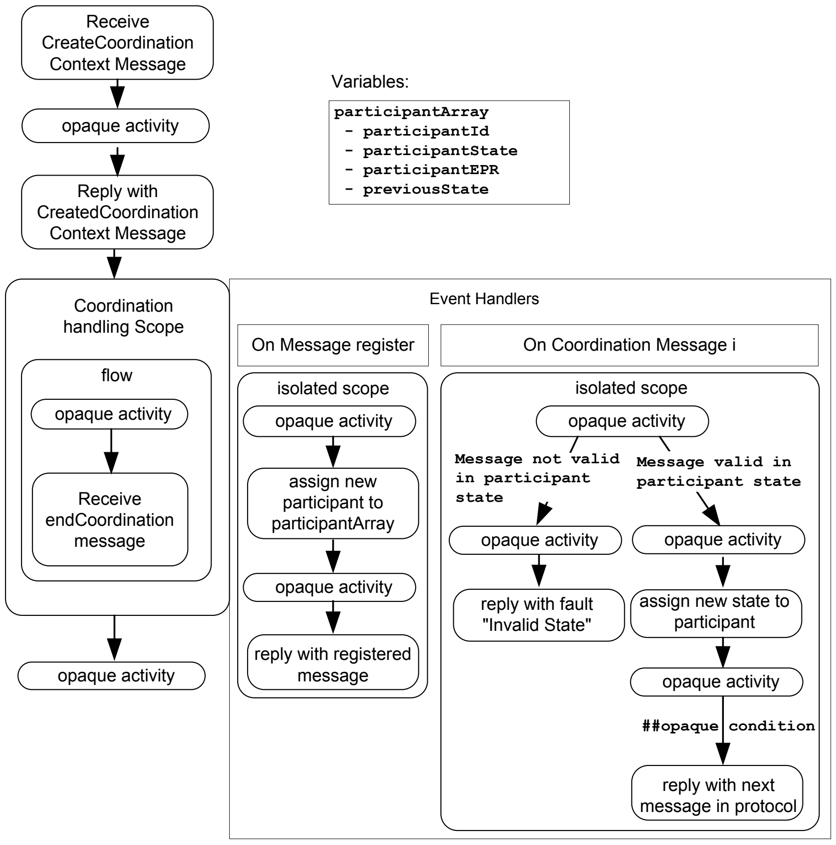

Receive CreateCoordination Context Message
opaque activity
Reply with CreatedCoordination Context Message
Variables:
participantArray
- participantId
- participantState
- participantEPR
- previousState
Coordination handling Scope
flow
opaque activity
Receive endCoordination message
opaque activity
Event Handlers
On Message register
isolated scope
opaque activity
assign new participant to participantArray
opaque activity
reply with registered message
On Coordination Message i
isolated scope
opaque activity
Message not valid in participant state
Message valid in participant state
opaque activity
reply with fault "Invalid State"
opaque activity
assign new state to participant
opaque activity
##opaque condition
reply with next message in protocol

Fig. 3. Pattern for the generation of coordinator scopes

We place opaque activities throughout the process template during generation to allow the actual coordination logic to be inserted as needed. We do not explicitly mention those in the following discussion, but Figure 3 shows where the opaque activities are placed in detail. In the following description we concentrate on the control flow and leave out details such as correlation of messages to the right process instance. For now, we assume that upon reception of each message the coordinator knows which participant has sent the message and that messages only are received by coordinator process instances that handle the participant that has sent the message. Means to ensure these assumptions are presented in [11].

Registration of Participants As shown in Figure 3 registration of participants is handled via a dedicated event handler. The event handler includes an assign activity that adds the new participant into the participant array and sets its current state to the first state that follows registration in the coordination protocol the coordinator has been created for. Afterwards the event handler responds with a “Registered” message. Both “Register” and “Registered” messages are defined in WS-Coordination. Opaque activities allow the handling of special cases by special coordination logic. Such a case may be the reception of registration messages after other participants have already faulted or completed. For example, WS-BA demands that such cases are allowed.

Handling of Protocol Specific Messages For each message the coordinator can receive (i.e., for each dashed line in a CPG) a separate event handler is created that handles that type of message. Upon receipt of a participant message, one out of two paths can be followed: The first path is followed if the message is not allowed in the state of the participant. In that case the “Invalid State” message is sent back. In case the message is allowed in the current state, the state of the participant is updated via an assign activity. The generated model contains opaque activities that can be replaced by arbitrary BPEL activities that perform the actual coordinator logic. For example, one or more invoke activities can be inserted that send the corresponding messages that follow the received message in the coordination protocol. However, the decision on whether a message is sent and which message in particular is sent depends on the actual coordinator logic and therefore is marked as opaque and needs to be completed during the customization of the template.

The second path also handles two special cases: (i) ignoring messages which were resent by the participant, (ii) reverting to a previous state. Both the WS-AtomicTransaction and the WS-BusinessActivity specification demand that not only messages that are allowed in the current state of the participant are allowed but also messages corresponding to the previous state of the participant. In order to comply with this demand an additional field in the array is introduced that stores the previous state. On reception of a message a new assign activity is introduced that reverts the state of a participant if a message corresponding to that state is received. Then the control flow can proceed as if it had originally received the message in the correct state.

Concurrent Reception of Messages All messages that can be received concurrently by the coordinator are handled by event handlers. Thus, we ensure that the BPEL engine can deal with the concurrent message reception. However in order to ensure that concurrent access to shared variables, such as the participant arrays, and resulting problems are avoided the logic of the event handlers is placed in isolated scopes. An isolated scope is a BPEL means to synchronize parallel access to variables.

6 Related Work

There are several approaches to map business processes modeled graphically to BPEL (e.g. [12, 13]). The approaches are similar to our work, since they are also generating BPEL processes, but the authors deal with generating a single BPEL process: they focus on orchestrations only. Hence, these approaches do not tackle the communication between processes as it is the case between the coordinator and the participant process.

In contrast to orchestrations, choreographies provide a global view on the interactions of all participants involved. In a coordination, the set of participants is unknown in advance. All choreography languages targeted to Web service technology either do not support modeling of a-priori unknown number of participants (WS-CDL [14, 15]) or do not support modeling the assignment of a participant to a different set (BPMN [16] and BPEL4Chor [17], Let's Dance [18]).

Another approach to model transactions is the UN/CEFACT's Modeling Methodology (UMM, [19]). While UMM can be mapped to BPEL [20], UMM does not support modeling of sets of a-priori unknown participants.

7 Conclusions and Future Work

The main contributions of this paper are: (i) the introduction of a model-driven approach for implementing coordination protocols, (ii) the concrete transformation of the CPG graph to abstract BPEL process models.

We have shown how a WS-Coordination-based coordination protocol can be modeled as a CPG graph. A CPG graph captures the essence of a coordination protocol: the states of the protocol and messages produced by both the coordinator and the participant. The generated BPEL processes are abstract and comply with the abstract process profile for templates. Opaque activities and expressions mark the locations where the programmer can include additional protocol logic not captured by the CPG to make the processes executable.

We demanded CPGs to be acyclic, since BPEL supports structured loops only. While this works for the protocols described in WS-AtomicTransaction and WS-BusinessActivity, there are coordination protocols such as the protocol for split loops [8]. We used an event handler approach for the coordinator to deal with the different states of each participant, which enables support for loops, too. For the participant model, we generated a BPEL process where the structure of the process directly reflects the structure of the CPG. Basically, when mapping CPGs with structured loops to BPEL, these loops can be captured using BPEL loop constructs. The current mapping style to participant processes does not support loops, since the BPEL flow activity only supports acyclic graphs. When mapping unstructured loops to a BPEL process there are two general approaches: (i) mirror the semantics using event handlers and (ii) untangle the loop by duplication of the activities [21]. The event handler approach is similar to the presented realization of the coordinator. However, the approach has the drawback that the control-flow is captured using event-action rules and not the "usual" BPEL constructs to model the main path of execution. The second approach uses the BPEL flow activity but duplicates the activities. This duplication can be avoided if sub-processes (as defined in BPEL-SPE [22]) are used: instead of mapping each activity directly, each original activity is mapped to a sub-process call. In addition, for each original activity, a separate sub-process is generated. The details of the transition conditions, data passing to and from the sub-process are open issues. Our future work is to evaluate the two possibilities in depth and to realize the more suitable one.

8 Acknowledgements

The research leading to these results has received partial funding from the European Community's 7th Framework Programme under the Network of Excellence S-Cube - Grant Agreement no. 215483 and the German Federal Ministry of Education and Research (project number 01ISE08B).

References

1. Curbera, F., et al.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR
2. OASIS: Web Services Business Process Execution Language Version 2.0
3. Jim Gray, A.R.: Transaction Processing: concepts and techniques. Morgan Kaufman
4. OASIS: Web Services Coordination. Version 1.1
5. OASIS: Web Services Atomic Transaction. Version 1.1
6. OASIS: Web Services Business Activity Framework. Version 1.1
7. Leymann, F., Pottinger, S.: Rethinking the Coordination Models of WS-Coordination and WS-CF. In: ECOWS 2005
8. Khalaf, R., Leymann, F.: Coordination protocols for split BPEL loops and scopes. Technical Report Computer Science 2007/01, University of Stuttgart
9. Pottinger, S., et al.: Coordinate BPEL Scopes and Processes by Extending the WS-Business Activity Framework. In: CoopIS 2007
10. Frankel, D.S.: Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley (2003)
11. Kopp, O., et al.: A model-driven approach to implementing coordination protocols in BPEL. Technical Report Computer Science 2008/02, University of Stuttgart
12. Mendling, J., Lassen, K.B., Zdun, U.: Transformation strategies between block-oriented and graph-oriented process modelling languages. In: XML4BPM 2006
13. Ouyang, C., et al.: Translating standard process models to BPEL. In: Advanced Information Systems Engineering. Volume 4001., Springer Berlin / Heidelberg
14. Kavantzas, N., et al.: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation
15. Decker, G., et al.: On the Suitability of WS-CDL for Choreography Modeling. In: EMISA 2006
16. Object Management Group: Business Process Modeling Notation, V1.1
17. Decker, G., et al.: BPEL4Chor: extending BPEL for modeling choreographies. In: ICWS 2007
18. Zaha, J.M., et al.: A Language for Service Behavior Modeling. In: CoopIS 2006
19. UN/CEFACT: UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module. `http://www.unece.org/cefact/umm/UMM_Foundation_Module.pdf`.
20. Hofreiter, B., et al.: Deriving executable BPEL from UMM business transactions. In: SCC 2007
21. Zhao, W., et al.: Compiling business processes: untangling unstructured loops in irreducible flow graphs. International Journal of Web and Grid Services **2** (2006)
22. IBM, SAP: WS-BPEL Extension for Sub-processes – BPEL-SPE. (2005)

Towards Transformations from BPMN to Heterogeneous Systems

Tobias Küster and Axel Heßler

DAI-Labor, Technische Universität Berlin
Faculty of Electrical Engineering and Computer Science
{tobias.kuester|axel.hessler}@dai-labor.de

Abstract. By now, the mapping from BPMN to BPEL has been implemented in numerous tools, greatly assisting the business architect in the creation of BPEL processes. However, most of these tools are tailored especially for this transformation, neglecting the original purpose of BPMN: Providing a language independent process model. To address this shortcoming, a pure BPMN editor is needed, being dynamically extensible with several export features and added editing functionality. In this paper, we present a tool that follows this approach, not only providing a compelling transformation to BPEL but at the same time being extensible to other languages.

Key words: BPMN, Process Design Tools, Model Driven Development, Transformations, Multi-agent Systems

1 Introduction

The goal of process modelling, as of Model Driven Engineering in general, is to provide an abstract view on systems, and to design those systems in a language and platform independent way. For that purpose the Business Process Modelling Notation (BPMN) [10] has been standardised by the Object Management Group. It can be understood intuitively by all business partners, even those who have great knowledge in their domain but do not know too much about Service Oriented Architecture (SOA) or programming in general. At the same time, BPMN is formal enough to provide a basis for the later implementation and refinement of the business process. Given a respective mapping, a BPMN diagram can be used for generating readily executable code from it. A brief introduction to BPMN is given for instance in [16].

Today, the Business Process Modelling Notation and the specified mapping to the Business Process Execution Language (BPEL) are supported by a growing number of tools — we will have a closer look on some representatives later in Section 4. However, the problem with the majority of existing tools is that while they do provide the usual transformations from BPMN to BPEL, they are focused only on this one aspect of BPMN. Often the editors and even the underlying metamodels are adapted to BPEL in many ways. While this may be desired in order to provide highest possible usability and to support the user in the creation of executable BPEL code, the consequence is that business process diagrams created with these tools can neither be transformed to other executable languages, nor can the process model be used with other tools that might provide different transformations. Thus, while process modelling and BPMN should be independent of a specific executable language, the *tools* are not.

The solution to this problem is to keep both the underlying BPMN metamodel and the diagram editor free from influences from the BPEL world and to use pure BPMN instead, so that diagrams created with such a tool will be truly independent of any concrete language — apart from what influenced the BPMN specification in the first place. Based on this, several mappings to different target languages can be implemented and integrated into the editor as plugins, which may also contribute to the editor in order to support the business architect with language-specific support.

Following this approach, the *Visual Service Design Tool* (VSDT) has been implemented as an Eclipse plugin, inherently providing the necessary modularity, as we will see in Section 2. For the export of BPMN diagrams to executable languages a transformation framework has been designed, which we will describe in more detail in Section 3. The transformation has been subdivided in distinct stages, so that significant parts of it are reusable, e.g. the challenging transformation of the control flow. Thus the actual mapping to a given language can be integrated in a very straight-forward way. While the usual mapping from BPMN to BPEL has been realised as a proof of concept (see Section 3.2), the main intent behind the VSDT is to provide a transformation from business processes to multi-agent systems such as the JIAC language family [14]. The respective mappings are currently under development and will be discussed briefly in Section 3.4. Our ultimate goal is to provide transformations not only in different, but also in heterogeneous systems — just like they are used in the real business world. Future work in order to achieve this goal, as well as a conclusion to this paper, will be discussed in Section 5.

2 The Visual Service Design Tool

The first version of the VSDT has been developed as a diploma thesis [7] in the course of the *Service Centric Home* (SerCHo) project at TU Berlin in early 2007. As the work continued it matured to a feature-rich BPMN editor with an extensible transformation framework and has already been used in a number of service orchestration scenarios in the context of a smart home environment, one of which will be shown later in Section 3.3.

2.1 The Metamodel

The BPMN specification [10] describes in detail how the several nodes and connections constituting a BPMN diagram have to look, in which context they may be used and what attributes they have to provide. However, it does neither give a formal definition of the syntax to be used for the metamodel, nor an interchange format, e.g. using an XML Schema Definition (XSD). Thus the editor's metamodel had to be derived from the informal descriptions in the specification. As it was our main concern to keep as close to the specification as possible, we decided not to reuse the existing Eclipse STP BPMN Editor, which uses a simplified model of BPMN.[1] Instead, almost every attribute and each constraint given in the specification has been incorporated into the metamodel, allowing the creation of any legal business process diagram. Still, some attributes have not been adopted in the metamodel: For instance the possibility to model nested or even crossing Lanes has been dropped, as it turned out that this feature seems to be virtually never used in practical business process design. Further, redundant attributes, such as the Gateway's `defaultGate` attribute, are emulated using getter methods to prevent inconsistency in the diagram model.

Concerning the transformation to BPEL and other executable languages, which in most cases are block-oriented, an extension to the usual BPMN metamodel has been designed, featuring equivalents to the basic block structures, such as sequences, decisions, parallel blocks and loops. These elements are described in a separate metamodel, extending the editor's metamodel. They are used only during the transformation process, especially for the mapping of the structure, as we will see in Section 3.

2.2 The BPMN Editor

Like many others, the VSDT editor has been created using the Eclipse Graphical Modelling Framework (GMF), automatically equipping the editor with numerous features, such as support for the Eclipse properties, outline and problem view and unlimited undo and redo, just to name a few. Being embedded in the Eclipse workbench, the editor is easy to use while at the same time providing a powerful tool for professional business architects and service developers.

While GMF provides a solid basis for the editor, several customisations have been made to the code, further improving the editor's overall usability and supporting the creation of new business processes. For example, the generated property tables have been supplemented with custom-made sheets, in which the several attributes are more clearly arranged. For managing the non-visual elements given in the BPMN specification, such as Properties, Messages and Assignments, a number of clear and uniform dialogs has been created. The various constraints given in the specification were translated to several audit constraints used to validate a given business process diagram. A screenshot showing some of the editor's features can be seen in Figure 1.

As already mentioned, the VSDT was designed to be a pure BPMN editor and independent of BPEL, so the business process diagrams can be transformed to other languages, too, given the respective export plugins. Of course, the downside of this approach is that the editor lacks built-in support for BPEL, e.g. the editor itself does not validate an expression given in the diagram to conform to the

[1] `http://www.eclipse.org/stp/bpmn/`

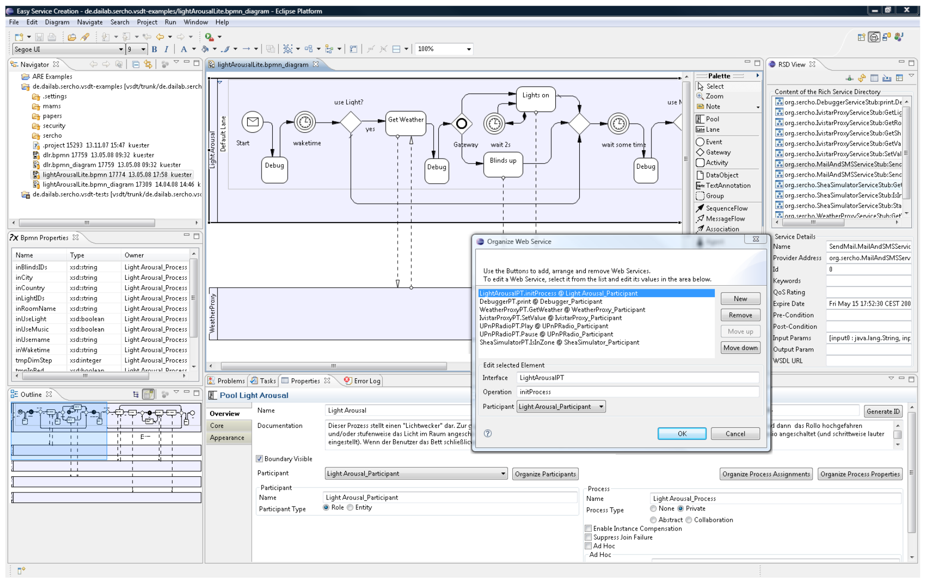


Fig. 1. The Visual Service Design Tool. Clockwise: Editor view, RSD client, web services dialog, property sheet, visual outline, properties inspector, navigator.

BPEL syntax. However, it is possible to supplement the editor with additional plugins, which can contribute e.g. to the property sheets or provide whole new views with language-specific functionality.

One example of how the VSDT can be extended with features specific to a certain target language — in this case: BPEL — is the RSD View, which can bee seen in Figure 1, too: A client for the *Rich Service Directory*, a special kind of Web service repository. Using the RSD View, existing Web services that have been registered at the RSD server can be inspected and imported into the diagram. In the process, an Implementation object is created for the Web service as well as a set of Message objects, matching the service's input parameters and result. Optionally, also a new Pool will be created for the service, which can be connected to the currently selected Activity via a pair of Message Flows. Further, the Implementation and the Message objects will be associated to the Activity and its type will be set to SERVICE. Thus, the orchestration of existing Web services in a BPEL process can be simplified greatly. Similar features can be created for other target languages, too.

Once the business process diagram is completed it can be validated and exported. As the VSDT is intended to provide export features to arbitrary target languages, and to support the tool smiths in the creation of these features, we have created an elaborate export framework, which we will have a closer look at in the next section.

3 The Transformation Framework

The core of the Visual Service Design Tool clearly is the transformation to executable code. While by now the transformation to BPEL is the only one that can be conveniently used in practice, and thus will serve as an example later in this section, there are currently several other transformations under development.

The transformation framework has been designed from the very beginning to be as *extensible* and *reusable* as possible. For that purpose the process of transformation has been subdivided into several stages, which are sequentially applied to the input model:

1. *Validation*: Validate the input model.
2. *Normalisation*: Prepare the input model for transformation.
3. *Structure Mapping*: Convert the input model to a block-like structure.
4. *Element Mapping*: Perform the actual mapping, create target model.
5. *Clean Up*: Remove redundancies, improve readability, etc.

The several stages are realised either as a set of graph transformation rules, a top-down pass through the input model, or a combination of both. For the graph transformation rules the *Tiger EMF Model Transformation* Framework [1] (EMT) has been employed, providing a fast pattern matching and backtracking algorithm for EMF models. In EMT, rules can be specified using a convenient graphical editor. For the VSDT, however, the EMT has been modified so that instead of a Left Hand Side (LHS) with Negative Application Conditions (NACs) and a Right Hand Side (RHS), the rules feature an LHS, NACs and an `execute` method, which may contain arbitrary Java code. Given the several cases to consider in BPMN this has proven more feasible. The transformation is operating on a copy of the model to be transformed, which can be modified in the course of the transformation without affecting the original diagram.

3.1 Stages of the Transformation

The Validation, Normalisation and Structure Mapping are to a great part independent of a specific target language, and in most cases the standard implementations provided with the transformation framework can be used. However, it can be advantageous to extend them with additional checks and rules.

For instance, in the *Validation* stage, all identifiers are validated to contain only characters that are legal with respect to the given target language, which can be achieved by extending the standard implementation and using a respective regular expression for the validation of names. Further, the validation includes a pass through the model, checking if each element needed is in place, thus reducing the number of checks necessary in the actual transformation, and providing clearer error messages to the user in case something is missing.

The intent of the *Normalisation* stage is to put the process diagram in a uniform form, and to transform it to what in the following will be referred to as the BPD's *normal form*, a semantically equivalent representation of the diagram following more strict constraints than those given in the BPMN specification. The transformation rules that are used in this stage are rather simple. For instance, one rule will check if there are any Activities with multiple incoming or outgoing Sequence Flows attached to it, in which case a Gateway of type XOR or AND, depending on whether the Sequence Flows have any conditions, will be inserted in between. Another rule will insert a "no-op" Activity in between any two Gateways that are directly connected by a Sequence Flow. The advantage is that after the application of the normalisation stage there will be much fewer cases to consider in the structure mapping, which will be described in the next paragraph. A simple example of the consecutive execution of normalisation and structure mapping can be seen in Figure 2.

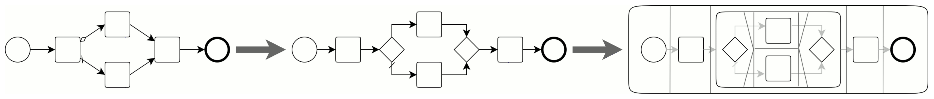


Fig. 2. Simple example of normalisation and structure mapping.

One of the challenges in transforming BPMN to executable languages is the mapping of the process model's graph-oriented structure to a more rigid block-oriented structure. For that reason it is of great benefit making this part, the *Structure Mapping*, independent of the actual target language, so it can be reused in mappings to other block-oriented languages. We decided to follow a *Structure Identification* strategy [9], being independent of BPEL's Link element. As mentioned in Section 2.1, the transformation is using an extension of the BPMN metamodel used in the editor, allowing the introduction of additional elements representing sequences, blocks for parallel and alternative routing, loops, and event handler blocks, i.e. the basic building blocks of block-oriented languages. Now, in the structure mapping stage, the model is searched for graph patterns which are semantically equivalent to these blocks, e.g. two Flow Object nodes connected with a Sequence Flow, or two Gateways connected by a number of branches of Flow Objects. When such a pattern is found, it is replaced with the respective structured element, removing the involved Sequence Flow edges in the course, which are then no longer needed (their conditions, if any, are preserved in the newly created structured elements). With these elements themselves being Flow Objects again, the rules of the structure mapping are applied until the entire process within each Pool has been reduced to a single complex element, e.g. a sequence, or until it can not be reduced any further due to structural flaws. Some examples of BPMN graphs that can successfully be mapped to equivalent block structures and further to executable BPEL code can be seen in Figure 3. Of course, this stage can be adapted or entirely omitted, too, if the target language is structured differently.

After the rule-based structure mapping, in the *Element Mapping* stage, the several BPMN elements can be mapped in a relatively simple top-down pass through the model. We decided to use a top-down pass instead of rules in this

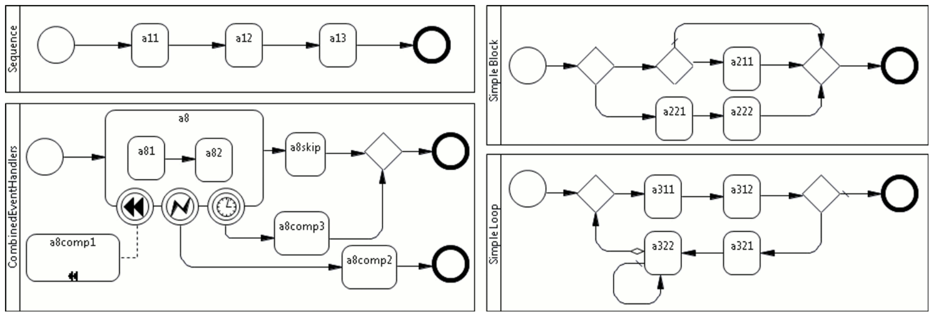


Fig. 3. Some examples of transformable BPMN graphs.

stage, as it is faster and easier to maintain, but the framework does allow for other implementations as well. As the element mapping is very dependent on the actual target language we will go further into detail later, regarding the transformation to BPEL.

Finally, in the *Clean Up* stage, a set of rules is applied on the newly created target model. While this stage is optional, it can be of great use for improving the readability of the generated code while at the same time keeping the required logic out of the earlier stages. For instance, nested sequences will be flattened, or sequences that hold a single element are replaced by that element itself. Further, elements that resulted from no-op Activities inserted in the normalisation stage should be removed again in this stage. As this stage operates on the target model, is has to be implemented anew for each target language.

For implementing a specific transformation, all that has to be done is to specify the element mapping, which can be done in any desired fashion by extending a special abstract class (see Figure 4). In case the target language uses different block concepts, the structure mapping has to be adapted, too, but should still be reusable to some point. In the majority of cases, implementing the other stages is optional.

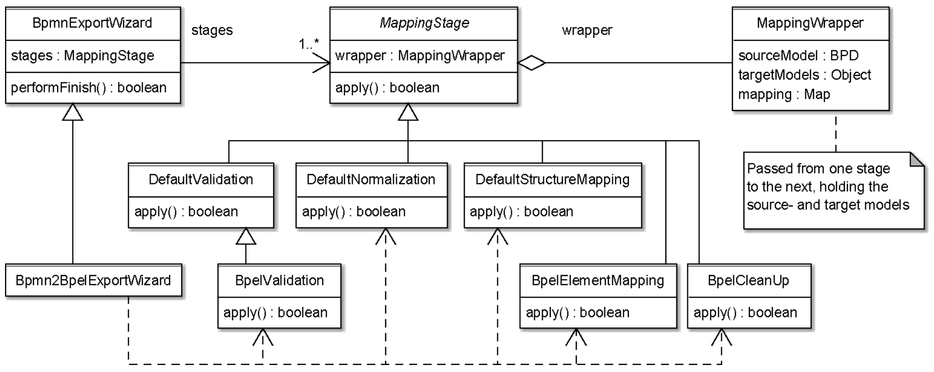


Fig. 4. Essential classes of the transformation framework, including the BPEL case.

3.2 Transformation to BPEL

The transformation to BPEL presented in this work covers nearly the entire mapping as given in the BPMN specification [10, Chapter 11], including event handlers, inclusive OR and event-based XOR Gateways, just to name a few. Still there are some elements for which the mapping is not given very clearly, such as TIMER Start Events, independent Sub Processes or multi-instance parallel loops. While these elements will be transformed as described in the specification, the resulting BPEL processes will require some amount of manual refinement. Besides the BPEL process files a WSDL definitions file is created, holding the message types derived from the process properties and the input and output messages and interfaces (port types) for the several Web services being orchestrated by the process. Still, the WSDL's binding and service blocks and necessary schema types, if any, can not be generated automatically yet, due to insufficient information in the source model. We are currently investigating ways of extending the BPMN metamodel in order to include more information in the model and at the same time making it more independent of the BPEL language.

In the validation used for the transformation to BPEL, all identifiers are tested to contain only characters that are legal with respect to BPEL. Additionally all expressions used e.g. in Assignments and loop conditions are scanned for occurrences of Property identifiers. So if a Process `Proc` has a Property `foo` and there is an Assignment with an expression like `"foo+1"`, the expression will be changed to `"bpws:getVariableData('Proc_ProcessData','foo')+1"`. Thus the user does not have to care about the way Properties are represented with messages in BPEL but can use a Property's plain name in expressions.

3.3 Example

The following example will show one of the scenarios being used in a smart home environment in the SerCHo project. The resulting BPEL processes were validated and tested with the *ActiveBPEL* designer and process engine.[2]

The BPMN diagram in Figure 5 is showing a "Light Alarm" process, that is used to open the blinds in the user's room to wake her up in a more pleasant way than the usual alarm clocks do. For that purpose, firstly information on the current weather is retrieved using an external Web service. Thereafter, based on the weather data, either the sunblinds are opened, or the ceiling light is turned on, or both. In case the user does not get up, which is checked using an RFID based localisation solution, the stereo is turned on, playing her favourite song or alternatively an unpleasant alarm sound.

For each of the above devices — blinds, lights, localisation, and stereo — Web service interfaces were written, so they can be integrated in a BPEL process. While the WSDL file that is used by the process, holding the definitions for the various orchestrated services, has to be extended with the service bindings, the BPEL code resulting from this example is readily executable.

[2] `http://www.activebpel.org`

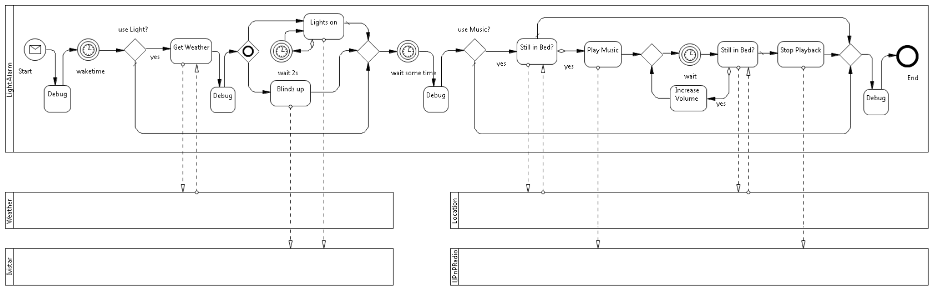


Fig. 5. "Light Alarm" Example Process

3.4 Transformation to JIAC

Concerning our goal of transforming BPMN diagrams to multi-agent systems (MAS) the work is still at an early stage. First, a *normal form* for BPMN diagrams has been investigated, to facilitate the mapping [2]. Later, the first steps of the actual mapping have been developed, basically mapping Pools to agents, Processes and Flow Objects to the agents' plans and the control flow, and Message Flow to the exchange of messages between the agents [3].

A first prototype targeting the agent framework JIAC IV [14] has already been implemented. As the theoretical part of the mapping is not yet fully matured, there is still some work to do. However, with the given transformation framework every addition to the mapping can quickly be adopted.

4 Related Work

The Business Process Modelling Notation has been adopted in a large number of tools. Although many of these are merely diagram drawing tools and do not support the transformation to BPEL, let alone other languages, there are some powerful tools as well. In the following we will introduce some of these. A more extensive list can be found at `http://www.bpmn.org/BPMN_Supporters.htm`.

With the free *eBPMN*, Soyatec provides a very nice BPMN editor, but it does not implement the mapping to BPEL.[3] The same applies to the free community edition of eClarus' *Business Process Modeller*, while the commercial *SOA-Architect* version provides a transformation to BPEL, although it seems to have some limitations.[4] A very mature BPM product can be found in the *Intalio BPMS*.[5] BPEL code is generated on-the-fly and can be deployed to the Intalio process engine. However, the mapping of workflow structures is limited, e.g. we found it impossible to merge a branch originating from an event handler back into the normal flow. Another limitation arises from the tight coupling to the inhouse BPEL engine, which is using some proprietary extensions. Further, Intalio

[3] `http://www.soyatec.com/ebpmn`
[4] `http://www.eclarus.com/`
[5] `http://www.intalio.com/`

has donated parts of the code to the Eclipse SOA Tools Project (STP). While the *STP BPMN Editor* itself does not provide a transformation to BPEL, Giner et.al. were able to combine it with the *BABEL Bpmn2Bpel* tool [4], yet both the editor and the transformation tool are using very simple metamodels.

Concerning the transformation from graph-oriented to block-oriented process models, as in the BPMN to BPEL case, Mendling et. al. have evaluated several transformation strategies [9], ranging from a straight-forward mapping of BPMN Sequence Flows to BPEL Links, similar to the one in [17], to a more sophisticated *Structure Identification* strategy, like the one applied in this work, or *Structure Maximisation*, as followed by Aalst and Lassen [15]. In their theoretically well-founded, pattern-based transformation from Petri nets to BPEL, they focus on the readability of the resulting code. However, they do not regard how highly *unstructured* workflows can be transformed to structured ones. As pointed out in [12], there is a "mismatch" between BPMN and BPEL, both on the domain representation and the control flow level, that is not easily to overcome. Many authors have investigated whether different graph patterns can be transformed to an equivalent structured form [5, 8, 13], and came to the conclusion that even slight unstructuredness can require the introduction of additional variables or the duplication of parts of the workflow, even though the workflow models used in these works are much simpler than BPMN. For structuring such workflows, Koehler et. al. present a rule-based transformation based on continuation semantics [6]. Another approach is followed by Ouyang et. al., using BPEL event handlers as a form of `goto` command [11]. Their examples show how complicated a simple workflow can become when being structured.

Thus, as workflow design will be facilitated greatly if the user is not restricted to the use of block-oriented processes, a transformation of unstructured workflows to readily executable code will be highly desirable, so that such processes can be created by means of Model Driven Engineering.

5 Conclusion

In this paper the *Visual Service Design Tool* (VSDT) has been introduced: a BPMN editor featuring a state of the art transformation to BPEL, while at the same time being easily extensible with export functionality targeting other languages. The editor itself has been designed to be language independent, so it can be used for generating code for any language, given that a respective mapping from BPMN to that language exists. Transformations implementing these mappings can be plugged in to the VSDT together with additional editing features helping the user in the creation of diagrams to be exported to that language. For supporting the developer of these plugins, the VSDT comes with a transformation framework, based on the EMT graph transformation tool. Being subdivided into several stages, large parts of it can be reused throughout different mappings, such as the refactoring of the process graph to block-oriented structures.

With respect to its BPMN editing functionality and the transformation to BPEL, the VSDT does not have to hide behind its commercial competitors.

Implementing the mapping to BPEL as given in the BPMN specification, the tool can be used to generate readily executable code. Still it is recommended to validate the results with a native BPEL editor: While the creation of processes will be easier and faster using the VSDT, its desired independence of a specific language prohibits some BPEL specific features, such as editing assistance for assignment expressions. However, due to the plugin architecture provided by the Eclipse platform, such functionality can be added together with the actual transformation features.

As the key feature of the VSDT is the extensibility with additional export features, further transformations from BPMN to executable languages are currently under development. One of the main goals of our research in this field is a mapping from BPMN to multi-agent systems, combining the intuitive design of business processes with the flexibility of software agent.

5.1 Future Work

Some work still can be done in the field of transformation of unstructured processes. Currently the tool can handle slightly unstructured workflows, such as one Gateway being used for merging multiple decision blocks, but will fail when faced with e.g. overlapping blocks or multiple exits from a loop. Here, further evaluations of the different possibilities to handle such workflows and ways of adapting them to the more complex BPMN diagrams will be necessary.

Concerning the transformation to BPEL, the support for complex data types will need further refinement. Here, the *Rich Service Directory* introduced earlier will be of great use, providing the necessary information about the involved Web services. Finally, the mapping to multi-agent systems has to be completed, and mappings to further languages will be evaluated.

References

1. Enrico Biermann, Karsten Ehrig, Christian Köhler, Günter Kuhns, Gabriele Taentzer, and Eduard Weiss. Graphical definition of in-place transformations in the eclipse modeling framework. In *Proc. 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, Genova, Italy, October 2006.
2. Holger Endert, Benjamin Hirsch, Tobias Küster, and Sahin Albayrak. Towards a mapping from BPMN to agents. In Jingshan Huang, Ryszard Kowalczyk, Zakaria Maamar, David Martin, Ingo Müller, Suzette Stoutenburg, and Katia P. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering*, volume 4505 of *LNCS*, pages 92–106. Springer Berlin / Heidelberg, 2007.
3. Holger Endert, Tobias Küster, Benjamin Hirsch, and Sahin Albayrak. Mapping BPMN to agents: An analysis. In Matteo Baldoni, Cristina Baroglio, and Viviana Mascardi, editors, *Agents, Web-Services, and Ontologies Integrated Methodologies*, pages 43–58, 2007.
4. Pau Giner, Victoria Torres, and Vicente Pelechano. Bridging the gap between BPMN and WS-BPEL. M2M transformations in practice. In *Proc. of the 3rd*

International Workshop on Model-Driven Web Engineering (MDWE 2007), Como, Italy, July 2007.

5. Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. On structured workflow modelling. In *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 431–445, London, UK, 2000. Springer-Verlag.
6. Jana Koehler and Rainer Hauser. Untangling unstructured cyclic flows - a solution based on continuations. In R. Meesman and Z. Tari, editors, *CooplIS/DOA/ODBASE 2004*, volume 3290 of *LNCS*, pages 121–138. IBM Zurich Research Laboratory, Springer-Verlag, 2004.
7. Tobias Küster. Development of a visual service design tool providing a mapping from BPMN to JIAC. Diploma thesis, Technische Universität Berlin, April 2007.
8. Rong Liu and Akhil Kumar. An analysis and taxonomy of unstructured workflows. In Wil M. P. van der Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera, editors, *Business Process Management*, volume 3649, pages 268–284, 2005.
9. Jan Mendling, Kristian Bisgaard Lassen, and Uwe Zdun. Transformation strategies between blockoriented and graph-oriented process modelling languages, 2005.
10. Object Management Group. Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification dtc/06-02-01, OMG, 2006. `http://www.bpmn.org/Documents/OMGFinalAdoptedBPMN1-0Spec06-02-01.pdf`.
11. Chun Ouyang, Marlon Dumas, Stephan Breutel, and Arthur H. M. ter Hofstede. Translating standard process models to BPEL. In Eric Dubois and Klaus Pohl, editors, *CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2006.
12. Jan Recker and Jan Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In T. Latour and M. Petit, editors, *In: T. Latour, M. Petit, eds.: CAiSE 2006 Workshop Proceedings - Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2006), June 5 - 6, 2006, Luxembourg, pages 521-532.*, pages 521–532, 2006.
13. Wasim Sadiq and Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134, 2000.
14. Ralf Sesseler. *Eine modulare Architektur für dienstbasierte Interaktionen zwischen Agenten*. PhD thesis, Technische Universität Berlin, 2002.
15. Wil M. P. van der Aalst and Kristian Bisgaard Lassen. Translating unstructured workflow processes to readable BPEL: Theory and implementation. *Inf. Softw. Technol.*, 50(3):131–159, 2008.
16. Stephen A. White. Introduction to BPMN. Technical report, IBM Corporation, May 2004. `http://bpmn.org/Documents/Introduction%20to%20BPMN.pdf`.
17. Stephen A. White. Using BPMN to model a BPEL process. Technical report, IBM Corporation, March 2005. `http://www.bpmn.org/Documents/Mapping%20BPMN%20to%20BPEL%20Example.pdf`.

Business Process Modelling with Continuous Validation

Stefan Kühne[1], Heiko Kern[1], Volker Gruhn[2], and Ralf Laue[2]

[1] Business Information Systems, University of Leipzig
Johannisgasse 26, 04103 Leipzig, Germany
{kuehne|kern}@informatik.uni-leipzig.de

[2] Applied Telematics/e-Business**, University of Leipzig
Klostergasse 3, 04109 Leipzig, Germany
{gruhn|laue}@ebus.informatik.uni-leipzig.de

Abstract. In this paper, we demonstrate the prototype of a modelling tool that applies graph-based rules for identifying problems in business process models. The advantages of our approach are twofold. Firstly, it is not necessary to compute the complete state space of the model in order to find errors. Secondly, our technique can even be applied to incomplete business process models. Thus, the modeller can be supported by direct feedback during the model construction. This feedback does not only report problems, but it also identifies their reasons and makes suggestions for improvements.

1 Introduction

Validation of business process models has been studied for a long time. In a recent paper [1], Wynn et al. write that "process verification has matured to a level where it can be used in practice". Although this is good news, we argue that many of the current approaches do not yet support the business process modeller in an optimal way. The reason for this statement is that most validation methods are applied only after the model has already been completed. For example, all those methods which transform a business process model into an analyzable Petri-net have problems with incomplete models.

In this paper, we present a validation approach that gives the modeller an immediate feedback about modelling errors. A prototypical implementation of our approach has been integrated into a business process model editor. It locates not only "technical" errors (such as deadlocks in the control flow), but also parts of the model that can be regarded as "bad style". The modeller not only receives the information that the model has problems, but our tool also shows the locations of error causes in the visual representation and suggests how to fix the problems. A rule language allows the user to add own rules, for example, rules for checking company-wide style guidelines.

Similar to techniques such as continuous compilation and continuous testing that are integrated into modern software development systems, our approach –

** The Chair of Applied Telematics/e-Business is endowed by Deutsche Telekom AG.

which we call *continuous validation* – can help to detect and fix errors at a very early stage.

2 Basic Concepts and Definitions

2.1 Event-Driven Process Chains

There exist several languages for graphical business process modelling. In this paper, we use Event-Driven Process Chains (EPC) [2] to demonstrate our approach. However, the underlying principles can be applied to other languages such as BPMN [3] as well.

We would like to start with a semi-formal description based on the metamodel given in Fig. 1. EPC models are finite directed coherent graphs consisting of non-empty sets of nodes and arcs. Nodes are either functions (activities which need to be executed, depicted as rounded boxes), or events (representing pre- and postconditions of functions, depicted as hexagons) or connectors. Arcs between these elements represent the control flow. A function has exactly one incoming and exactly one outgoing arc. An event has at most one incoming and at most one outgoing arc. An event without incoming arcs is called *start event*, and an event without outgoing arcs is called *end event*.

The connectors are used to model parallel and alternative executions. There are two kinds of connectors – splits and joins. Splits have one incoming and at least two outgoing arcs, joins have at least two incoming arcs and one outgoing arc.

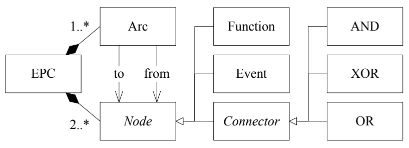


Fig. 1. EPC metamodel

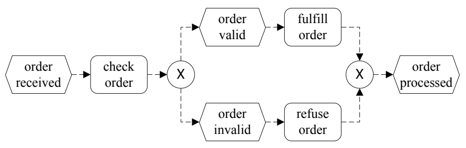


Fig. 2. A simple EPC

AND-connectors (depicted as $\owedge$) are used to model parallel execution. When an AND-split is executed, the elements on all outgoing arcs have to be executed in parallel. An AND-join connector waits until all parallel control flows on its incoming arcs are finished. XOR-connectors (depicted as $\otimes$) can be used to model alternative execution: An XOR-split has multiple outgoing arcs, but only one of them will be processed. An XOR-join waits for the completion of the control flow of one of its incoming arcs. If flow arrives from more than one arc most semantic definitions regard it as a synchronisation error. No flow of control is forwarded in this case. OR-connectors (depicted as $\ovee$) are used to model parallel executions of one or more flows. An OR-split starts the processing of one or more of its outgoing arcs. This means, after an OR-split with n outgoing arcs, at least one of those arcs and at most all n arcs become active. An OR-join waits until all control flows that can reach it are finished.

Fig. 2 shows a very simple order process modelled as EPC. Note that it is unnecessary that splits and joins occur pairwise and form a well-structured model. Actually, the notation allows arbitrary combinations of connectors which is often the cause for modelling errors.

A state of an EPC is a binary marking of its elements, i.e. some elements of an EPC are marked as “active” by placing tokens on them. A state is a start state iff start events are active. A sequence of states is an execution of the business process model. A transition relation defines the semantics, i.e. the rules that define under which circumstances a state S_1 in this sequence is allowed to be followed by a subsequent state S_2. Several different definitions exist for transition relations; because of space restrictions we omit a detailed discussion. The interested reader is referred to [4, 5, 6].

2.2 Control Flow Errors

Van der Aalst [2] has defined *soundness* for EPCs which is the most important correctness criterion for business process models. This definition includes three required properties:

1. In every state that is reachable from a start state, there must be the possibility to reach a final state, i.e. a state without a subsequent state according to the transition relation *(option to complete)*.
2. If a state has no subsequent state (according to the transition relation that defines the precise semantics), then only end events must be marked in this state *(proper completion)*.
3. There is no element of the EPC that is never marked in any execution of the EPC *(no needless elements)*.

Violations of the soundness criterion usually indicate an error in the model. A typical example is a deadlock situation with an XOR-split which outgoing arcs are joined later by an AND-join. This example would lead to a violation of the second property: It is possible that no further progress in the execution of the EPC can be made, but the elements at the incoming arcs of the AND-join are still marked because the AND-join has to wait until *all* incoming arcs have been traversed.

3 Existing Verification Methods

The problem to overcome when verifying EPC models is that the modelling language has been introduced *without defining their semantics*. For this reason, the first step in a verification process is usually to transform the model into a

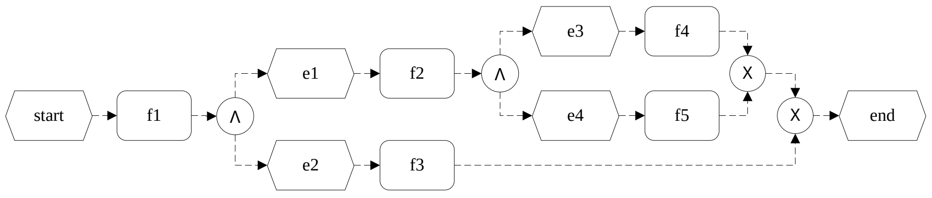


Fig. 3. Nested mismatched AND-splits and XOR-joins

formalism with well-defined semantics. In fact, the rules for such a transformation *define* the semantics of the model. Petri-nets are the natural choice for that purpose. They have been used by several authors [2, 7, 6][3].

Once an EPC model has been transformed into a Petri-net, the whole range of existing tools for analyzing Petri-nets is available. As shown in [9, 8, 5, 6], those tools can be used for checking the correctness of EPCs.

The Petri-net based approach works very well in practice, but has two disadvantages. First, often the analysis result covers only the information whether the model contains errors, without giving feedback about the reason for an error. Even if the verification tool "translates" the counterexample back to the EPC model, it is likely that information will be lost. For example, Fig. 3 obviously contains two synchronization problems – one in the outer AND-XOR control block and another one in the inner AND-XOR control block. However, the synchronization error in the inner control block makes the execution always blocks, and there will never be an execution where both incoming arcs of the rightmost XOR-join are enabled. For this reason, a dynamic analysis tool that explores the state space will be unable to report the problem of the outer AND-XOR control block. The second disadvantage is that it is often impossible to locate errors in models that are not yet completed (e.g., EPCs containing several subgraphs which are not yet connected with each other).

Another well-studied method for the validation of EPCs and similar models is the application of reduction rules (see e.g. [10]). The idea of the reduction approach is to delete repeatedly sections from an EPC which are well-structured (for example a control flow block where an AND-split is matched by an AND-join) and are thus trivially correct. If an EPC can be reduced to a single node in this way, it is correct. Otherwise, no answer about its correctness can be given. That is, the answer to the question "Are there any problems with the model?" is either "No" or "Don't know", which is far from the desired "No" or "Yes, and the problems are as follows...". However, recent work by Mendling [6] has made a fruitful contribution. By considering typical error situations in the reduction rules, Mendlings approach allows us to answer our question about errors in the model with "No", "Yes, and the problems are..." or "Don't know". In the latter case, Petri-net-based methods can still be used to come at least to a "Yes or No"

[3] The given references are not exhaustive. A more detailed categorization of related work can be found in [8] and [6].

answer. We considered the errors that can be found by reduction rules in [6] as a starting point for our own rules to locate errors (described in Sect. 5).

Our work has also been influenced by the approach described in [11]. This approach adapts ideas from Petri-net theory (like the concept of handles [12, 13]) to EPCs. In [11], the authors locate causal relationships between parts of an EPC. These relationships (called *causal footprints* in [11]) are relations like "after element X has been processed, at least one of the elements in the set $\{Y, Z\}$ has to be processed". Using this method, error patterns can be detected and the reasons for errors can be identified. Moreover, the method works on incomplete models as well. Unfortunately, in the form as described in [11], the approach has not yet matured for practical use. A minor reason is that it does not work with EPCs containing OR-connectors (but it would not be too much effort to expand the method such that OR connectors could be considered). But more important is that the computation of relationships among elements is far too slow because too many relations have to be considered.

Another promising heuristic approach has been described in [14]. It uses decomposition of workflow graphs into single-entry/single-exit fragments and can quickly classify some fragments as sound or unsound. A validation in [14] shows that the approach does well for the majority of a sample of industrial business processes.

4 Immediate Validation Feedback in Business Process Modelling

4.1 Validation Approach

From our point of view, a sophisticated validation support of business processes gives immediate and continuous feedback to business analysts about weaknesses and inconsistencies in possibly incomplete models. The established modelling process with sequential modelling, validation and evolution stages should be shortened as far as possible to a modelling process with integrated validation support. A analogy is a programming environment with continuous syntax checking which is in our case extended to semantic and even pragmatic issues.

Our intended validation support is based on three principles. (1) For adaptability and extensibility reasons the validation rules should be expressed in a modular and human-readable manner. For this purpose we propose declarative validation rules which enable the expression of additional error patterns and modelling idioms by adding new rules without effecting existing ones. (2) To avoid disadvantages of non-localised error messages, the validation strategy should work on input models in a native way. This can be realised by referring to model elements in validation rules or by delegating calculations of model properties to helper functions that navigate on models. (3) A rather soft requirement is that the validation solution is seamlessly integrated into the modelling tool of choice with the ability to annotate error causes and to suggest possible improvements.

In the area of business process modelling the instantiation of these principles results in process-specific validation rules and helper functions. Since EPC models as well as other business process modelling languages represent graphs, the required helper functions provide graph calculation functionalities such as “there is a path between two elements” or “every path between two elements goes through another element”. Furthermore, basic element-related operations, such as “this element is a start event” or “replace an element by another element” and set-related operations, such as “all start events of an EPC” or “intersection of all successors of an element and all predecessors of another element” are typically referenced in validation rules.

4.2 Implementation Strategy

Our approach is designed as an extension of the community-driven open-source EPC modelling tool *bflow*[4]. It, therefore, relies technically on the Eclipse Modeling Framework (EMF)[5] and the Graphical Modeling Framework (GMF)[6]. The implementation of our intended validation support requires a declarative validation language. Moreover, a model-to-model transformation language is required to express queries on models and possible error-resolving solutions. In the technical space EMF, openArchitectureWare (oAW)[7] and Epsilon[8] represent model management frameworks that provide these functionalities in an integrated manner. We chose oAW as implementation technology because of its build-in GMF integration.

Validation rules in our approach are expressed in the oAW Check language (see e.g. Listing 1). A declarative rule is introduced by a context specifying a metamodel element which instances are validated. The set of model elements can optionally be restricted by an if-clause. The ERROR keyword signals that a violated validation rule represents an error and specifies a corresponding advice. WARNING and INFO provide alternative error categories. The Boolean expression after the colon provides a validation assertion which holds for valid models.

Listing 1. A check rule

```
// Connectors are either splits or joins
context epc::Connector
    if this.isJoin()
    ERROR "Connector is a split and \
    a join as well. ..." :
    !this.isSplit();
```

Listing 2. An XTend function

```
// Is a connector a join−connector
Boolean isJoin(Element e) :
    epc::Connector.isInstance(e)
    && e.incomingArcs().size > 1;
```

[4] http://www.bflow.org/
[5] http://www.eclipse.org/modeling/emf/
[6] http://www.eclipse.org/modeling/gmf/
[7] http://www.eclipse.org/gmt/oaw/
[8] http://www.eclipse.org/gmt/epsilon/

The example given in Listing 1 assures that AND-, OR- or XOR-joins are not splits. The restriction expression as well as the assertion expression refer to separately defined oAW XTend functions. XTend is a functional programming language designed for model-to-model transformations. In our case XTend is used to express queries on the input model and to modify models. The definition of the *isJoin()* function used in Listing 2 is shown in Listing 2. Re-use is supported by the function concatenation operator "." and polymorphism.

While Check specifications work on EMF models, the oAW GMF adapter enables the execution of Check rules from a GMF editor. For this purpose the EPC-specific GMF plug-in representing the modelling editor has to be extended accordingly. Violated rules create corresponding markers that are assigned to the visual representation of model elements. In this way the modeller gets localised feedback. Furthermore, error messages may provide hints and improvement suggestions.

The build-in oAW GMF adapter supports the validation in read-only transactions. To enable model modifications in error cases with obvious improvements we changed this behaviour to read-write-transactions. A model modification that should be triggered by a validation rule can be expressed by a restriction expression matching the error pattern with a falsified assertion expression triggering the model modification (see Listing 3). XTend enables native Java calls which provides a technical interface for implementing user interactions in the case of non-obvious or alternative change suggestions.

Listing 3. A validation rule which triggers model modifications

```
// All elements are named
context epc::Node
    if (this.name == null)
    WARNING "Name was set to a default value" :
    (this.setDefaultName() -> false);
```

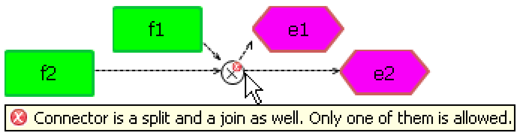
The mentioned technologies are independent of the EPC modelling language represented by the bflow tool and may be used to validate other model types. The contribution for the validation of BPM models results from a set of Check rules validating common classes of errors (discussed in Sect. 5). These rules rely on a limited set of performance optimised XTend functions.

The underlying principles of our approach are not restricted to the mentioned technologies. They can be applied to other EPC modelling tools, e.g. ARIS Design Platform with its macro language by IDS Scheer[9].

5 Examples

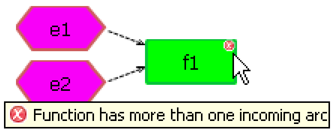
In this chapter, we discuss typical classes of modelling errors that are identified by checking rules included in our tool.

[9] http://www.ids-scheer.com



Connector is a split and a join as well. Only one of them is allowed.

(a)



Function has more than one incoming arc

(b)

Fig. 4. Connector with several incoming and outgoing arcs (a), function with several incoming arcs (b)

5.1 Syntax Errors

A less complicated task is to check an EPC (or to be precise: a construction that is supposed to be one) for syntactical correctness. Not all syntactical requirements are included into the EMF meta-model. Some constraints like non-existence of cycles made from connectors only require a few separate rules [15]. Writing syntax checking rules is in no way difficult [16] and is supported in our approach.

Fig. 4(a) shows a syntax error which violates the Check rule given in Listing 1. Another common syntax error is that an event or a function has more than one incoming or outgoing arcs (see e.g. Fig. 4(b)). Such errors are not uncommon: We found 14 of them in the 604 models of the SAP reference model.

5.2 Connector Mismatch

Informally spoken, deadlocks and synchronisation errors in an EPC result from a mismatch between the type of a split and the type of the corresponding join. If an XOR-split starts only one of two possible flows of control which are joined later by an AND-join waiting for both flows being completed, this will always result in a deadlock.

In an EPC that is not well-structured, the term of a "corresponding" join needs to be defined first. We do this as follows:

Definition 1 *A split s is matched by a join j (symbol: $match(s, j)$) iff there exist two directed paths from s to j which only common elements are s and j.*

If there are no "entries into" or "exits from" the control structure that starts with an XOR-split s and ends with and AND-join j, the process will always have an error regardless of the elements "between" s and j. The terms entries and exits are defined as follows:

Definition 2 *Let s be a split and j be a join with $match(s, j)$. We say that there are no entries and no exits between s and j, $seseMatch(s, j)$, if the following conditions hold:*

1. *Every path from s to an end event must contain j.*
2. *Every path from a start event to j must contain s.*
3. *Every path from s to s must contain j.*
4. *Every path from j to j must contain s.*

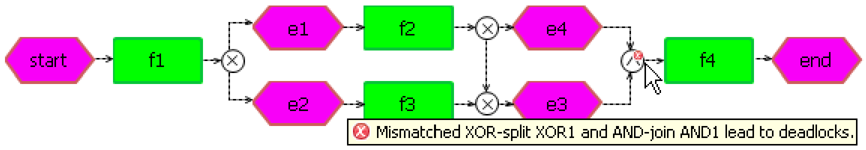
The above definitions allow us to find a certain type of control flow errors where the type of the split differs from the type of the join. From the 178 errors found in [6], 44 fell into this category.

We have similar rules for finding errors that occur in control flow blocks with entries and exits and for finding errors that occur in iterations (circles) in an EPC. Due to space restrictions, we do not describe them in detail.

Fig. 5 shows an obvious error consisting of a mismatched XOR-split (the left one)/AND-join combination. The two XOR-connectors inside the fragment illustrate that the rule given in Listing 4 also finds errors in non well-structured models.

Listing 4. Mismatched XOR-split and AND-join

```
// XOR−AND−Mismatch
context iepc::Connector if (this.isAndJoin())
    ERROR "Mismatched XOR−split ..."
    this.allPredessesors().notExists(p| p.isXorSplit() && p.seseMatch(this));
```



Mismatched XOR-split XOR1 and AND-join AND1 lead to deadlocks.

Fig. 5. Mismatched XOR-split and AND-join

For an XOR-split/OR-join combination, the situation is different: The OR-join waits for all incoming flows to be completed. If there is only one such flow (as the result of the choice at the XOR-split), the execution will continue without problems. For this reason, most verification approaches (a remarkable exception is [5]) do not complain about such combinations. Our rules produce a warning, because it is a good advice to the modeller to change the OR-join into an XOR-join in order to avoid misunderstandings.

For EPCs that occur in practice, the checks discussed in this section can be performed fast; running them repeatedly as a background task is not a problem.

5.3 Synchronisation Problem in AND-join

The most common type of error found in [6] (102 errors out of 178 found) falls into a category that is depicted in Fig. 6: If the upper outgoing path at the XOR-split is processed, a deadlock will be produced at the AND-join.[10] Fig. 6 shows how our tool signals the problem. Listing 5 shows the corresponding Check rule for detecting such problems.

[10] In practice, we would not regard all such models as erroneous, but this discussion is outside the scope of this paper.

Listing 5. Check rule for (X)OR-split caused synchronisation problems in AND-joins

```
context epc::Connector if this.isAndJoin() // An AND−Join J
    ERROR "AND−split might not get control" :
    // has no S with S is connected to J and
    this.allPredessesors().notExists(S|
        // S is an XOR−split or an OR−split
        (S.isXorSplit() || S.isOrSplit())
        // and S has a successor SSucc not conntected to J
        && S.successors().exists(SSucc|
            SSucc != this && SSucc.isNotConnectedTo(this))
        // and J has a predecessor JPred not connected from S
        && this.precessors().exists(JPred|
            JPred != S && S.isNotConnectedTo(JPred)) );
```

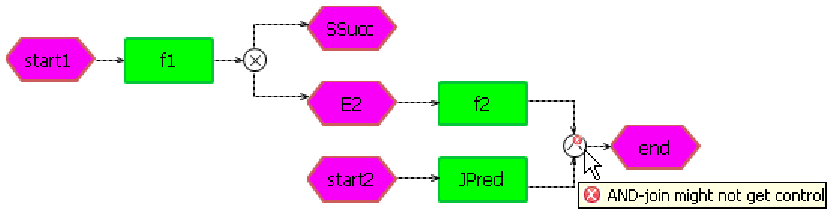


Fig. 6. Synchronisation problem in an AND-join

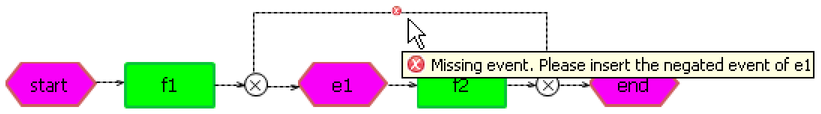
5.4 Company-Wide Style Rules

The rule set in our tool is open for adding new rules. If, for instance, a company wants to transform the EPC model later directly into executable BPEL code which enforces well-structuredness, it could be a requirement to use certain company-wide style rules for EPCs in order to disallow unstructured EPC models.

Such domain-specific adaptations can simply be realised in our tool by extending the predefined rule set. An example for such a rule is given in Fig. 7. The intention of the fragment between the two XOR-connectors is that function f2 is executed only if event e1 occurs which corresponds to a single if-expression in a programming language. However, because of a missing condition the arc connecting the two XOR-connectors might also be regarded as path that can always be executed. To avoid ambiguities a distinguishing event is desirable. With a rather simple check rule this pattern can found and the suggestion to insert a negation of event e1 can be created.

6 Validation

We used the appendix of Jan Mendlings PhD thesis [6] for a first validation of our checking rules. It contains the results from validating 604 EPCs from the



Missing event. Please insert the negated event of e1

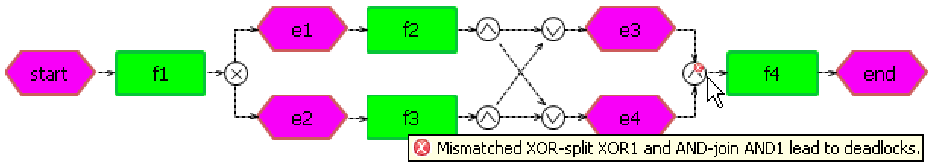
Fig. 7. Missing event between an XOR-split and XOR-join with a suggestion for improvement

SAP reference model. By using reduction rules, 178 error patterns have been found in 90 models. Our tool detected 176 of these errors (and several other errors that remained undetected by reduction rules). The remaining 2 errors are loops that start with an OR-join instead of an XOR-join. We regard the fact that this results in a deadlock with the semantics from [6] rather as a shortcoming of the semantics definition in [6].

For 57 models, the reduction rules in [6] did not reveal an error. Mendling used a state space exploring algorithm for judging about the soundness of these 57 models. 36 have been proved to be unsound, and our tool located one or more errors in *all* of these models. The remaining 21 models have shown to be sound, and for those models our tool produced only one alert.

7 Conclusions and Directions for Future Work

With the error patterns discussed in Sect. 5, we can already identify the vast majority of control flow problems in an EPC. However, the presented validation approach basically has a heuristic nature. The checks in the editor are not meant to be a complete validation. The EPC depicted in Fig. 8 shows two weaknesses of our approach. Although the model represents a sound EPC, the rules discussed so far produce a false error message for the AND-join. Furthermore our rules do not create hints according to the OR-joins which might be replaced by two XOR-joins. We deliberately did not try to include rules for such exotic cases.



Mismatched XOR-split XOR1 and AND-join AND1 lead to deadlocks.

Fig. 8. Sound EPC for which our tool would still report a problem

An evaluation based on the set of EPCs given in the SAP reference model and other EPC repositories shows that a limited set of rules already detects the vast majority of common errors. We see the main advantage of our approach in the fact that information about possible problems in a model is immediately reported to the modeller, even before the model has been completed. These alerts provide suggestions for improvement and are given in a way that does not force the modeller's attention away from the modelling task.

It will be a direction of future research to deal with patterns where a model change could result in more *readable* models – even for EPCs where the original model did not have deadlocks and similar control-flow problems [17].

References

1. Wynn, M.T., Verbeek, H., van der Aalst, W.M., Edmond, D.: Business Process Verification - Finally a Reality! Business Process Management Journal **to appear**
2. van der Aalst, W.M.: Formalization and verification of event-driven process chains. Information & Software Technology **41**(10) (1999) 639–650
3. Business Process Management Initiative: Business Process Modeling Notation. Technical report, BPMI.org (2004)
4. Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: Business Process Management. (2004) 82–97
5. Wynn, M.T.: Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins. PhD thesis, Queensland University of Technology, Brisbane (2006)
6. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration (2007)
7. van Dongen, B.F., van der Aalst, W.M., Verbeek, H.M.W.: Verification of EPCs: Using Reduction Rules and Petri Nets. In: CAiSE. (2005) 372–386
8. van Dongen, B.F., Jansen-Vullers, M., Verbeek, H.M.W., van der Aalst, W.M.: Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. Comput. Ind. **58**(6) (2007) 578–601
9. Langner, P., Schneider, C., Wehler, J.: Relating Event-driven Process Chains to Boolean Petri Nets. Report (9707) (December 1997)
10. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Information Systems **25(2)** (June 2000) 117–134
11. van Dongen, B., Mendling, J., van der Aalst, W.: Structural Patterns for Soundness of Business Process Models. EDOC (2006) 116–128
12. Esparza, J., Silva, M.: Circuits, handles, bridges and nets. In: Applications and Theory of Petri Nets. (1989) 210–242
13. van der Aalst, W.M.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers **8**(1) (1998) 21–66
14. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. Service-Oriented Computing – ICSOC 2007 (Jan 2007) 43–55
15. Nüttgens, M., Rump, F.J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen. (2002) 64–77
16. Gruhn, V., Laue, R.: Checking Properties of Business Process Models with Logic Programming. In: MSVVEIS 2007, INSTICC Press (2007) 84–93
17. Gruhn, V., Laue, R.: Good and Bad Excuses for Unstructured Business Process Models. In: Proceedings of 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007). (2007)

On the Formal Generation of Process Redesigns

Mariska Netjes, Hajo A. Reijers, and Wil M.P. van der Aalst

Eindhoven University of Technology, PO Box 513, NL-5600 MB Eindhoven, The Netherlands
{m.netjes,h.a.reijers,w.m.p.v.d.aalst}@tue.nl

Summary. Business Process Redesign (BPR) is a process oriented methodology to improve organizations. Although literature on BPR is available in abundance, little concrete support on how to get from *as-is* towards *to be* is available [11]. We propose the use of an evolutionary redesign approach that is based on BPR best practices to fill this gap. The approach is *evolutionary* in nature, because local updates are made to an existing process. In this paper we focus on one part of the approach: the generation of redesign alternatives. The first step in the generation of an alternative process is the selection of a process part for redesign. This is followed by a process transformation that determines an alternative for this selected part. Finally, the original process part is replaced by the transformed part resulting in the alternative process. Using Petri net analysis techniques the correctness of such a redesign creation is ensured.

Key words: Business Process Redesign, process modeling, Business Process Management, workflows, best practices

1 Introduction

Business Process Redesign (BPR) combines a radical restructuring of a business process with a wide-scale application of information technology [7]. Although many methods and tools are available to facilitate a BPR effort (e.g. [5, 6, 8]), little concrete support is provided on how to create the *to be* situation from the *as is* [11]. Many existing approaches and tools are limited in their application domain, while none of the approaches has succeeded to gain widespread adoption in industry. We would like to mention the approach and the tool that are most related to our work. Weber, Rinderle-Ma and Reichert [16] provide a comprehensive set of change patterns with formal semantics. The KOPeR tool [12] supports the detection of malfunctioning process parts and suggests the type of change that has to be made.

A BPR initiative would benefit from the use of *best practices*. A best practice is a historical solution that seems worthwhile to replicate in another situation or setting. A list of BPR best practices is presented in [14]. Such a list allows companies to use well-performing solutions from earlier redesign efforts. A BPR best practice supports practitioners in developing a process redesign by making evolutionary, local updates to an existing process. With our *evolutionary* approach to process redesign we aim to fill the gap between *as-is* and *to be* by taking an existing process model and improving it using the BPR best practices (from [14]). It is evolutionary, because an existing process is taken as a starting point.

Our evolutionary approach consists of six steps, cf. Figure 1:

1 model an existing process,
2 determine weaknesses in the process,
3 select applicable best practices,
4 generate alternative models,
5 evaluate the performance of the generated alternatives, and
6 choose the best alternative and implement the new process.

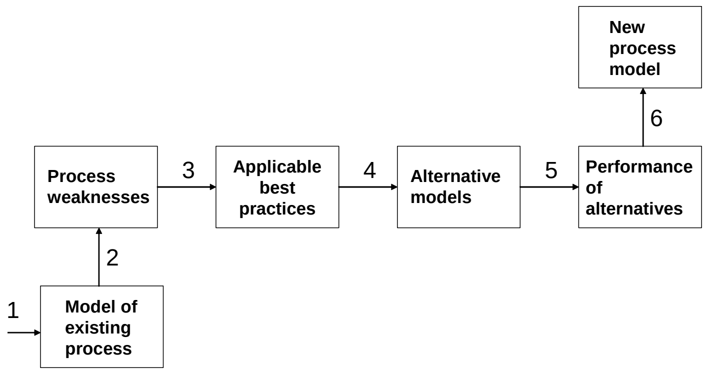

New process model
6
Process weaknesses
3
Applicable best practices
4
Alternative models
5
Performance of alternatives
2
1
Model of existing process

Fig. 1. Evolutionary approach towards redesign

The first three steps of Figure 1 we addressed in [9]. With regard to the modeling of an existing process (step 1), we gave a formal process definition and showed that it is not straightforward to spot inefficiencies in a process. For the determination of weaknesses (step 2), we presented a set of process measures, which provide a global view on the weaknesses in the process. For the selection of applicable best practices (step 3), we used and combined the set of process measures to evaluate the applicability of each best practice for the process.

In the next step, the generation of alternative redesign models (step 4), the best practices are applied. This step performs the actual change on part of the process, thus making a local update to the process. Then, the process can be redesigned again resulting in another update. Each iteration results in a redesign alternative which may be used as a starting point for another local update. In such a way, a tree of alternative process models is generated with the original process model as its root node.

In the final steps of our approach, the performance of the various alternatives is evaluated (step 5) and one redesign alternative is selected as the best alternative (step 6). Evaluation of process models can be done by simulating the model or (for simple processes) with analytical techniques. For such an evaluation, performance data (time, cost and quality indicators) are required. These data may be collected in event logs which are derived from the execution of the actual process. Log-based extension of a process model with a new aspect or perspective (e.g., enriching the model with performance data) is part of the process mining research [4].

In this paper, we focus on the fourth step of the evolutionary approach: *the generation of redesign alternatives*. An alternative is created based on the original model by selecting a 'malfunctioning' part of the model and 'curing' it with one of the best practices. We consider the reasons for the selection of a specific part of the process and a specific best practice to be out of the scope of this paper since these issues are addressed by the first three steps of the approach [9]. Our contribution with this paper is a formally defined method for the generation of a process redesign. This generation is performed in three steps. First, a process part is selected. Then, an alternative part is determined by process transformation. Finally, the original process part is replaced by the alternative part.

In Section 2 of this paper, we define our notion of a process. In Section 3, we discuss the three steps of the redesign generation: selection, transformation and replacement and illustrate these with a transformation called *parallel*. In Section 4, we describe three other transformations: *sequence*, *unfold* and *merge*. Finally, we conclude and give an outlook on future work.

2 Process Definition

The starting point of a redesign effort with our approach is a process that is currently being used in an organization. A process has certain characteristics, like its structure, and several properties.

2.1 Process Characteristics

In our formal process definition we distinguish between the *process structure* and *process information*. Both are necessary to model a realistic business process and to generate process redesigns that are applicable in practice. First, we focus on the process structure, for which we introduce the notion of a *SISO-net*. A SISO-net, as given in Definition 1, is a generalized WorkFlow net (WF-net) [1] with a single input (SI) and a single output (SO). Note that a WF-net is also a SISO-net, but that, for example, transition-bordered SISO-nets are not WF-nets.

Definition 1 (SISO-net) (P,T,F) *is a SISO-net if and only if there is a unique source node i and a unique sink node o such that:*

- $\{n \in P \cup T \mid \bullet n = \emptyset\} = \{i\}$,
- $\{n \in P \cup T \mid n\bullet = \emptyset\} = \{o\}$,
- $\forall_{n \in P \cup T} :\ (i,n) \in F^* \wedge (n,o) \in F^*$.

With the SISO-net we describe the process structure. Next to the process structure or control flow, we introduce process information. Process information consists of *dependencies* and *labels* that present the process information in a generic way. Dependencies exists between transitions, i.e., one transition depends on another. For each transition the dependencies, that should be satisfied before it can be executed, are defined as its input dependencies. The dependencies that are fulfilled after its execution are its output dependencies. Each transition may have a label assigned to it representing the type of the transition. Process information is defined as an *annotation* of the SISO-net, i.e., an annotated SISO-net is a SISO-net enriched with process information. An example of an annotated SISO-net can be found in Figure 2.

Definition 2 (Annotated SISO-net) *$S = (P, T, F, D, D_I, D_O, L, A)$ is an annotated SISO-net with:*

- *(P, T, F) is a SISO-net,*
- *D is a finite set of dependencies,*
- *$D_I \in T \rightarrow \mathcal{P}(D)$ [1] relates transitions to sets of input dependencies, i.e., these dependencies should be satisfied before a transition can become enabled,*
- *$D_O \in T \rightarrow \mathcal{P}(D)$ relates transitions to sets of output dependencies, i.e., these dependencies are fulfilled when a transition fires,*
- *L is a finite set of labels, and*
- *$A \in T \not\rightarrow L$ [2] assigns an optional label to each transition.*

We introduce some shorthand notations for the annotated SISO-net S:

- $\pi_P(S) = P$ are the places of an annotated SISO-net,
- $\pi_T(S) = T$ are the transitions of an annotated SISO-net,
- $\pi_F(S) = F$ is the flow relation of an annotated SISO-net,
- $\pi_D(S) = D$ is the set of dependencies in an annotated SISO-net,
- $\pi_{D_I}(S) = D_I$ is the set of input dependencies of an annotated SISO-net,
- $\pi_{D_O}(S) = D_O$ is the set of output dependencies of an annotated SISO-net,
- $\pi_L(S) = L$ is the set of labels in an annotated SISO-net,
- $\pi_A(S) = A$ is the label assignment of an annotated SISO-net.

We require an annotated SISO-net to be structured in such a way that during the execution of the net the input dependencies for each transition are fulfilled before it is possible to execute that transition. This means that each dependency present in an annotated SISO-net has to be fulfilled, i.e., is an output dependency, before it can be used as an input dependency. That is, for all $d \in D$ with $d \in D_O(t_1)$ and $d \in D_I(t_2)$ the structure of the process should be such that t_1 is placed before t_2. Note that this means that a dependency has to be used as an output dependency first, but that it does not imply that it has to be used as an input dependency. A dependency can be used zero, one or multiple times as an input dependency. It is in the nature of a dependency that it is used as an output dependency only once. Note that the requirements for the use of dependencies also imply that a transition can not have the same dependency both as an input and as an output.

Dependencies and *labels* may be specified according to the process information that is available within the organization and the specific redesign goals identified by the organization. Data elements are a specific way to represent the dependencies between transitions. The dependencies, then, model the data perspective of the process. Sun, Leon Zhao, Nunamaker and Liu Sheng describe the data perspective in more detail and provide an approach similar to ours for the detection and correction of data errors [15]. Other specific dependencies between transitions may be the ordering between

[1] $\mathcal{P}(X)$ is the powerset of X, i.e., $Y \in \mathcal{P}(X)$ if and only if $Y \subseteq X$.
[2] A is a partial function, i.e., the domain of A is a subset of T.

transitions in the original process or business dependencies identified by the organization. Roles represent a specific means to label transitions. The labels, then, model the resource perspective of the process. Other types of labeling, like departments, applications or geographical locations, may be used in a similar way.

2.2 Process Properties

The single input (SI) and the single output (SO) are important nodes in the annotated SISO-net and we give two operations to find the in- and output of the SISO-net.

Definition 3 (Input, output) *Let S be an annotated SISO-net with source node i and sink node o. Operation* `in` *returns the source node of S, i.e.,* $\mathtt{in}(S) = i$, *with* $i \in \pi_P(S) \cup \pi_T(S)$. *Operation* `out` *returns the sink node of S, i.e.,* $\mathtt{out}(S) = o$, *with* $o \in \pi_P(S) \cup \pi_T(S)$.

A SISO-net has properties similar to a WF-net. A WF-net is constructed from a SISO-net by adding a place to the begin (the end) of the SISO-net when it starts (ends) with a transition. Definition 4 gives some properties of a SISO-net, namely safeness and soundness. The basic idea is that these properties are based on the corresponding WF-net [1].

Definition 4 (Safe and sound annotated SISO-net) *An annotated SISO-net S is defined to be safe and sound if and only if the associated WF-net $\widetilde{S}$ is safe and sound.*

In the next section we describe the generation of a process redesign for an annotated SISO-net.

3 Selection, Transformation, and Replacement

When conducting a process redesign with the evolutionary approach, part of an existing process model is changed. This change is performed in three steps: selection, transformation and replacement.

3.1 Selection

The first step of the generation of a process redesign is the selection of the part of the process that should be changed. Figure 2 shows an example of a selected process part. The process part is selected from a blood donation process. This process described the donation of blood by donors in a donation center. In the first part of the process, the intake and condition check of the donor is performed. In the second part, that is selected, blood is taken from a donor and tested for several diseases. The donation center wants to shorten the duration of the lab tests.

For the selection we use the notion of a *component*. A component can be seen as a selected part of the annotated SISO-net with a clear start and end, i.e., a component is a sub net satisfying properties similar to an annotated SISO-net. Our definition of a component is derived from van der Aalst and Bisgaard Lassen [3] who introduced the notion of a component.

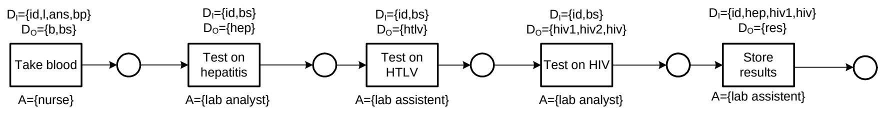


Fig. 2. Example: A selected process part

Definition 5 (Component) *Let S be an annotated SISO-net. Then C is a component in S if and only if:*

- $C \subseteq \pi_P(S) \cup \pi_T(S)$,
- *there are source and sink nodes* $i_C, o_C \in C$ *such that:*
  - $i_C \neq o_C$
  - $\bullet(C \setminus \{i_C\}) \subseteq C \setminus \{o_C\}$,
  - $(C \setminus \{o_C\})\bullet \subseteq C \setminus \{i_C\}$, *and*
  - $(o_C, i_C) \notin \pi_F(S)$.

Note that any component contains at least one transition and one place. We only consider non-trivial components, i.e., components with more than one transition. The following definition provides the projection of the net on the component.

Definition 6 (Projection) *Let S be an annotated SISO-net and let C be a component in S. The projection of S on C,* $S||_C$*, is then defined as* $S||_C = (P, T, F, D, D_I, D_O, L, A)$ *with:*

- $P = \pi_P(S) \cap C$ *is the set of places,*
- $T = \pi_T(S) \cap C$ *is the set of transitions,*
- $F = \pi_F(S) \cap (C \times C)$ *is the flow relation,*
- $D = \{d \in \pi_D(S) \mid \exists_{t \in \pi_T(S) \cap C} : d \in D_I(t) \vee d \in D_O(t)\}$ *is the set of dependencies,*
- $D_I \in T \rightarrow \mathcal{P}(D)$ *such that* $\forall_{t \in \pi_T(S) \cap C} : D_I(t) = \pi_{D_I}(S)(t)$ *is the set of input dependencies,*
- $D_O \in T \rightarrow \mathcal{P}(D)$ *such that* $\forall_{t \in \pi_T(S) \cap C} : D_O(t) = \pi_{D_O}(S)(t)$ *is the set of output dependencies,*
- $L = \{l \in \pi_L(S) \mid \exists_{t \in dom(\pi_A(S)) \cap C} : l \in A(t)\}$ *is the set of labels, and*
- $A \in (dom(\pi_A(S)) \cap C) \rightarrow L$ *such that* $\forall_{t \in dom(\pi_A(S)) \cap C} : A(t) = \pi_A(S)(t)$ *is the label assignment.*

Note that the shorthand notations for an annotated SISO-net can also be used for $S||_C$, the projection of S on C. The next theorem is used to prove the compositional nature of safe and sound annotated SISO-nets.

Theorem 1. *Let S be a safe and sound annotated SISO-net and let C be a component of S. The projection* $S||_C$ *is a safe and sound annotated SISO-net.*

Proof. See [10].

Soundness and safeness are desirable properties. Theorem 1 shows that these desirable properties are propagated to any component in the net. The component is the input for the next step: the transformation.

3.2 Transformation

In the transformation, an alternative process part is generated for the selected process part. The transformation is the actual change that is made to the process. The transformations presented in this paper are extensions of the basic soundness preserving transformation rules invented by van der Aalst [1]. A transformation rule presents the substitution of a transition with either two sequential tasks, two conditional tasks, two tasks in parallel, or an iterative task or the opposite transformation. Due to page limitations we discuss only one transformation in detail: the *parallel transformation* and describe the other transformations briefly in the next section.

In an existing process, tasks may be required to be executed one after another while no real dependencies exist between these tasks. This gives unnecessary delays in the process execution, because a task may be waiting for another task to finish, while this task is not depending on the results of that task. Often, it is a better option to perform tasks without dependencies between one another in parallel. Parallel tasks may be executed in any order or simultaneously. The obvious benefit would be a reduction in throughput time. A possible negative effect is a loss of quality because processes with concurrent behavior are more complex [13].

With the parallel transformation several non-dependent transitions, i.e., transitions with a disjoint set of dependencies, are placed in parallel. The parallel transformation is started with a component free of choice constructs [3] and results in an annotated SISO-net with the maximum parallel process structure.

For an annotated SISO-net S and for a choice construct free component C in S with its projection $S||_C$ being acyclic, we define the `parallel` operation. The transitions of the component are translated to the nodes of a graph, but transitions that solely serve as routing transitions are removed. Edges between the nodes reflect the dependencies and places are not necessary and therefore omitted.

Definition 7 (Parallel) *Operation* `parallel` *changes C into the annotated graph $G = (N,E,D,D_I,D_O,L,A)$, i.e.,* `parallel`$(S,C) = (N,E,D,D_I,D_O,L,A)$ *with:*

- $N = \{n \in (\pi_T(S||_C) \mid \pi_{D_I}(S)(n) \neq \emptyset \vee \pi_{D_O}(S)(n) \neq \emptyset\}$[4] *is the set of nodes,*
- $E = \{(n_1,n_2) \in N \times N \mid (\pi_{D_O}(S)(n_1) \cap \pi_{D_I}(S)(n_2)) \neq \emptyset\}$ *is the set of edges,*
- $D = \pi_D(S||_C)$ *is the set of dependencies,*
- $D_I \in N \rightarrow \mathcal{P}(D)$ *such that* $\forall_{n \in \{t \in \pi_T(S||_C) \mid \pi_{D_I}(S)(t) \neq \emptyset \vee \pi_{D_O}(S)(t) \neq \emptyset\}} : D_I(n) = \pi_{D_I}(S)(n)$ *is the set of input dependencies,*
- $D_O \in N \rightarrow \mathcal{P}(D)$ *such that* $\forall_{n \in \{t \in \pi_T(S||_C) \mid \pi_{D_I}(S)(t) \neq \emptyset \vee \pi_{D_O}(S)(t) \neq \emptyset\}} : D_O(n) = \pi_{D_O}(S)(n)$ *is the set of output dependencies,*
- $L = \pi_L(S||_C)$ *is the set of labels, and*
- $A \in dom(\pi_A(S||_C)) \rightarrow L$ *such that* $\forall_{t \in dom(\pi_A(S||_C))} : A(t) = \pi_A(S)(t)$ *is the label assignment.*

[3] Putting choice tasks in parallel would interfere with the choice behavior.

[4] Routing transitions, i.e., transitions for which holds $\pi_{D_I}(S)(t) = \pi_{D_O}(S)(t) = \emptyset$, are not considered and henceforth removed.

The annotated graph has to be further transformed into an annotated SISO-net. First, a source (or sink) node is added and connected to the start (or end) nodes if the graph starts (or ends) with more than one node. Then, superfluous relations are removed from the graph. Finally, the nodes in the annotated graph are translated into the transitions of an annotated SISO-net and places and corresponding arcs are added between these transitions. The result is a safe and sound annotated SISO-net. The complete formalization of the parallel transformation can be found in [10].

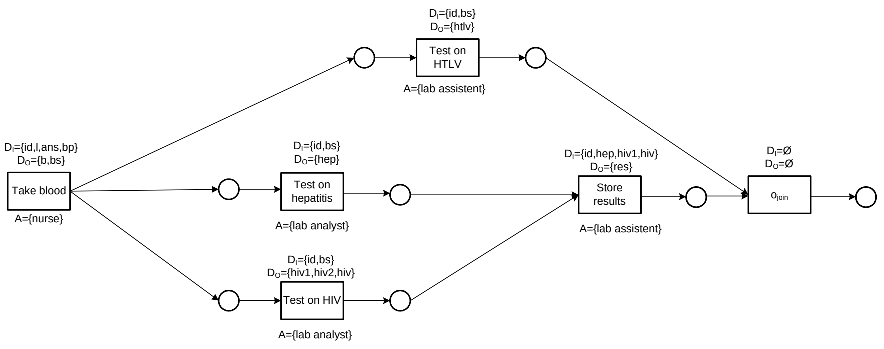


Fig. 3. Example: a transformed process part

Figure 3 shows the alternative process part generated with the parallel transformation for the selected process part in Figure 2. In the example it can be seen that, for instance, transition *Test on hepatitis* and transition *Test on HIV* do not share any dependencies and are placed in parallel. Transition *Store results*, however, depends on these two transitions, and is therefore placed after the two transitions. It is also interesting to note that the output of transition *Test on HTLV* is not used in any way. Real life processes may contain such inconsistencies and Sun *et al.* provide an approach to detect and correct such errors [15].

3.3 Replacement

After the selection of a process part and the creation of an alternative for this selected part with process transformation, an alternative process is constructed. The selected component is replaced by the created alternative process part.

Definition 8 (Replace) *Let S_1 and S_2 be two annotated SISO-nets. Let C, with source node i_C and sink node o_C, be a non-trivial component in S_1. Let $\mathtt{in}(S_2)$ be a place if and only if i_C is a place and let $\mathtt{out}(S_2)$ be a place if and only if o_C is a place. Operation $\mathtt{replace}$ substitutes $S_1||_C$ in S_1 with S_2 resulting in $S_3 = (P_3, T_3, F_3, D_3, D_{I3}, D_{O3}, L_3, A_3)$, i.e., $\mathtt{replace}(S_1, C, S_2) = (P_3, T_3, F_3, D_3, D_{I3}, D_{O3}, L_3, A_3)$ with:*

– *$P_3 = (\pi_P(S_1) \setminus C) \cup \pi_P(S_2)$ [5] is the set of places,*

[5] We assume there are no “name clashes”.

- $T_3 = (\pi_T(S_1) \setminus C) \cup \pi_T(S_2)$ *is the set of transitions,*
- $F_3 = (\pi_F(S_1) \cap ((P_3 \times T_3) \cup (T_3 \times P_3))) \cup$
  $\pi_F(S_2) \cup$
  $\{(n, \texttt{in}(S_2)) \mid (n, i_C) \in \pi_F(S_1)\} \cup$
  $\{(\texttt{out}(S_2), n) \mid (o_C, n) \in \pi_F(S_1)\}$ *is the flow relation,*
- $D_3 = \pi_D(S_1) \cup \pi_D(S_2)$ *is the set of dependencies,*
- $D_{I3} \in T_3 \rightarrow \mathcal{P}(D_3)$ *such that* $\forall_{t \in \pi_T(S_1) \setminus C} : D_{I3}(t) = \pi_{D_I}(S_1)(t)$ *and* $\forall_{t \in \pi_T(S_2)} : D_{I3}(t) = \pi_{D_I}(S_2)(t)$ *is the set of input dependencies,*
- $D_{O3} \in T_3 \rightarrow \mathcal{P}(D_3)$ *such that* $\forall_{t \in \pi_T(S_1) \setminus C} : D_{O3}(t) = \pi_{D_O}(S_1)(t)$ *and* $\forall_{t \in \pi_T(S_2)} : D_{O3}(t) = \pi_{D_O}(S_2)(t)$ *is the set of output dependencies,*
- $L_3 = \pi_L(S_1) \cup \pi_L(S_2)$ *is the set of labels, and*
- $A_3 \in (dom(\pi_A(S_1)) \setminus C) \cup dom(\pi_A(S_2)) \rightarrow L_3$ *such that* $\forall_{t \in dom(\pi_A(S_1)) \setminus C} : A_3(t) = \pi_A(S_1)(t)$ *and* $\forall_{t \in dom(\pi_A(S_2))} : A_3(t) = \pi_A(S_2)(t)$ *is the label assignment.*

In the next theorem we show that the result of the replacement is again a safe and sound annotated SISO-net.

Theorem 2. *Let S_1 and S_2 be two annotated SISO-nets. Let C be a non-trivial component in S_1. Let $S_3 = \texttt{replace}(S_1, C, S_2)$.*
Then:

- *S_3 is an annotated SISO-net.*
- *If S_1 is safe and sound and S_2 is safe and sound, then S_3 is safe and sound.*

Proof. See [10].

The net resulting from the replacement may contain routing transitions (AND-splits and -joins), that have become superfluous. An AND-split (AND-join) is superfluous when it is preceded (followed) by exactly one transition, i.e., the two transitions are in a sequence. The routing transition and the other transition may be combined (thus removing the routing transition) with one of the soundness preserving transformation rules, namely *aggregation*, described by van der Aalst [1].

4 Other Transformations

When generating alternative process models the selection and replacement operations are the same regardless of the change that is made. The type of change is determined by the transformation that is used. In the previous section, the parallel transformation was used to generate a more parallel process redesign. In this section we describe three other process transformation informally: *sequence*, *unfold* and *merge*.

In [10], we introduce the notion of a *layered annotated SISO-net* to allow an annotated SISO-net to contain *aggregated transitions*. An aggregated transition has an underlying sub process that divides the aggregated transition into several smaller pieces of work. Such a sub process is used for more complex transitions to give more insight in what should exactly be done. Aggregated transitions are executed just like any other transition, that is, when a resource starts the execution of an aggregated transition, (s)he performs the complete underlying sub process as a whole. A layered annotated SISO-net has a two layered process structure: 1) the upper layer is an annotated SISO-net, including aggregated transitions, 2) the lower layer contains an annotated (sub) SISO-net for each of the aggregated transitions at the upper layer. The *unfold* and *merge* transformation are performed on the layered annotated SISO-net and the *parallel* and *sequence* transformation are performed on the upper layer of the layered annotated SISO-net. The definitions and details of the transformations can be found in [10].

Sequence transformation: This transformation is the counterpart of the parallel transformation. A sequential process may be perceived as a simpler process by employees and clients, because the order of the transitions is fixed. Further, transitions are likely to be executed in the most logical way which may reduce errors. Furthermore, the synchronization that is required after the execution of transitions in parallel is not necessary in sequential processes [13].

With the sequence transformation transitions are placed in a fixed order, i.e., a sequence. As a preparatory step, the selected choice construct free [6] component is translated into an annotated graph. The *sequence* operation is performed on this graph. Afterwards, the graph is further transformed into an annotated SISO-net in a similar way as the graph resulting from the *parallel* operation (Definition 7).

Unfold transformation: With this transformation aggregated transitions (which are at the upper layer of the process) are split up into several smaller transitions. Unfolding may result in a higher run-time flexibility, because the scheduling and execution of some smaller transitions allows for more possibilities. Furthermore, a transition that is too large may be unworkable and dividing it into smaller transitions could enhance the quality realized by the involved resource(s). A drawback of smaller transitions are the longer set-up times, i.e., the time a resource spends to become familiar with a case [13].

The unfold transformation is performed by replacing the aggregated transitions present in the component by the underlying lower layer SISO-nets. A lower layer SISO-net is defined such that it always starts and ends with a transition. This simplifies the unfolding considerably, because the lower layer net can simply replace the aggregated transition without violating the bipartite structure of a SISO-net.

Merge transformation: This transformation is the counterpart of the unfold transformation. With the merging of transitions, multiple transitions are combined into one aggregated transition. Combining transitions results in the reduction of setup times, i.e., only one resource has to get acquainted with the case before the execution of the aggregated transition. An additional reason to merge transitions is the expected positive effect on the quality of the delivered work due to fewer hand-overs of the work between resources [13].

With the merge transformation several similar transitions, i.e., transitions with the same label, are combined into one aggregated transition. The transformation is started

[6] Putting choice transitions in a sequence would interfere with the choice behavior.

with the translation of an unfolded, choice construct free [7] component into an annotated graph. Edges between the nodes in the graph are created in such a way that nodes with the same label are connected while preserving all dependencies. Within this graph the largest possible groups of sub graphs with the same label are determined. The *merge* operation creates a layered annotated graph with the identified sub graphs as its lower layer graphs. Then, this layered annotated graph is translated into a layered annotated SISO-net.

5 Conclusion and Outlook

In this paper, we present a concrete method for the generation of alternative process models. It is part of our evolutionary approach to process redesign based on BPR best practices and consists of three steps: 1) the selection of a process part, 2) the transformation of this process part into an alternative part and, 3) the replacement of the original process part with the alternative part. The actual change in the process is made with process transformations. With the process transformations we aim for *extreme* changes, e.g., we place as many transitions as possible in parallel. Of course, depending on the specific process the redesign effort is made for, more conservative changes may be preferable. In a supporting tool it will be possible to adapt the transformations and generate such redesigns.

The transformations are performed with a generic set of process attributes that may be specified depending on the specific attributes of the process under consideration. Therefore, the presented transformations give much more redesign possibilities than their number may suggest. In addition, the use of more or other process attributes may also lead to new process transformations. The current set of process transformations is not exhaustive and serves as a starting point for process redesign.

Next to the further development of the theory of evolutionary process redesign, we work on the development of a tool that provides concrete support to practitioners. The ProM tool [2] is used as a framework to experiment with process mining and redesign techniques. We envision as the ultimate goal of our research the delivery of an automated redesign tool. This tool would support all steps of the approach in an "intelligent" way. By this, we mean that the tool will not only automate the various steps of the approach, but will also interact with the redesigner. Our approach is a solution that should primarily help redesign novices in finding process alternatives based on best practices. Secondly, more experienced redesigners are supported in the creation and evaluation of such alternatives in a structured and less time-consuming manner.

Acknowledgement

This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

[7] Choice transitions may be divided over several aggregated transitions and this would interfere with the choice behavior.

References

1. Aalst, W.M.P. van der: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) Application and Theory of Petri Nets 1997. LNCS, vol. 1248, pp. 407–426. Springer-Verlag, Berlin (1997)
2. Aalst, W.M.P. van der, Dongen, B.F. van, Gunther, C.W., Mans, R.S., Alves de Medeiros, A.K., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W., Weijters, A.J.M.M.: ProM 4.0: Comprehensive Support for Real Process Analysis. In: Kleijn, J., Yakovlev, A. (eds.) Petri Nets and Other Models of Concurrency. LNCS, vol. 4546, pp. 484–494. Springer-Verlag, Berlin (2007)
3. Aalst, W.M.P. van der, Bisgaard Lassen, K.: Translating Unstructured Workflow Processes to Readable BPEL: Theory and Implementation. Information and Software Technology 50(3), 131–159 (2008)
4. Aalst, W.M.P. van der, Reijers, H.A., Weijters, A.J.M.M., Dongen, B.F. van, Alves de Medeiros, A.K., Song, M., Verbeek, H.M.W.: Business Process Mining: An Industrial Application. Information Systems 32(1), 713–732 (2007)
5. Al-Mashari, M., Zairi, M.: BPR implementation process: An analysis of key success and failure factors. Business Process Management Journal 5(1), 87–112 (1999)
6. Grover, V., Jeong, S., Kettinger, W., Teng, J.: The implementation of Business Process Reengineering. Journal of Management Information Systems 12(1), 109–144 (1995)
7. Hammer, M., Champy J.: Reengineering the corporation: a manifesto for business revolution. Harper Business Editions, New York (1993)
8. Kettinger, W., Teng, J., Guha, J.: Business process change: A study of methodologies, techniques, and tools. MIS Quarterly 21(1), 55–80 (1997)
9. Netjes, M., Limam Mansar, S., Reijers, H.A., Aalst, W.M.P. van der: An Evolutionary Approach for Business Process Redesign: Towards an Intelligent System. In: Cardoso, J., Cordeiro, J., Filipe, J. (eds.) Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS 2007). pp. 484–494. INSTICC, Setubal (2007)
10. Netjes, M., Reijers, H.A., Aalst, W.M.P. van der: The Creation of Process Redesigns by Selecting, Transforming and Replacing Process Parts. BETA Working Paper Series, WP 240, Eindhoven University of Technology, Eindhoven (2008)
11. Netjes, M., Vanderfeesten, I., Reijers, H.A.: "Intelligent" Tools for Workflow Process Redesign: A Research Agenda. In: Bussler, C., Haller, A. (eds) Business Process Management Workshops: BPM 2005. LNCS, vol. 3812, pp. 444–453. Springer-Verlag, Berlin (2006)
12. Nissen, M.: Redesigning Reengineering through Measurement-Driven Inference. MIS Quarterly 22(4), 509–534 (1998)
13. Reijers, H.A.: Design and Control of Workflow Processes: Business Process Management for the Service Industry. LNCS, vol. 2617. Springer-Verlag, Berlin (2003)
14. Reijers, H.A., Limam Mansar, S.: Best Practices in Business Process Redesign: An Overview and Qualitative Evaluation of Successful Redesign Heuristics. Omega: The International Journal of Management Science 33(4), 283–306 (2005)
15. Sun, S.X., Leon Zhao, J., Nunamaker, J.F., Liu Sheng, O.R.: Formulating the Data-Flow Perspective for Business Process Management. Information Systems Research 17(4), 374–391 (2006)
16. Weber, B., Rinderle-Ma, S.B., Reichert, M.U.: Change Support in Process-Aware Information Systems - A Pattern-Based Analysis. Technical Report TR-CTIT-07-76, ISSN 1381-3625, http://eprints.eemcs.utwente.nl/11331 (2007)

Translating BPMN Models into UML Activities

María Agustina Cibrán

THALES
ThereSIS Security & Information Systems
Route Départamentale 128 - 91767 Palaiseau Cedex - France
maria-agustina.cibran@thalesgroup.com

Abstract. In business-driven development, IT architects must face the challenge of building IT solutions that are aligned with business processes in order to satisfy business requirements. To this endeavor, they must understand those business processes, which requires extensive collaboration with domain experts and analysts. Because business processes are typically expressed using dedicated modeling notations - e.g. BPMN - IT architects are forced to master their particularities. In order to support IT architects in performing the business-IT alignment, this paper presents an MDE approach that proposes the automatic translation from BPMN models to UML activity models. This translation aims at facilitating the understanding of business processes and enabling their synchronization with other UML architectural models of the pertinent IT solution. In addition, architects are relieved from having to master BPMN. This paper reports on the challenges of defining and implementing such a translation. ATL is used as the model transformation language.

Key words: BDD, BPM, BPMN, UML Activities, Model Transformations, ATL, MDA, MDE

1 Introduction

Business Process Modeling (BPM) [1] aims at representing the processes of a domain or business by creating dedicated business process models. BPM is typically performed by business analysts and managers with the objective of improving the efficiency and quality of the processes. Having explicit business process models improves clarity and accuracy of business requirements and makes the communication between domain and IT experts more efficient. A business process model typically represents interrelated business organizations and roles as well as it depicts the flow or progression of activities - tasks and subprocesses - each of which representing the work of a person, an internal system, or the process of a partner company towards some business goal. Nowadays, one of the most popular standard languages for the representation of business process models is the Business Process Modeling Notation (BPMN) [2].

In Business-Driven Development (BDD) [3] the goal is to develop IT solutions that directly satisfy business requirements and needs. Because the starting step for BDD is the creation of business process models, the alignment of business processes and IT solutions becomes crucial. This alignment implies the need for adapting the methodology used to derive the IT solution, to be able to use those business process models as input artifacts to the design and development phases of the software development life cycle.

However, aligning business processes to existing software methodologies is not an easy task: it does not only imply the need for understanding the business goals and expectations but it also requires distilling the parts of the business processes that need to be automated - vs. the parts that are manually performed - and clearly defining and realizing functional and non-functional requirements for those automated parts. These steps typically imply the need for extensive collaboration and communication efforts between domain experts, business and functional analysts, and IT architects. Moreover, in the effort of understanding business processes, IT architects are confronted with the particularities - i.e. notation and semantics - of the adopted business process modeling notation. Also, IT architects are forced to reason in terms of *process-centric* abstractions, which do not necessarily fit into the architectural abstractions, typically centered around concepts such as *object*, *component* and *service*.

In this article, an approach for BDD is proposed which relies on principles of Model-Driven Engineering (MDE). This approach aims at facilitating the alignment between business processes and IT solutions by means of automating the translation from BPMN models to UML activity models. Because UML has become the de-facto industry standard for modeling software systems, architectural models of contemporary industrial solutions typically rely on its formalisms and modeling support. We claim that re-expressing - in an automated way - the BPMN business process models in the form of UML activity models can (i) help improve IT architects' understanding of those business processes, (ii) enable the synchronization of those business processes with other architectural models expressed in UML, (iii) relief IT architects from having to master BPMN's notation and formalisms. In addition and contrary to UML, no standard BPMN metamodel exists today which complicates BPMN's practical adoption in the context of MDE for BDD. Translating business processes to UML can enable the subsequent definition of model transformations that rely on the UML metamodel.

The main contributions of this approach are: (i) the definition of a mapping from BPMN concepts to corresponding UML activity elements, and (ii) the implementation of this mapping in the form of a model-to-model transformation using ATL [8]. We observe that defining a conceptual mapping between these two languages is not trivial - due to their different purposes and semantics. Thus, as extra contributions, advanced mapping scenarios are identified (iii) and ATL's features are evaluated against the realization of those mapping scenarios (iv).

This paper is organized as follows: Section 2 introduces BPMN and UML activity models, Section 3 presents the proposed conceptual mapping between the two, describing advanced mapping scenarios for which ATL solutions are proposed in Section 4. Sections 5 and 6 present related and future work respectively, and conclusions are summarized in Section 7.

2 Selected Approaches to Business Process Modeling

The Business Process Modeling Notation (BPMN) is a standard graphical notation - initially developed by the Business Process Management Initiative (BPMI) and currently being maintained by the Object Management Group (OMG) - for the representation of business process models. BPMN was developed with the aim of having a notation that is understandable by all business stakeholders who typically manage and monitor those processes and to facilitate the link towards a technical implementation of the process. BPMN graphical models typically depict the control flow of activities, the organizations and roles involved and the message exchange between them. Among the most common BPMN concepts we can mention *activities* (e.g. *tasks* and *subprocesses*) which are the units of behavior, *swimlanes* (e.g. *pools* and *lanes*) that help organize activities into roles and interacting parties involved in the process, *events* that occur during the process and *gateways* for expressing sequence flow. For more details on BPMN, the interested reader is referred to [2]. The choice of BPMN as source business process language is driven by current needs from THALES customers manifesting their interest for modeling business interactions explicitly.

UML provides support for process modeling in the form of *Activity* diagrams or models. Although this support is commonly known as UML *Activity Diagrams (ADs)*, in this article we adopt the name of UML *activities* or *activity models* to put the focus on the modeling capabilities and not on the graphical details. With respect to the offered UML support, the concept of *activity* can be used to depict the overall behavior of a system. In addition, *actions* encapsulate business behavior and can be used to either represent business activities or business events; *control nodes* are used to depict typical control flow patterns. The suitability of UML activities for business process modeling has been proven by existing approaches [14, 15, 12]. Existing work also identifies limitations of this modeling language when it comes to representing resource-related or organizational aspects [14]. More information on UML activities can be found in [6] (Section 12, p.297).

It is important to note that while BPMN focuses on business processes, UML focuses on software design and therefore the two are not competing notations but allow to represent different views on systems [5].

3 A Conceptual Mapping from BPMN to UML Activities

This section reports on the proposed conceptual mapping from BPMN models to UML activity models. This mapping considers BPMN elements at the source side - as described in the adopted BPMN specification (v1.0) [2] - and UML elements at the target side, in particular the portion of the UML metamodel specification (v.2.1.2) corresponding to *activities* [6] (Section 12, p.297). The reason why the BPMN specification is used is the current lack of a standard BPMN metamodel[1]. Note that only a portion of the BPMN specification is considered in this mapping, namely common BPMN elements: *business process diagram*, *swimlanes*, *activities*, *sequence* and *message flows*, *gateways* and *events*. However, more advanced BPMN features such as transactions, multiple instances, exception handling and data flow have not been addressed yet. Mapping solutions for these features would require the investigation of more advanced solutions that rely on the identification and automatic detection of sophisticated patterns. The proposed mapping aims at preserving the same behavior, granularity and level of abstraction expressed in the source model. Thus, the target activity model aims at being an equivalent representation of the original BPMN model. Note that due to limited space, the complete mapping is not shown here. Instead, the rest of this section presents representative mapping scenarios which are of interest from the point of view of model transformation.

3.1 One-to-One Mappings

A natural approach for the definition of this mapping is to define, for each of the BPMN elements, a single UML correspondent which complies with the semantics of that BPMN element. However, a one-to-one mapping can only be established for those elements for which a direct counterpart - a *UML Activity* metaclass - exists. Figure 1 gives examples of one-to-one mappings. For example, a Pool in BPMN represents a partition of the process which concern a participant in the process, such as a business entity. The *UML Activity* metaclass named *ActivityPartition* allows to group actions that have some characteristic in common. An equivalence can be drawn between these two concepts. The same analysis is done for the other examples in the figure.

BPMN element	UML Activity metaclass
Pool	ActivityPartition
Lane	ActivityPartition
Activity	ActivityNode
Task	OpaqueAction
Subprocess	StructuredActivityNode
SequenceFlow	ControlFlow

Fig. 1. Examples of mappings denoting a single UML counterpart for each individual BPMN element

When defining these mappings we observe that for some BPMN elements it is not possible to find a unique distinct *UML Activity* metaclass that denotes ex-

[1] The Business Process Definition Metamodel (BPDM) aims at addressing this issue. More information on BPDM can be found at http://modeldriven.org/web/bpdm.

actly its semantics. This is the case for e.g. the BPMN *Pool* and *Lane* elements, which represent participants and roles respectively. In the UML Activity metamodel, only the concept of partition is available, not distinguishing between the two different ways of partitioning a model. Thus, as a workaround, we are forced to map BPMN pools and lanes to the same UML concept of activity partition - embodied by the *ActivityPartition* metaclass.

The same situation is observed in the case of BPMN events. One of the main differences between BPMN and more traditional modeling notations is that BPMN provides extensive means to describe event-driven behavior. We observe however that some - very specific - types of events cannot have a straightforward mapping at the *UML Activity* side, as UML's support for events is much more limited than that of BPMN. Thus for example, the event types *error*, *cancel*, *compensation*, *link*, *terminate* and *multiple*, each of them with specific semantics, are represented at the UML side by the same metaclass *FlowFinalNode*.

The reason for this mapping mismatch lies on the different focus and level of expressivity of the languages. As a consequence, elements with a more specific semantics get mapped to elements with more generic semantics. Note that these mappings do not ensure the preservation of the full BPMN semantics. To overcome this limitation, an approach based on the semantical enrichment of UML - by means for instance of the definition of a UML profile - could be explored. This is subject of future work.

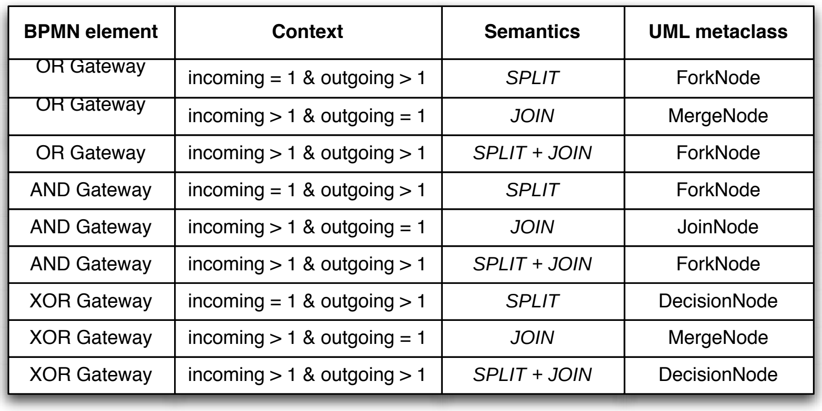
3.2 Mapping *Rich* BPMN Elements

Some BPMN elements have *rich* semantics because they embody different meanings in the same single concept. This is the case for example of BPMN events. For example, a BPMN *start message event* combines the semantics of *starting the process* with *accepting a message*. The UML Activity metamodel however does not offer a metaclass that combines these two meanings. The UML metaclass *InitialNode* denotes a control node at which the flow starts when the activity is invoked; in turn, the metaclass *AcceptCallAction* denotes an accept event action representing the receipt of a synchronous call request. Thus, to overcome this mismatch we propose a solution that involves combining many *UML Activity* elements for the representation of a single BPMN element. In our example, we can define the counterpart of a *start message event* as a sequence composed of an *InitialNode* followed by an *AcceptCallAction*.

3.3 Mapping *Overloaded* BPMN Elements

We observe that certain BPMN elements are *overloaded* since they can have different meanings according to the way they are used in a model. This means that their semantics cannot be analyzed by looking at each of those elements in isolation but also at their relations with other model elements.

This is the case for example of BPMN *gateways*. In BPMN, the context in which the gateway is used determines different interpretations of the gateway and thus a different mapping to UML *control nodes*. According to the BPMN specification, a gateway can have multiple incoming and outgoing sequence flows which determine a different role of that gateway as part of the process. For example, an *OR gateway* can play the role of a *fork node* when it is used to split the control flow (i.e. multiple outgoing sequence flows) or it can play the role of a *merge node* when it is used to join different control flows (i.e. multiple incoming sequence flows). A similar analysis can be done for the *AND* and the *XOR gateways*. Figure 2 summarizes the different semantical interpretations of BPMN gateways and the UML correspondent for each contextual case. As a conclusion, the mapping of overloaded BPMN elements differs according to contextual information.



BPMN element	Context	Semantics	UML metaclass
OR Gateway	incoming = 1 & outgoing > 1	*SPLIT*	ForkNode
OR Gateway	incoming > 1 & outgoing = 1	*JOIN*	MergeNode
OR Gateway	incoming > 1 & outgoing > 1	*SPLIT + JOIN*	ForkNode
AND Gateway	incoming = 1 & outgoing > 1	*SPLIT*	ForkNode
AND Gateway	incoming > 1 & outgoing = 1	*JOIN*	JoinNode
AND Gateway	incoming > 1 & outgoing > 1	*SPLIT + JOIN*	ForkNode
XOR Gateway	incoming = 1 & outgoing > 1	*SPLIT*	DecisionNode
XOR Gateway	incoming > 1 & outgoing = 1	*JOIN*	MergeNode
XOR Gateway	incoming > 1 & outgoing > 1	*SPLIT + JOIN*	DecisionNode

Fig. 2. Contextual interpretation of BPMN gateways and their mapping to UML control nodes

4 Automating the Mapping using ATL

THALES is involved in the MODELPLEX project[2] the aim of which is to demonstrate the value of using MDE technologies for tackling the increasing complexity of software systems. One of the goals of THALES as industrial partner in MODELPLEX is to assess the maturity and suitability of MDE tools - including model transformation languages - for the engineering of industrial solutions for complex systems. In this context, THALES implemented a model transformation of the conceptual mapping proposed in Section 3. For this implementation, the BPMN metamodel which is implemented as part of the Eclipse SOA Tools Platform project [7] was chosen.

[2] More information on MODELPLEX can be found at http://www.modelplex.org

The ATLAS Transformation Language (ATL) [8] is a model transformation solution - including a language and development environment - for the implementation of model transformations. ATL is developed by the ATLAS Group (INRIA & LINA) and it is now one of the major components of the M2M Eclipse project. ATL is hybrid because it allows mixing two programming styles, declarative and imperative. ATL rules define how source model elements are matched and navigated to create and initialize the elements of the target models. More information on ATL can be found in [8].

In the rest of this section, we report on the experiences of using ATL for the implementation of the translation from BPMN to *UML Activities*. Solutions are proposed for the mapping cases described in Section 3. Sections 4.1, 4.2 and 4.3 revisit the mapping scenarios described in Sections 3.1, 3.2 and 3.3, showing how ATL can be used to realize them, as well as they identify challenges.

4.1 One-to-One Mappings

Implementing one-to-one mappings in ATL is done by defining matching rules that receive as input - i.e. *from* clause - a single BPMN element and produce as output - i.e. *to* clause - a single UML element. This implementation is straightforward.

With respect to the challenge posed by mapping different BPMN elements to the same UML element, as this is a conceptual mismatch, a solution cannot be found at the technological level. In order to avoid loose of information, and preserve the full semantics of BPMN elements, two solutions are envisaged (not explored yet): i) define complex UML patterns, ii) extend UML's semantics by using a profile approach that allows annotating the target model elements with extra semantical information. Exploring these alternatives are subject of future work.

4.2 Single Source to Composed Target

ATL matched rules allow specifying how target model elements must be generated from source model elements. To this end, a matched rule defines which source model element must be matched (*from* clause), the number and type of the generated target model elements (*to* clause) and the way these target model elements must be initialized from the matched source elements (body of the rule).

Many different target model elements can be generated for a single source model element, feature that can be used to realize the translation of a single BPMN element to many UML elements. These target elements are linked together in a sequence by setting their attributes accordingly. An extract of this solution for the example of the *end message event* is shown below:

```
rule EndMessageEvent {
  from
    i: bpmn!Activity (i.activityType = #EventEndMessage)
  to
    ssa: UML!SendSignalAction (
      ...
      target <- inputpin ),
    inputpin: UML!InputPin (...),
    cf: UML!ControlFlow (
      ...
      source <- ssa,
      target <- ffn ),
    ffn: UML! FlowFinalNode(...)
}
```

This solution poses challenges at the implementation level: translating to several target elements does not only have a local impact - in the scope of the involved transformation rule - but it also impacts other transformation rules which need to pick the right element among the many generate target elements. To illustrate this impact, consider a BPMN sequence flow *sflow1* that has as target element an end message event *event1*. According to the mapping (Section 3), *sflow1* translates to a UML control flow *cflow1* whereas *event1* translates to a sequence composed of a send signal action *ssaction1*, a control flow *cflow2*, and a flow final node *ffnode1*. Thus, the transformation rule in charge of translating *sflow1* needs to assure that the generated *cflow1* has as target element the same send signal action *ssaction1*. This example shows that when single sources are translated to composed targets, transformation rules become dependent of each other, reducing this way the modularity of the implementation. Using ATL, these dependencies can be tackled by adding code in the dependent rules. This is however not ideal, as the realization of the dependency results in crosscutting code, violating separation of concerns. More advanced support is needed in transformation languages to better realize separation of concerns.

4.3 Single Source, Different Contextual Interpretations

In Section 3.3, the situation in which mapping a BPMN element depends on contextual information was described. ATL allows to implement these extra contextual conditions as part of the *from* clause of rule specifications. Thus for example, implementing the mapping of *OR gateways* can be done as shown below. Helpers are used in ATL to implement those conditions in modules that can be invoked from rules.

```
helper context bpmn!Activity def: usedAsSplit(): Boolean =
  self.incomingEdges -> size() = 1 and self.outgoingEdges -> size() > 1;
helper context bpmn!Activity def: usedAsJoin(): Boolean =
  self.incomingEdges -> size() > 1 and self.outgoingEdges -> size() = 1;
helper context bpmn!Activity def: usedAsSplitAndJoin(): Boolean =
  self.incomingEdges -> size() > 1 and self.outgoingEdges -> size() > 1;
rule ORGatewaySplit {
  from i: bpmn!Activity (i.activityType = #GatewayDataBasedInclusive
                         and i.usedAsSplit())
  to o: UML!ForkNode(...)}
rule ORGatewayJoin {
  from i: bpmn!Activity (i.activityType = #GatewayDataBasedInclusive
                         and i.usedAsJoin())
  to o: UML!MergeNode(...)}
rule ORGatewaySplitAndJoin {
  from i: bpmn!Activity (i.activityType = #GatewayDataBasedInclusive
                        and i.usedAsSplitAndJoin())
  to o: UML!ForkNode(...)}
```

Adding contextual conditions to transformation rules as a solution for distinguishing between different interpretations of source elements is suitable when only a few conditions exists and when those conditions are based on information that is directly accessible from the source element. In the example shown above, all incoming and outgoing edges are accessible from the OR gateway. However, when more global context that is not directly accessible from the source element needs to be inspected, then this solution falls short. More advanced support is needed in transformation languages to better deal with inspection of global context.

In this section we reported on the experience of implementing the automatic translation from BPMN to UML Activities using ATL. A complete translation has been implemented and tested with various real-world models from complex domains being considered in the THALES' MODELPLEX case study.

5 Related Work

State-of-the-art on BDD typically aims at translating business processes directly into executable implementations. Today, the Business Process Execution Language (BPEL) is the standard for representing executable business process behavior based on Web services [9], and thus it is a natural candidate for the implementation of workflow-based business processes. However, the translation of BPMN to BPEL is far from trivial due to conceptual mismatch. Some approaches exist today that analyze this mismatch and realize the automatic translation for certain scenarios [10].

However, while some approaches pursue a one-step translation to executable processes that can easily be triggered by business analysts, others consider this goal as 'one of the fallacies of the business process management' [11]. The reasons for this disbelief is that it is observed that most business processes typically require IT involvement in order to derive a suitable implementation. Moreover, having a direct translation from business processes to executable processes also assumes the existence of a one-to-one mapping between the abstractions of business processes on the one hand (e.g. tasks, subprocesses) and their implementing Web services of the BPEL process on the other hand, creating a tight conceptual coupling between the business and implementation spaces. The limitations of the one-step translation motivate the need for a more advanced approach that considers having an architectural layer in between the two.

The mapping of BPMN and UML activities has been explored by other existing approaches. In [12] a conceptual mapping between these two languages based on the workflow patterns (as identified by [13]) is proposed. This work shows how two graphical process modeling notations BPMN and UML can be used to represent workflow patterns. The notations are evaluated with respect to technical ability and readability and several alternative - graphical - solutions for each pattern are proposed. Thus the focus is put on the expressivity of the languages for realizing the pattern, rather than on how the solutions can be mapped to each other (semi-)automatically. In [13, 14, 15], Van der Aalst et al. use the workflow patterns as an analysis framework. They provide a survey of the expressive power and features of process modeling languages that are suitable for the realization of the patterns - among them BPMN and UML - identifying similarities and differences between them. However, the actual automatic mapping between BPMN and UML is not investigated.

In [16], Murzek et al. point out that transformation languages are too generic to be suitable for solving more domain-specific transformation issues. They identify a list of problems when transforming business process models expressed in different languages, among them BPMN and UML activity diagrams and where the focus is put on basic control flow patterns. The authors also identify challenges in the implementation of automatic transformations between business process languages and use ATL as the transformation language.

In [17], Redding et al. explore a bottom-up approach for the transformation of object-oriented models to process-oriented models. The focus is put on control flow logic. The authors point out their interest on exploring the challenge of a top-down approach, as presented in this paper.

6 Future Work

Possible future directions of work include the investigation of solutions to overcome the challenge of translating - automatically or semi-automatically - entire workflow patterns, reusing conceptual solutions already proposed in state-of-the-art work [13, 14]. Realizing this (semi-)automatic support is not a trivial task, as it involves first of all the automatic identification of patterns, which in turn requires solving the complex issue of delimiting the pertinent global context involved in the patterns. Also, from an implementation point of view, current transformation languages only allow the manipulation of individual model elements in the transformation rules. This is not enough to be able to modularize groups of elements involved in complex patterns.

Another direction of future work is the investigation of solutions to overcome global semantical issues. For example, in both BPMN and UML activities, it is possible to define multiple start nodes for one process model. However, the semantics of initial nodes is different in both languages, as pointed out by [16]. In activity models, the start of the process implies the trigger of all initial nodes, whereas in BPMN, only the triggering of one start event can trigger the beginning of the process. Realizing this solution in ATL would imply being able to capture and analyze global context. The more global the context, the more dependencies there are between the transformation rules, which are more difficult to manage. More advanced transformation language features need to be investigated in order to facilitate this task. Moreover, in order to avoid loosing information at transformation time and preserve the full semantics of BPMN elements, more advanced solutions that rely on complex patterns and UML profiles for semantical enrichment are to be explored.

The BPDM aims at defining a common metamodel for the various process notations, methodologies and standards, among them BPMN. Although this is a very promising initiative, it is still under development. Once available, the approach presented in this paper can be adapted to consider BPDM as the source metamodel.

Ongoing work is the definition of a MDE methodology for BDD. This methodology relies on the use of UML activity models as intermediate models in between BPMN business process models and core architectural models of the IT solution - i.e. Platform Independent Model (PIM) and Platform Specific Model (PSM) in the context of Model-Driven Architecture (MDA) [4]. The goal is to use model transformations to support the entire software development process, moving from BPMN models towards architectural models and finally code artifacts. A special focus is put on the synchronization and traceability between models at the same and different levels of abstraction.

7 Conclusions

In this paper, an approach that facilitates the alignment between business processes and IT solutions was presented. To this end, a model transformation was proposed - including a conceptual mapping and implementation - to bridge from a business process model expressed in BPMN to an activity model expressed in UML. This paper also reported on the experiences in pursuing this automatic translation using ATL as the transformation language. The generated UML activity model serves as a starting point for the definition of the core architectural steps. This work shows that the implementation of such a translation using contemporary model transformation languages such as ATL is possible to certain extent. However, more challenging mapping scenarios were identified which motivate the need for more advanced model transformation support.

Re-expressing business processes in UML - the current de-facto modeling language typically familiar to most IT architects - can help improve IT architects' understanding of those business processes, relieving them from having to master BPMN. Moreover, implementing this translation in the form of a model-to-model transformation allows the encapsulation of expert and reusable transformation knowledge and reliefs the need for manual work. In addition, in a typical architectural scenario where many interrelated UML models coexist, having business processes expressed in the form of UML activities can enable the synchronization and traceability between those different models.

Acknowledgments. This work has been carried out in the context of the MODELPLEX project contract number 034081 (co-funded by the European Commission under the IST Sixth Framework Programme). Thanks to Dhouha Ayed, Quentin Dallons, Didier Boulet, Ivo Bettens, Jérome Le Noir and Christophe Cubat for their interesting comments on the work presented in this article.

References

1. Havey M.: What Is Business Process Modeling. Article at www.onjava.com (2005), http://www.onjava.com/pub/a/onjava/2005/07/20/businessprocessmodeling.html
2. OMG, Business Process Modeling Notation Specification v1.0. OMG Final Adopted Specification. February 2006. Available via: http://www.bpmn.org/
3. Mitra T.: Business-Driven Development. Article at IBM DeveloperWorks. Available at: http://www-128.ibm.com/developerworks/webservices/library/ws-bdd/ (2005)
4. OMG, http://www.omg.org/mda/
5. BPMI.org, Business Process Modeling Notation (BPMN) Information, FAQs at http://www.bpmn.org/Documents/FAQ.htm (2004)
6. OMG, Unified Modeling Language (UML), Version 2.1.2
7. Eclipse, SOA Tools Platform Project, BPMN Modeller, http://www.eclipse.org/stp/bpmn/
8. ATLAS Group (INRIA & LINA), ATLAS Transformation Language, http://www.eclipse.org/m2m/atl/
9. OASIS, Web Services Business Process Execution Language Version 2.0 (2007)
10. Ouyang C., van der Aalst W., Dumas M. and ter Hofstede A. and Mendling J.: From Business Process Models to Process-Oriented Software Systems. In ACM Transactions on Software Engineering Methodology (to appear). Preliminary version available at: http://eprints.qut.edu.au/archive/00005266/01/5266.pdf
11. Dubray J.: The Seven Fallacies of Business Process Execution. Article at InfoQ. Available at: http://www.infoq.com/articles/seven-fallacies-of-bpm (2007)
12. White S.: Process Modeling Notations and Workflow Patterns. IBM. Available at http://www.bpmn.org/ (2004)
13. Wohed P., van der Aalst W., Dumas M., ter Hofstede A. and Russell N.: On the Suitability of BPMN for Business Process Modeling. Proceedings of the 4th International Conference on Business Process Management (BPM). Austria. Springer Verlag (2006)
14. Russell N., van der Aalst W., ter Hofstede A. and Wohed P.: On the Suitability of UML 2.0 Activity Diagrams for Business Process Modeling. Proceedings of the 3rd Asia-Pacific conference on Conceptual Modeling. Australian Computer Society, Inc. (2006)
15. Wohed P., van der Aalst W., Dumas M., ter Hofstede A. and Russell N., Pattern-based Analysis of UML Activity Diagrams. Proceedings of the 25th International Conference on Conceptual Modeling. Springer (2005)
16. Murzek M., Kramler G.,: Business Process Model Transformation Issues - The Top 7 Adversaries Encountered at Defining Model Transformations. ICEIS, vol. 3, pp. 144-151, (2007)
17. Redding G., Dumas M., ter Hofstede A., Iordachescu A.: Transforming Object-Oriented Models to Process-Oriented Models. Business Process Management Workshops (2007)

Author Index