Structural Patterns for the Transformation of Business Process Models*

Marion Murzek
Women's Postgraduate College for Internet Technologies (WIT), Institute for Software Technology and Interactive Systems, Vienna University of Technology, Austria
murzek@wit.tuwien.ac.at

Gerhard Kramler
Business Informatics Group (BIG), Institute for Software Technology and Interactive Systems, Vienna University of Technology, Austria
kramler@big.tuwien.ac.at

Elke Michlmayr
Women's Postgraduate College for Internet Technologies (WIT), Institute for Software Technology and Interactive Systems, Vienna University of Technology, Austria
michlmayr@wit.tuwien.ac.at

Abstract

Due to company mergers and business to business interoperability, there is a need for model transformations in the area of business process modeling to facilitate scenarios like model integration and model synchronization. General model transformation approaches do not consider the special properties of business process models and horizontal transformation scenarios. Therefore, we propose a model transformation approach based on domain-specific patterns which are applied for analyzing business process models in a precise way. This approach facilitates the definition of business process model transformations, which can be easily adapted to different business process modeling languages and specific transformation problems. At the same time it supports the intuitive understanding of the domain-experts in business process modeling.

1. Introduction

As companies discovered the benefits of Business Process Modeling (BPM), the use of Business Process (BP) models moved from a "luxury article" to an "everyday necessity" in the last years. Meanwhile many companies own thousands of models which describe their business. Since business changes over the years, e.g., business to business interoperability came up with new inventions in communication and companies merge with others, there arises a need to keep existing business models up-to-date and to synchronize or translate them into a contemporary BPM language. To facilitate these scenarios, a model transformation technique for BP models is needed.

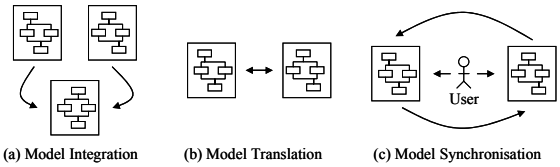

Figure 1. Different Model Transformation Scenarios

According to these scenarios, model transformation in the area of BPM can be classified into three different kinds as illustrated in Fig. 1: model integration, model translation and model synchronisation. In the case of (a) *Model Integration*, two or more models which conform to different BPM languages are merged into one new model conforming to another BPMlanguage. In case of (b) *Model Translation*, all models conforming to one BPM language are translated into models conforming to another BPM language once. In case of (c) *Model Synchronisation*, the aim is to keep pairs of models after user changes up-to-date. A suitable model transformation approach for BP models should support all of these three scenarios.

* This research has partly been funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

Inspired by the vision of the model driven architecture (MDA) [1], model transformations are currently a hot research topic. The main research interest lies on vertical transformations, i.e., transformations between models of different levels of abstraction such as platform independent models (PIMs), platform specific models (PSMs) and code. Another important aspect are horizontal transformations, where models are translated into other models on the same level of abstraction, e.g. on the PIM level. For models in the area of BPM, this aspect is vital for integrating, synchronizing or transforming models of different BPM languages.

Current techniques or specifications used for defining model transformations, such as ATL [2] or QVT [3], operate at the level of metamodel elements. Therefore, transformation definitions can become quite difficult when direct 1:1 mappings between metamodel elements are not sufficient for specifying a transformation.

For instance, consider transforming the source business process model depicted in Fig. 2 into the target business process model in Fig. 3. In the BPM language of the source model, there is no explicit model element to denote the join of alternative threads, whereas in the target modeling language there is. Moreover, the semantics of multiple arcs depends on the context, i.e., on the origin of these arcs. In the given example, the arcs belong to two different threads, originating in a Split element. However, if they had been originating from a Decision element, their semantics and hence the desired target model would be different.

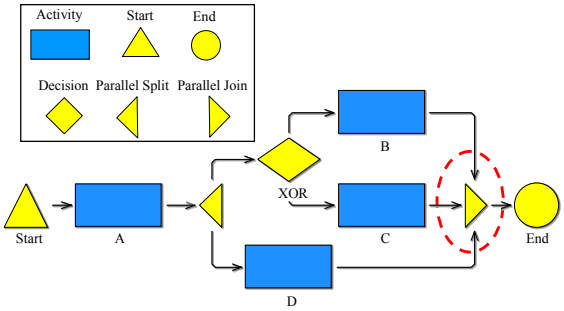

Figure 2. Source business process model in ADONIS® notation [4]

Implementing such non-trivial transformations requires defining complex queries upon the source model, since the semantics and the required target element cannot be decided by simply considering the source element and its local properties (see Fig.2 and Fig.3). This solution is not satisfying, because complex queries are error-prone and difficult to maintain and reuse.

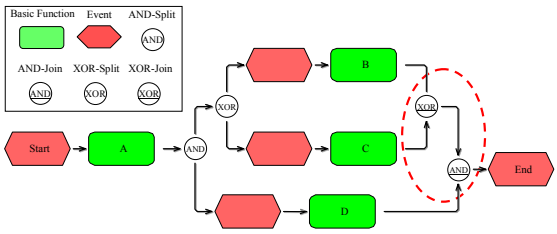

Figure 3. Target business process model in EPC notation [5]

Another problem regarding BP model transformations arises from the fact that model transformation scenarios are stemming from the field of Software Engineering. This means that the content of the models and the transformation technique belong to the same field, namely Software Development. In case of transforming models in a different domain - in our case BPM - the knowledge about the content of the models belongs to another area (BPM) than the technique (Software Development). Due to this problem, the definition of model transformations for BP models is a hard job. Misunderstandings and exhausting discussions between the domain-experts (BPM) and the technical experts complicate and protract the process of model transformation definitions.

To address these problems, we are currently developing a framework for domain-specific model transformations in the area of BPM [6]. It comprehends a method to define a model transformation definition in well structured steps based on business aspects [7, 8]. Furthermore, the framework will offer BP model patterns to facilitate the definition of BP model transformation scenarios, on the one hand to solve non-trivial transformation requirements, and on the other hand to support the intuitive understanding of the domain-experts (BPM).

In this work we concentrate on the definition and detection of the BP model patterns. We focus on the structural patterns that can be found within the control flow of a BP model. Moreover, we show how the defined patterns assist the transformation definition for BP models. The contribution of this paper is twofold. First, we thoroughly investigate the control flow aspect of BP models as described by the workflow patterns [9]. As we will show, workflow patterns are not sufficient for certain model transformations. Therefore, we define higher-level patterns based on the existing workflow patterns in order to detect structural

transformation patterns in business process models. Second, we present a case study that shows how the patterns are used in the context of BP model transformations. The case study illustrates the intuitive use of the patterns and their advantages compared to general model transformation approaches.

The remainder of the paper is structured as follows. Section 2 gives an overview of different transformation definitions, business aspects and workflow patterns. In Section 3 the transformation patterns for the control aspect are introduced. Section 4 provides a case study which illustrates the use of the patterns for the transformation of BP models. In Section 5 related work is discussed. Finally Section 6 provides the conclusion and gives an outlook on further research directions.

2. Pattern-oriented Transformation of Business Process Models

In this section we discuss the problems of metamodel-oriented transformation definitions and semantic abstraction to solve these problems. Furthermore, the applicability of the workflow patterns which are the basis of our work is discussed.

2.1. Problems of Metamodel-oriented Transformation Definition in defining model transformations for BP models

Established model transformation techniques, such as QVT [3] or ATL [2] are metamodel-oriented. That means, that the definition of model transformation scenarios is based on the correspondences of the elements in two or sometimes more different metamodels. This concept covers the local correspondences of each element which is used for copying, splitting and merging elements.

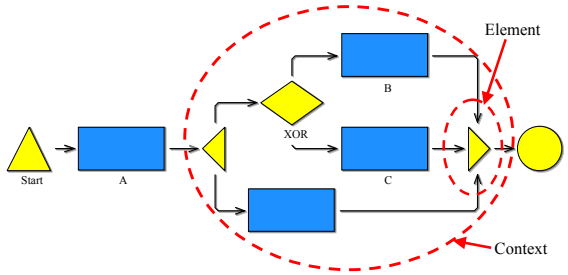

Figure 4. Local view and context view

For transformation correspondences which require knowledge of the context in which an element is used, e.g. deleting elements or integrating new elements, these techniques offer additional imperative programming concepts which, however, lead to complex transformation definitions.

To reduce the complexity of defining context-dependent correspondences imperatively, we introduce structural patterns for the domain of BP models. These patterns make it possible to relate the elements of two BPM languages considering their context. So we can make use of the ”bigger picture” as shown in Fig. 4.

2.2. Patterns to Simplify Transformation Definition

As opposed to ad-hoc and metamodel-oriented transformation definitions, we propose to use patterns as basis for transformation definitions. Patterns abstract from individual metamodel elements to represent semantic concepts. The intention of using these higher-level abstractions is to simplify transformation definition and increase reusability.

Patterns allow for a divide and conquer approach to model transformation. The transformation definition can be divided into pattern matching, i.e., analyzing the source model and identifying pattern occurrences, and pattern instantiation, i.e., synthesizing the target model such that the same patterns as in the source model appear. Defining a model transformation in two steps promises simplification of each step and re-use of steps when transformations between multiple BPM languages need to be defined.

We presume that this approach copes with the fine grained heterogeneities of modeling languages, because differences in particular details can be abstracted as higher-level patterns. Of course, a transformation can only be defined based on patterns that can be identified and generated using the source and target modeling languages, respectively. Furthermore, compared to metamodel-oriented transformations using patterns is more of a top-down approach. This will help to quickly identify cases that can be transformed (i.e., identified and generated), and cases that cannot.

2.3. Workflow Patterns

The behavior aspect is the main aspect of process modeling. It is used to describe how the business process proceeds from its beginning to its end, and the order in which its tasks1 should be executed. This aspect is generally seen as an integrating aspect, meaning that in BP models all other aspects, such as

informational and organization aspect, are integrated by the behavior aspect [7].

Kindler et al. [8] made a more precise differentiation and divided the behavior aspect into two parts. The integral aspect covering only the tasks and the control flow aspect which consists of the concepts used to model the succession of the tasks. Only the tasks are used to integrate other aspects.

Since the integral and control flow aspects are the ”heart of a process”, we focus on them and leave other business aspects to future work. Of particular interest is the control flow aspect, because it is very rich in concepts and variability, as will be discussed in the following. There is a huge body of work relevant to the control flow aspect, most 1Throughout this paper, we use the terms task and activity interchangeably prominently the catalogue of workflow patterns (WF patterns) [9] that has been used for analyzing the features and expressiveness of a set of workflow and BPM languages.

There are various publications which analyze distinct BPM languages regarding their support for the WF patterns. White [10] inspected the Business Process Management Notation (BPMN) and the UML 2.0 Activity Diagram (AD) to find out how far the WF patterns can be represented in these languages. The AD has also been examined in [11]. With the aim of assessing the weaknesses and strengths of the AD and its coverage of business process modeling problems. In [12], EPCs have been analyzed with regard to the representation of the WF patterns. Additionally, an extended EPC has been developed which covers all WF patterns.

All of the above mentioned publications conclude that not all of the WF patterns are supported in the various BPM languages. But the basic control flow patterns (WF patterns 1-5), the advanced branching and synchronization patterns (WF patterns 6-9) and the structural patterns (WF patterns 10 and 11) are supported by every BPM language which has been inspected.

Based on these contributions and the fact that the control flow aspect is similar in BPM and Workflow Models, we used the WF patterns as the starting point for our work. Furthermore we inspected four different BPM languages, ADONIS® Standard Modeling Method (ADONIS®) [4], Event-driven Process Chains (EPC) [5], Business Process Definition Notation (BPMN) [13] and UML Activity Diagrams (AD) [14] to find out if WF patterns are sufficient to define model transformations on them.

Soon it became apparent that three of the four BPM languages we inspected lack formal semantics, i.e., there is no executable semantics defined. Furthermore WF patterns are realized differently in these BPM languages, i.e., some languages use explicit join elements (EPC, BPMN) whereas others do not (ADONIS®). In case of AD the decision is left to the designer of a model. This potentially leads to ambiguities that have to be considered in transformation definitions. Due to the missing executable semantics, we restricted our work to the WF patterns 1-11 which cover the control flow aspect of each inspected BPM language. When testing the applicability of the selected WF patterns we found out that they are not sufficient for our purposes, but useful as a generic basis for defining higher-level patterns independent of particular BPM languages.

3. Structural Transformation Patterns

In this section we propose a set of transformation patterns for the control flow aspect that cover the majority of cases occurring in BP models. The transformation patterns have been developed based on the WF patterns and on the analysis of existing BP models of different BPM languages.

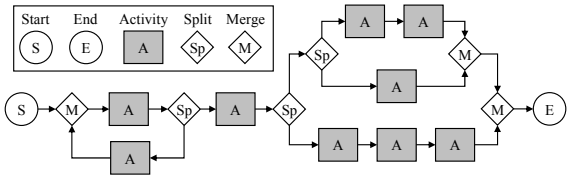

Figure 5. Block-structured Process Model

The transformation patterns are introduced by means of block-structured and graph-structured models. For the initial definition of the patterns, a block-structured model (see Fig. 5) is used. This kind of representation seems to be most suitable for an intuitive understanding of the patterns. As BPM languages also support the creation of graph-structured models, i.e., models that are not restricted to a block-structure, the difficulties which arise in pattern matching are discussed by means of graph-structured models (see Fig.7 and Fig.8).

3.1. Patterns in block-structured models

As the name implies, block-structured models are composed of blocks of elements. Blocks could be nested within other blocks but it is not allowed that blocks partly overlap or that blocks include an end element. This eases the identification and definition of the transformation patterns, because each kind of block

can be identified as a pattern. In the following, seven transformation patterns are proposed (see Fig. 6).

The *Start Pattern* consists of a single element found in the Model in Fig.5 which marks the begin of a process model.

The *End Pattern* consists of a single element which marks the end of a process model.

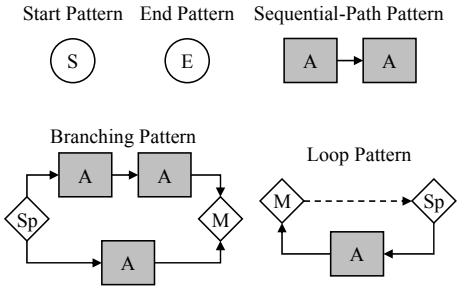

Figure 6. Structural Transformation Patterns

The *Sequential-Path Pattern* is composed of one or more directly successive activities. For this pattern, WF pattern 1 (Sequence) has been extended. To enable uniform transformation definitions, a single activity is also recognized as Sequential-Path Pattern.

The *Branching Pattern* is a part of a process model between the point where the control flow is split into multiple flows, and the point where the control flow is merged from multiple into one successive flow (see Fig. 6). In contrast to the patterns described so far, a Branching Pattern contains nested patterns (Sequential-Path and/or further Branching Patterns and/or Loop Patterns). The Branching Pattern is further categorized according to the kinds of branchings that are used quite often in BP models. These are:

- Parallel Branches. The *Parallel Pattern* starts with a parallel split and ends with a parallel join. This pattern combines WF pattern 2 (Parallel Split) and 3 (Synchronization).
- Alternative Branches. The *Alternative Pattern* starts with a alternative split and ends with a alternative join. This pattern combines WF pattern 4 (Exclusive Choice) and 5 (Simple Merge).
- Multiple Alternative Branches. The *Multiple Alternative Pattern* starts with a multiple alternative split and ends with a multiple alternative join. This pattern combines WF pattern 6 (Multi-choice) and 7 (Synchronizing Merge).

A special kind of Branching Pattern is the Loop Pattern. It is a part of a BPM where a flow leads from a split element to a merge element which is a predecessor of the split element (see Fig. 6). This pattern includes all elements on the branch leading backwards. It is based on WF pattern 10 (Arbitrary Cycles).

Each block-structured process model consists of a combination of different patterns, and the above defined patterns are sufficient to describe every block-structured process model. The different patterns can be found in sequence or nested. While finding transformation patterns in block-structured models is straightforward, the next section deals with finding transformation patterns in graphstructured models.

3.2. Patterns in graph-structured models

Compared to block-structured models, patterns in graphstructured models can be overlapping (see Fig. 7) and branching patterns are not necessarily merged by a common merge element (see Fig. 8). The pattern definitions given in Section 3.1 need to be extended in adequate order to enable matching these patterns in graph-structured models.

The definitions for the *Start Pattern*, the *End Pattern* and the *Sequential-Path Pattern* remain unchanged.

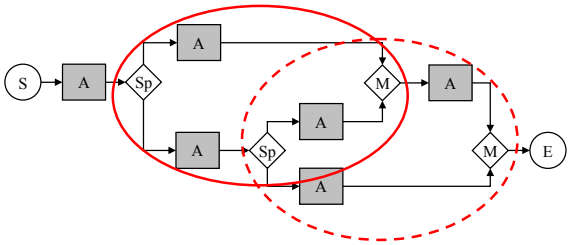

Figure 7. BP model containing overlapping patterns

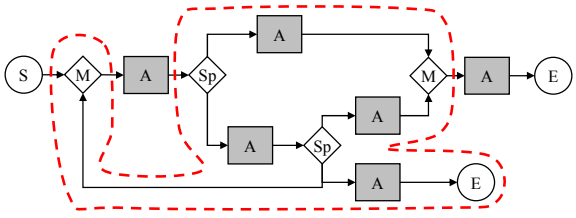

Figure 8. BP model containing patterns without common merge

For the Branching Pattern the end-condition needs to be refined: All of the outgoing branches (including the branches of subsequent split elements) of a split

element lead either (a) to a common merge element (see Fig. 9a) or (b) to an end element that can not be reached from the common merge element (see Fig. 10b) or (c) to an predecessor element of the split element (see Fig. 10c).

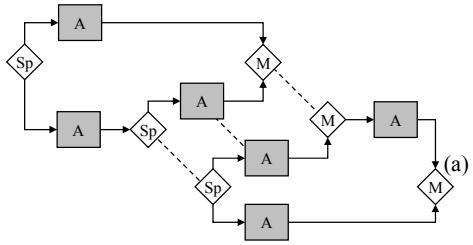

Figure 9. Nested Branching Patterns

The definition for the Loop Pattern itself remains unchanged but a Loop Pattern starting in a Branching Pattern must be considered in the definition of the Branching Pattern as follows: In addition to leading to an end element which is not the common merge element a branch could also lead (c) to a predecessor element of the common split element (see Fig. 10).

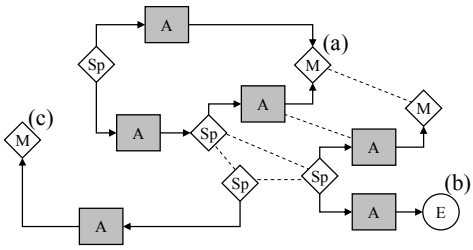

Figure 10. Nested Branching Patterns and Loop Pattern

By adding these extensions, it is possible to identify the transformation patterns in each model of a generic graphstructured language. In the next section, a case study demonstrates the use of transformation patterns.

4. Case Study - Using Structural Patterns for BP model transformation

In the following we shortly introduce the involved BPM languages and their main elements. Then the use of our structural patterns is demonstrated by means of a case study, which treats the translation from a ADONIS® BP model into the according EPC BP model. Finally we compare our approach with a supposed definition of the same scenario in QVT and summarize the advantages of our approach.

4.1. ADONIS® Standard Modeling Language

The ADONIS® Standard Modeling Language [4] provides different kinds of model types which cover different business aspects. The BP model is used to model the business processes itself, i.e., its behavior and control flow aspect. Furthermore it is used to integrate the organizational and the information aspect. Since our work focuses on the control flow aspect, we concentrate on the BP model. ADONIS® is a graph-structured BPM language.

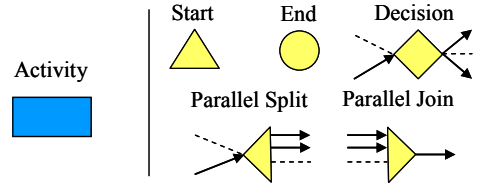

Figure 11. Main elements of ADONIS®

The integral model element is the activity (see Fig. 11a). A sequence of activities is modeled by means of the successor which represents the control flow in the BP model. The control flow elements (see Fig. 11b) are used to model the control flow.

The ADONIS® BP model provides no special element for modeling merges of alternative control flows. Furthermore, the decision element does not distinguish between alternative split and multiple alternative split.

4.2. Event Driven Process Chains

Event Driven Process Chains (EPCs) [5] have been introduced by Keller, Nuettgens and Scheer in 1992. EPCs are basically used to model processes.

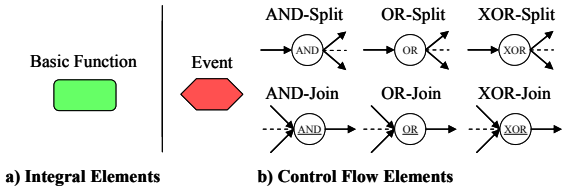

Figure 12. Main symbols of EPCs

We focus on the main elements which are used to model the integral and control flow aspect of a BPM, the elements Function, Event, AND, OR and XOR (see Fig. 12).

The Function describes an activity. It creates and changes Information Objects within a certain time. The event represents a BP state and is related to a point in time, it could be seen as passive element compared to the function as an active element (compare [15]). The remaining control flow elements (see Fig. 12b) are used to structure the proceed of the BP model. Different from ADONIS®, EPCs do not provide a specific element to indicate the begin and the end of a BP model. Event elements are used instead. Event elements are not allowed to be in front of an OR and XOR element. Function and event elements must alternate in the proceed of the BP model.

4.3. Translation of an ADONIS® model into an EPC model

For the case study we have chosen a model translation scenario in which a BP model defined in the ADONIS® Standard Modeling Language (see Fig. 13) is transformed to a BP model defined in EPC (see Fig. 17). The aim is to leave the given semantic of the BP model unchanged and adapt the syntax of the model according to the target BPM language (EPC).

The case study consists of four parts, definition of the patterns in each language, decomposition of the source BP model (see Fig. 13) according the defined patterns, transformation of each pattern, gluing of the target patterns to form the target BP model (see Fig. 17).

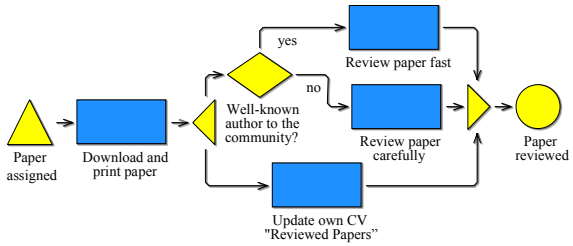

Figure 13. Source model in ADONIS® notation

The source BP model de scribes a simplified reviewing process (see Fig. 13). When a paper is assigned the first activity is to download and print it. After that, the flow is split into two flows indicating that the update of the own CV and the Reviewing can be executed in any order. The intensity of the reviewing is dependent on the celebrity of the author. In case of a well-known author the paper could be assumed as good and according to this be reviewed fast. Otherwise the paper must be reviewed carefully. After that the control flows merge in a parallel join element and the process ends. The BP model consists of one start, one end, four activity, one parallel split, one parallel join and one decision element. The merge of the decision ”Well-known author?” is modeled implicitly. For the transformation we suppose an algorithm that contains the general configuration which is equal for all inspected BPM languages, e.g. the start pattern has no predecessor and the end pattern no successor elements and that the loop pattern ends in a predecessor element etc. Based on this, specific configuration parameters have to be defined for each BPM language (cf. definitions in Table 1).

Pattern	Begin element	Explicit end/merge element	End/Merge element	Other end elements
Start	Start	n/a	n/a	n/a
End	End	n/a	n/a	n/a
Sequential-Path	Activity	n/a	Last activity in sequence	n/a
Parallel Branch	Parallel Split	yes	Parallel Join	End
Alternative Branch	Decision	no	Any element	End
Loop	Decision	no	Any element	n/a

Table 1. Pattern defined in ADONIS®

At the moment we assume four different parameters, the begin element, the explicit end/merge element (indicating whether the end/merge is explicitly modeled or not), the merge/end element and other end elements (if branching patterns are not merged - see graph-structured models).

The second step in our transformation scenario is to walk through the source BP model and decompose it into our defined patterns. Since the example includes nested patterns, multiple decomposition cycles are used:

- In the first decomposition cycle we found a start pattern (see Fig. 14a), a Sequential-Path pattern (see Fig. 14b), a Parallel Branch pattern (see Fig. 14c) and an end pattern (see Fig. 14d).
- The second decomposition cycle analyzes the Parallel Branch pattern. It is further decomposed in an Alternative Branch pattern (see Fig. 14e) and a sequential-path pattern (see Fig. 14f).
- In the last cycle the sequential-path pattern is decomposed into two Sequential-Path patterns as depicted in Fig. 14g).

Now we have decomposed our source BP model into adequate patterns which will in the following be transformed into patterns of the EPC language as defined in Table 2.

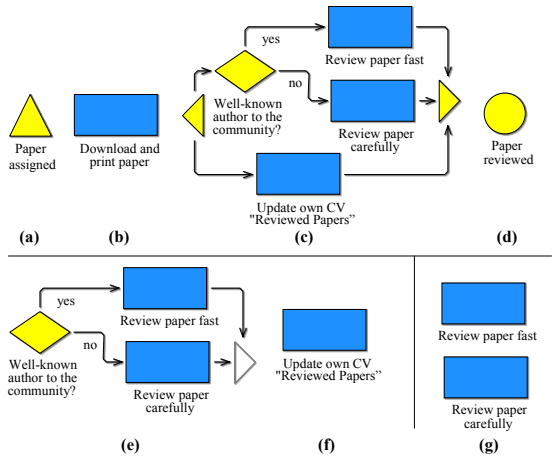

Figure 14. Decomposition of the model

The transformation of each pattern has to be done according to the information captured in the tables 1 and 2. The transformation of the start and end patterns is straight forward (see Fig. 15a and c).

Pattern	Begin element	Explicit end/merge element	End/Merge element	Other end elements
Start	Event	n/a	n/a	n/a
End	Event	n/a	n/a	n/a
Sequential-Path	Event or Function	n/a	Last event or funtion in sequence	n/a
Parallel Branch	AND	yes	AND	End Event
Alternative Branch	XOR	yes	XOR	End Event
Loop	XOR	yes	XOR	n/a

Table 2. Pattern defined in EPCs

In case of transforming the Sequential-Path patterns from ADONIS® to EPC we have to take care, that the EPC conditions, such as alternate Events and Functions and no Event followed by an OR or XOR element, are fulfilled. Therefore, we transform each ADONIS® Sequential-Path pattern into a EPC Sequential-Path pattern that starts with an Event. The information contained in the Events is derived from the preceding model element and the control flow edge that connects them. In case that the control flow edge contains no information, the postfix ”-done” is added (see Fig. 15b, d and e). The transformation of the Branching pattern consists of a translation and a composition.

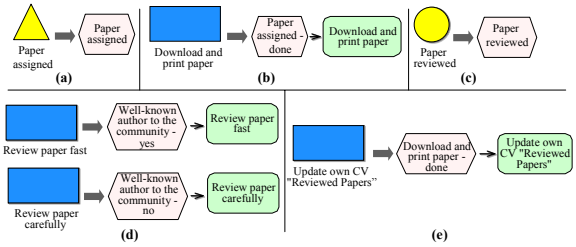

Figure 15. Transformation of Start, End and Sequential-Path pattern

Figure 16a illustrates how the Alternative Pattern is transformed into EPC. Here the knowledge of the explicit merge element in EPC is used to create this element at the end of the Alternative Pattern. After that, the Parallel Pattern is translated and composed as shown in Fig. 16b.

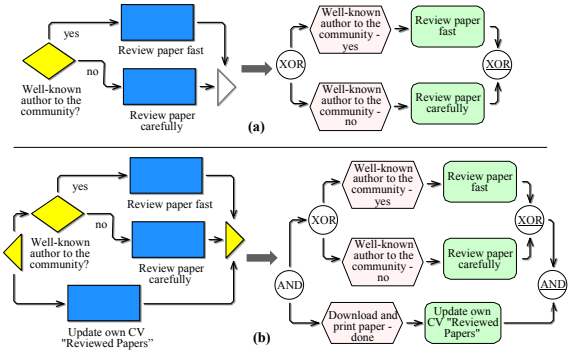

Figure 16. Transformation and Composition

Finally we have to glue all the parts together to form the target EPC model as depicted in Fig. 17.

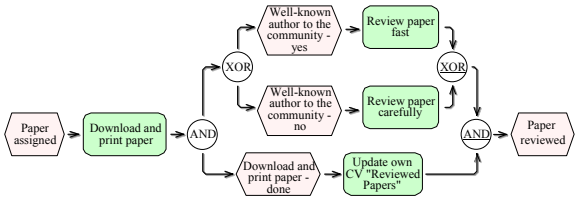

Figure 17. Translated target model

Attention must be given to the sequential-path pattern after the start pattern. In this case the event of the target pattern has to be deleted because of the preceding start event (see Fig. 15b) to avoid consecutive events. In case of defining the above BP model transformation scenario with a metamodel-based language such as ATL or QVT, there are similarities concerning the Start, End and Sequential-

Path pattern. These patterns could also be transformed by using simple metamodel correspondences. But in case of creating the explicit merge element in the target language we have to write a lot of imperative code. This code is in most cases not reusable for other model transformation scenarios, whereas the pattern-definition for a given BPM language is reusable when trans-forming its models into another BPM language. Additionally pattern-based definitions can be easily adopted for translating the models in the other direction. This is useful in case of the synchronization of BP models as depicted in Fig. 1c. In case of merging BP models (see Fig. 1a) the patterns fulfill two different functions. First the detection of the equal structures in the source models and second, their correspondence to the target model.

A general observation that we made is that in case of a transformation from a BPM language with few elements for expressing control flow structures, such as ADONIS®, to a BPM language which provides more elements to express control flow structures, such as EPC, it is preferable to take a model transformation approach that additionally supports pattern-based definitions.

5. Related work

There is a formal method for decomposing flowgraphs [16] which at first sight looks very similar to our approach of defining patterns for BP models. Basically this approach aims to improve the measurability of imperative source code. Based on the fact, that imperative programming languages could be illustrated as directed graphs the authors show that specific programming constructs could be modeled as sub graphs, so called primes. Decomposition of flowgraphs into primes assumes block-structured models. In case of graph-structured models which occur in BPM languages, occurring primes can become arbitrarily complex. Furthermore the number of different primes is not limited in case of graph-structured BP models. Thus it is not possible to define the number of primes in advance. So structural patterns for analyzing BP models have to be more abstract as the introduced primes. Defining the begin and the possible ends of a distinct pattern reduces the number of different patterns to be defined.

In defining model transformations, patterns have already been used for various purposes.

Design patterns in Model Driven Engineering (MDE), as initially discussed in [17], capture recurring problems in the design of metamodels and model transformations. In [17], examples of design patterns relating to transformation implementation are given. Different to our work, these patterns are domain-independent and thus the patterns address issues with MDE technologies rather than a particular modeling domain.

Pattern-based model refactoring [18] is a concept for controlled model evolution and development employing model transformations. Model refactoring focuses on ”forward transformation”, i.e., trans-formations that modify a model but do not change the modeling language, as opposed to horizontal transformations.

The MDA tool OptimalJ combines the MDA with pattern-based development [19]. It uses two kinds of patterns. First, so-called transformation patterns perform vertical transformations, i.e., PIM to PSM and PSM to code; these patterns embody design knowledge. Second, so called functional patterns reside on a particular layer, i.e., PIM, PSM or code, and represent recurring solutions to problems of application development. The well-known GoF design patterns are an example of functional patterns. While our patterns are also situated within a particular layer, they differ in that they do not address application development but rather recurring problems of horizontal transformations.

The final submissions for the QVT language [3] includes patterns as core concept of a model transformation language, to be used for queries on source models and as templates on target models. The QVT submission uses a declarative formalism for defining patterns. The patterns and the transformation rules using the patterns are designed to support aspect-driven transformations, as they allow construction of a target model element by application of several rules. While QVT defines a language to define patterns independent of particular domains, our work focuses on identifying particular patterns in the BPM domain. Another difference is that the QVT language requires distinct patterns for source and target, whereas our approach proposes generic patterns that are used as a bridge between source and target models, thus exploiting the specifics of horizontal transformations.

Using higher-level abstractions to design horizontal transformations is also proposed by [20]. In particular, [20] defines horizontal model transformations based on semantic concepts as defined in ontologies rather than on syntactic concepts as defined in metamodels. Our patterns are not focused on particular semantics but rather are motivated by their usefulness in transformation definition. Furthermore, the use of patterns, i.e., complex structural abstractions, is not elaborated in [20].

6. Conclusion and Outlook

In this paper a general approach to BP model transformation based on patterns has been introduced. We focus on the control flow aspect of BP models, which is considered as one of the catchiest of the aspects with respect to BPM transformations.

The objective was on the one hand to introduce a transformation approach which abstracts from the single metamodel elements usually used. On the other hand, we have provided a detailed description of the transformation patterns which helps to comprehend the structure of BP models. Furthermore, a case study about the translation of a model from ADONIS R to EPC has been discussed. An apparent advantage of covering the structure of BP models with transformation patterns is that it makes the differences between the various BPM languages visible. This knowledge helps to focus the transformation development effort especially on the differences between BPM languages.

This work represents the first step in realizing the proposed domain-specific transformation approach for BP models. We are currently working on an algorithm for detecting the defined patterns automatically. The next steps include the definition of the transformation of the individual patterns between different BPM languages, and it is planned to evaluate this approach in the scope of an industry project where BP models have to be merged.

References

[1] J. Miller and J. Mukerji, *MDA Guide*, version 1.0.1, Object Management Group, Inc., June 2003.

[2] J. Bézivin, F. Jouault, and D. Touzet, "An Introduction to the ATLAS Model Management Architecture," LINA, Tech. Rep. 05-01, 2005.

[3] *MOF QVT Final Adopted Specification*, Object Management Group, Inc., http://www.omg.org/docs/ptc/05-11-01.pdf, November 2005.

[4] BOC, "ADONIS 3.7 - User Manual III: ADONIS Standard Modeling Method." BOC Ltd.

[5] G. Keller, M. Nuettgens, and A.-W. Scheer, "Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK)", Universitaet Saarbruecken, Tech. Rep., 1992.

[6] M. Murzek, "ASPAT - An Aspect- and Pattern-oriented Approach for Business Process Model Transformation," WIT, Tech. Rep., May 2006.

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

[8] B. Axenath, E. Kindler, and V. Rubin, "An Open and Formalism Independent Meta-Model for Business Processes," in *Proceedings of the Workshop BPRM 2005*, September 2005.

[9] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14(1):5-51, 2003.

[10] S. A. White, "Process Modeling Notations and Workflow Patterns," *BPTrends*, March 2004.

[11] P. Wohed, W. M. van der Aalst, M. Dumas, A. H. ter Hofstede, and N. Russell, "Pattern-based Analysis of UML Activity Diagrams," Eindhoven University of Technology, Eindhoven, Tech. Rep., 2004.

[12] J. Mendling, G. Neumann, and M. Nuettgens, "Towards Workflow Pattern Support of Event-Driven Process Chains (EPC)," in *Proceedings of the 2nd GI Workshop XML4BPM at 11th GI Conference BTW 2005*, 2005.

[13] *Business Process Modeling Notation Specification*, Object Management Group, http://www.bpmn.org/, 2006.

[14] *UML 2.0 Superstructure Specification*, Object Management Group, Inc., http://www.omg.org/docs/formal/05-07-04.pdf, July 2005.

[15] J. Mendling and M. Nuettgens, "EPC Modelling based on Implicit Arc Types," in *Proceedings of the 2nd International Conference on Information Systems Technology and its Applications (ISTA)*, 2003.

[16] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. Boston, MA, USA: PWS Publishing Co., 1998.

[17] J. Bézivin, F. Jouault, and J. Palies, "Towards model transformation design patterns," in *Proceedings of the First European Workshop on Model Transformations (EWMT 2005)*, 2005.

[18] S. R. Judson, "Pattern-based model transformation," in *OOPSLA Companion*, 2003, pp. 124–125. [19] A. Herzog, "Ueber Transformationen und Patterns: Wie Compuware OptimalJ die MDA implementiert, " *Objektspektrum*, January 2004.

[20] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer, "On Models and Ontologies - A Layered Approach for Model-based Tool Integration," in *Proceedings of the Modellierung 2006 (MOD2006)*, March 2006, pp. 11–27.