The Model Morphing Approach - Horizontal Transformations between Business Process Models *

Marion Murzek[1] and Gerhard Kramler[2]

[1] 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
[2] Business Informatics Group (BIG),
Institute for Software Technology and Interactive Systems
Vienna University of Technology, Austria
kramler@big.tuwien.ac.at

Abstract. Due to company mergers, acquisition and business to business interoperability, there is a need for model transformations in the area of business process modeling to facilitate scenarios like model translation, integration and synchronization. Thus this paper concentrates on transformations of models between different business process modeling languages. As current transformation languages provide general solutions and do not support the special properties of business process models, it is still a challenge defining such transformations. To tackle this problem we introduce the model morphing approach. Our main idea is to create an integrated metamodel containing all concepts of the languages of a given domain. Based on this integration the model transformation can be defined in terms of morphing steps. Our approach is demonstrated by model transformation in the area of business process modeling but is generally suitable for domains in which a variety of languages is used that express similar concepts.

1 Introduction

Since companies discovered the value of modeling their business processes, many languages for this purpose have been developed. Some are theoretically funded specifications, for example UML 2.1 Activity Diagrams [20], Event-driven Process Chains [13] or the Business Process Modeling Notation [19], but the bulk of the existing languages evolved from consulting projects in industry, for example ADONIS® Standard Modeling Language [5]. Consequentially nowadays we are faced with a proliferation of business process modeling languages that are used to model real world processes.

* 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.

Changes concerning business process modeling languages due to company mergers, acquisitions or software changes often require transforming models complying to one modeling language into models complying to another modeling language. Thus this work concentrates on providing and supporting solutions for the problems occurring during the definition of business process model transformation. After analyzing the issues arising at transforming process models [17], the first step was to inspect state-of-the-art transformation languages for solving these problems.

Inspired by the vision of MDA [16] current techniques or specifications used for defining model transformations, such as ATL [4] or QVT [18], operate nearly exclusively at the level of metamodel elements. From this it follows that direct 1:1 mappings between metamodels are supported very well. But for the transformation requirements in a specific domain, in our case BPM, these techniques are not sufficient [17]. On the other hand, using pure Java for defining model transformation makes it possible to define everything needed, but it does not support the infrastructure that model transformation languages provide.

This motivated us to provide the Generic Model Morphing Framework that supports a software architecture which combines the use of an existing model transformation language, and a programming language. ATL is used for the definition of "simple" 1:1 correspondences between the different metamodels. Java is used to implement the special transformation issues of a specific domain, in our case the the BPM domain.

Accordingly, the contribution of this work is threefold:

1. The transformation mappings between four selected business process languages.
2. The model morphing approach and also the architecture is defined domain-independently and thus can be specialized for any distinct domain.
3. The architecture of the model morphing approach supports reuse. On the one hand concerning existing technologies used (in our case ATL and Java) and on the other hand concerning the morphing methods which are encapsulated solutions for non-trivial transformation requirements that can be applied to different transformation scenarios within a given area.

The reminder of this work is structured as follows. The next section introduces the model morphing approach. In Section 3 the framework for business process model transformations is presented. Section 4 demonstrates the approach in the business process modeling domain. Furthermore it outlines the reuse potential of the morphing methods in in this domain. Section 5 puts our work into context of related work. The conclusion in section 6 summarizes this work and gives an insight into ongoing work.

2 Model Morphing

Model morphing is a new approach to tackle model transformations in a specific domain. This approach represents a software architecture for model transfor-

mation, that enables the reuse of model transformation definitions and allows the embedding of different technologies. Model morphing operates mainly on the differences between the languages dedicated to a specific application domain, whereas the similarities of the languages can be treated by using simple declarative 1:1 transformations.

The main assumption is, that the languages of a specific domain are all used to express similar concepts characterizing this domain. This makes it possible to create a common *integrated metamodel* which covers all concepts found in the participating languages. Conceptually this integrated metamodel represents a union of all concepts found in the languages belonging to a given area.

Through this integration it is possible to map models complying to specific metamodels directly to the integrated metamodel. The corresponding transformations can be easily developed using an existing model transformation approach, for example ATL.

The core transformation is performed by using so-called *model morphing methods*. These methods are derived from the differences between the specific languages and are defined based on the integrated language (IntL). They are applied step by step on the source model which has already been translated to the IntL until it fits to the structure of the target language. In the last step the model is translated from the IntL into the target language.

This approach responds to the non-trivial transformation issues occurring when defining transformations between models in a distinct area. The incremental nature of the model morphing approach makes it possible developing the transformation step-by-step. This supports a transformation definition where the result could be inspected after every step.

As this approach strongly singles out the particularities of a specific area in form of the morphing methods it makes it possible for the domain specialists to define model transformations by only the use of their expert knowledge and also without programming skills.

2.1 The Generic Model Morphing Framework

To realize the model morphing approach a so called Generic Model Morphing (GeMM) framework has been developed. This framework consists of four main components, the *Integrated Metamodel* connected by the *Adapter Layer* with the *Specific Metamodel Layer* and the *Method Repository* which makes use of the graph and the patterns operating on the graph (see Fig. 1).

To develop the GeMM framework we have used the Eclipse Modeling Framework (EMF) [8], [6] because it provides an environment to develop metamodels very fast and easily. Furthermore EMF has implemented an automatic (java) code generation mechanism for such metamodels. Moreover this java code provides the basis for the morphing methods. As we wanted to use a metamodel-oriented transformation language for the adapter layer we found that ATL is a good solution because it offers a declarative way to define metamodel correspondences and moreover it operates on EMF metamodels.

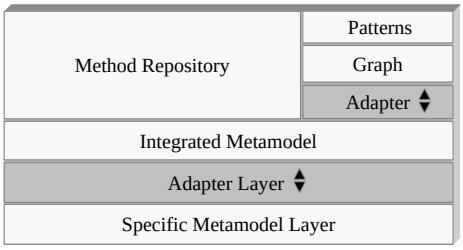

Fig. 1. The Generic Model Morphing Framework

The *integrated metamodel* has been defined under the consideration that every graphical model when reduced to the minimum is a graph consisting of nodes and edges. Consequentially the core of the integrated metamodel could be defined as a model having nodes and edges. Therefore the fix parts of the integrated metamodel are the class `Integrated Model` and the abstract classes `Node` and `Edge` each containing two attributes, `name` and `description` (see Fig. 2, the three classes in the dashed rectangle). The core of the integrated metamodel is the starting point for connecting further classes needed for the integrated metamodel of a distinct domain. Note that during the use of a specialization of this framework, for example when attaching another language to the framework, further classes or attributes can be added easily.

The *method repository* consists of templates for methods and mechanisms which are needed to construct the concrete methods for a distinct area. These methods make use of the generated code from the integrated metamodel implemented in EMF. The graph component is realized according to the theory of well-known graph algorithms as for example characterized in [21], [14]. At the moment the basic patterns which are defined in this graph are structural patterns on directed graphs. As the core of the integrated metamodel always stays the same the graph and the pattern implementation can be reused in different domains without changes.

The *specific language layer* has to be defined newly for every domain. For the metamodels of the specific languages, we also used EMF to be able to use ATL for the adapter layer.

The *adapter layer* between the specific metamodel layer and the integrated metamodel is designed to achieve a 1-to-1 mapping from the metamodel of a specific language to the integrated metamodel. In our case this means that each element of a languages metamodel is mapped to one element of the integrated metamodel. In case this is not possible, the integrated metamodel has to be extended with adequate concepts (classes or attributes) until the adapters can be written by using only 1-to-1 mappings.

The next section demonstrates for the business process modeling domain, how the GeMM framework can be specialized for a distinct area.

3 The Model Morphing Framework for Business Process Model Transformation

Having the ability to design an integrated metamodel (IMM) for the domain of business process modeling it is necessary to understand the concepts [23] and aspects [24], [11], [3] used in this domain. Furthermore it is required to get familiar with the syntactical and semantical expressions of the concepts and aspects in each of the participating business process modeling languages (BPMLs).

Due to the lack of space we have decided to present the GeMM framework by means of the control flow aspect [3] which is the main aspect of business process modeling. This should be sufficient to understand how the construction of such a framework has to be done.

Four BPMLs, ADONIS® Standard Modeling Language (ADONIS®) [5], Business Process Modeling Notation (BPMN) [19], Event-driven Process Chains (EPC) [13] and UML 2.0 Activity Diagrams (AD) [20] have been selected to participate in our example.

In the following the concepts which characterize the control flow aspect of the business process modeling domain are described.

3.1 The Business Process Modeling Domain

Process modeling languages for describing the behavioral aspect in the domain of Business Processes are used to model the flow of work which has to be done to reach a certain aim. Illustrating such processes in form of a directed graph on a two-dimensional drawing area it is necessary to create syntactic elements which are representatives of the concepts used in reality. The connection between an element and its corresponding concept clarify the semantics of the element. In the following the concepts and the modeling elements are described.

The main concept of a business process is a working step, represented by the task element. Such a working step lasts a distinct period of time, is done by somebody using some resources and converts some input into some output. This integral concept could be found in each of the participating BPMLs, for example a task in BPMN or an activity in ADONIS® . To support the clear arrangement of a process model a concept for structuring processes into main and sub processes is provided, represented as Sub-Process element. Every participating BPML provides an element for this concept. Some languages provide an explicit element - event - to model the actual state of a process whereas others do not model states explicitly.

Beside these integral concepts like working step, sub process and event some concepts in form of logical operators to model the behavioral aspect of a process are needed. They are used to depict the begin and the end(s) and different kinds of pathes a process could run. Some kind of start element is used to depict the begin and some kind of end element is used to illustrate the end of a process. In the course of a business process decisions lead to different continuations of a

process. This is realized in the form of elements expressing alternative paths. In most cases a finer differentiation is given by exclusive and inclusive alternatives. As some working steps could be done concurrently there are elements like parallel forks for modeling such circumstances.

As glue between and within the integral and the behavioral elements the flow of control itself is depicted by using arrows with some kind of arrowhead.

The concepts which are used to describe the integral and behavioral aspect of a business process model could be found more or less in every participating language (see Table 1).

3.2 Analyzing the Languages

Table 1 provides the result of an analysis regarding the similarities and differences between the BPMLs we use. Note that the terminology of the integrated language (IntL) has been chosen freely.

Table 1. Comparison of the elements in the participating BPMLs

IntL	ADONIS®	BPMN	EPC	AD
Task	Activity	Task	Basic Function	Opaque Action
Sub-Process	Subprocess	Sub-Process	Complex Function	Call Behavior Action
Event	n/a	n/a	Event	n/a
Start	Start	Start Event	Event	Initial Node
Multiple Starts	n/a	yes XOR	yes XOR	yes AND
End (local)	End (local)	End Event	Event	Flow Final Node
End (global)	End (global)	n/a	n/a	Activity Final Node
AND (split)	Parallel Split	Parallel Fork	AND	Fork Node
AND (join)	Parallel Join	Parallel Join	AND	Join Node
OR (split)	Decision	Inclusive (OR)	OR	Fork Node + Conditions
OR (join)	implicit	Inclusive (OR)	OR	n/a
XOR (split)	Decision + Conditions	Exclusive (XOR)	XOR	Decision Node
XOR (join)	implicit	Exclusive (XOR)	XOR	Merge Node
Control Flow	Successor	Sequence Flow	Control Flow	Control Flow

After analyzing the participating BPMLs four types of differences can be observed:

(1) Context-dependent semantics of elements.
   (a) Local context is needed - adjacent elements. This type of difference can be observed in the case of the concept XOR-Split in ADONIS® where the same element Decision is used for the OR and the XOR-Split. The distinction can only be made by inspecting the conditions of the outgoing Successors. Similarly the OR-Split in AD is realized by using a Fork Node and entering or-conditions into the attribute condition on the outgoing Control Flows. Furthermore AD provides the possibility to model the AND-split and the AND-join implicitly instead of using the fork and join node.
   (b) Global context is needed - structure of a part of the process. The *OR-join* and the *XOR-join* can only be modeled explicitly in BPMN and EPC. In ADONIS® this element can only be depicted implicitly by two or more Successors leading into the following element (this is also the case for the *XOR-join* in ADONIS®).

(2) Missing possibility to represent a concept - *no element provided*.
   In row 3 in table 1 it can be seen, that the concept *Event*, which is a process state in a distinct point of time, is only provided in the EPC language. In BPMN and EPC there is no possibility to depict a *global end*. And in case of AD there has not been found any valid solution for the OR-join in the UML 2.1 language specification [20].

(3) Elements are provided for a distinct concept (same syntax), but they have *different semantics*.
   There is a concept for modeling the *start* of a process in every language. But one language (ADONIS®) only allows one start element for each process model, whereas the other three allow more than one. Furthermore there is a difference between the semantics of the BPMLs which allow more than one start elements. In BPMN and EPC it is enough if one of multiple start elements is activated to trigger the process, whereas in AD it is required that all of the used start elements are activated to run the process.

3.3 An Integrated Metamodel

As a result of this analysis we have built the integrated business process metamodel (IBPMM) depicted in Figure 2. The IBPMM is built upon the core of the integrated metamodel as described in Section 2.1.

The `Task`, the `Subprocess` and the `Event` are implemented as simple classes. As a subprocess has to be linked to a another process the association `refProcess` has been established between the class Subprocess and the class `Integrated Model`.

The grouping of the Logical Operators has been made to provide a general access to these structuring elements of a model. To keep the number of classes small there is only one class for each of the logical operators `AND`, `XOR` and `OR`. The differentiation between a split and a join/merge operator is indicated by the attribute `kind` which is of type `LogicalOperationKind`. This design decision was made to make changes to a model during incremental transformation easy, only attributes must be changed instead of replacing whole elements.

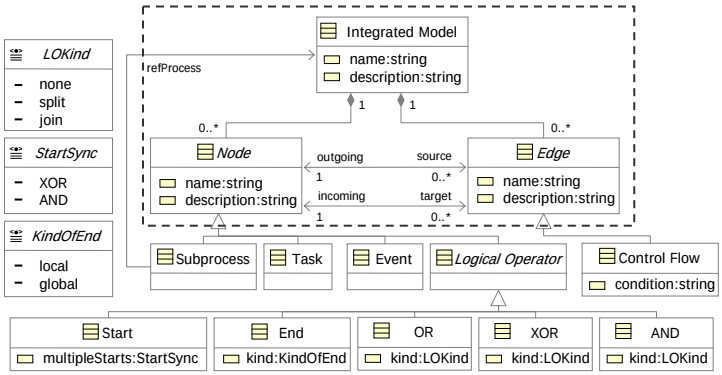

Fig. 2. The integrated business process metamodel

As most of the BPMLs provide an element for explicitly marking the start and the end of a process the class `Start` and `End` has been implemented.

In case of the class `Start` the attribute `multipleStarts` has been introduced to differ between the different semantics of multiple start elements. The type of this attribute is the enumeration `StartSynchronisation` which could have the value XOR for exclusive alternative or AND for synchronisation semantics.

In case of the class `End` a distinction has to be made between a global or a local kind of end. This is also realized by an attribute namely `kind` of the enumeration type `KindOfEnd`.

The `ControlFlow` summarizes the concept of a flow of control which connects all other elements and thus is the linking element within a model. The attribute `condition` contained by the class `ControlFlow` is needed to indicate conditions after decisions. The generalization of the Control Flow to Edge seems a bit unnecessary at the moment but as the IBPMM is dedicated to grow this was a preparation for further extensions of edges.

3.4 The Adapters

To be able to specify the necessary adapters all metamodels of the participating BPMLs have to be realized as EMF metamodels. Based on these metamodels the adapters have to be implemented between all of the participating BPMLs

and the IntL in both directions. As the IntL contains all concepts found in each of the participating BPML the relations between each element of each BPML to the elements of the IntL are 1:1. For implementing these correspondences we used ATL.

3.5 Model Morphing by Example

The morphing methods implemented in the Method Repository (MR) reflect the differences between the participating BPMLs (see Table 1). That means that they are derived from these differences. Furthermore the methods provide the possibility to overcome these differences at transformation time and thus make the main contribution to the transformation. In the following the methods and their mode of operation within the GeMM framework are described by means of an example business process model transformation.

Problem Description

The ADONIS® source model (see Fig. 3) should be transformed into the semantically equivalent target model (see Fig. 4) conforming to the EPC language.

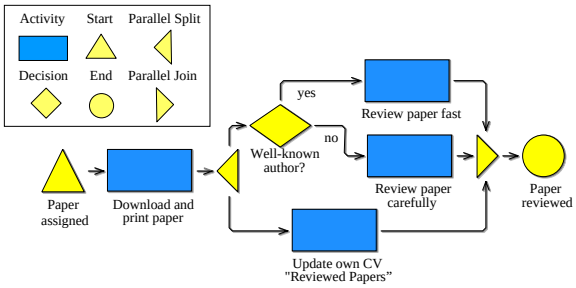

Fig. 3. "Review Paper" Process Model in ADONIS®

According to the model transformation issues described in [17] this model transformation poses the following difficulties:

- *Decision (Un)Ambiguity.* In ADONIS® the decision element is used to express inclusive and exclusive alternatives. Whereas EPCs provide one distinct element for each of the two concepts, namely OR and XOR.
- *Invisible Merger.* In ADONIS® the merge of a decision is implicitly modeled by two or more successors leading into an object. In EPCs there are explicit elements for illustrating such mergers, namely XOR and OR.
- *Mandatory Events.* In EPCs events are mandatory. Furthermore it is mandatory that event and function elements are alternating during the flow of the process model. There is no corresponding element in ADONIS® .

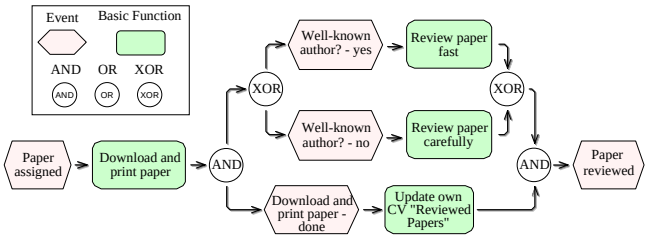

Fig. 4. "Review Paper" Process Model in EPC

Furthermore the Start and End elements of the ADONIS® model have to be transformed into Event elements in EPC.

Solution Description

The first step is to transform the ADONIS® model via the *ADONIS® to IntL* adapter to a model conforming the integrated metamodel, the resulting model is illustrated in Fig. 5. Note that this is an 1:1 translation so only the abstract and concrete syntax has changed. The further transformation definition is realized

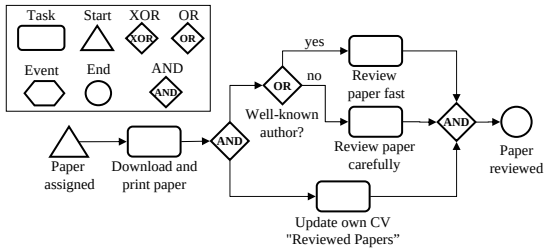

Fig. 5. "Review Paper" Process Model in the IntL

by using the following methods:

1. The `convertStartToEvent()` method substitutes the start with an event element.
2. The `convertEndToEvent()` method changes the end to an event element.
3. The `makeLOsUnambig()` method operates on a heuristic making use of antonyms. It checks if the values of condition attributes of the outgoing control flows of the OR element are antonyms and converts them to XOR elements if necessary.
4. The `makeJoinsExplict()` method uses the graph and the pattern component to find out, if there is an implicit merge. If so an explicit merge element

(OR or XOR depending on the branch element) and necessary control flows are created and integrated into the model.

After applying the methods 1-4 the model looks as shown in Fig. 6.

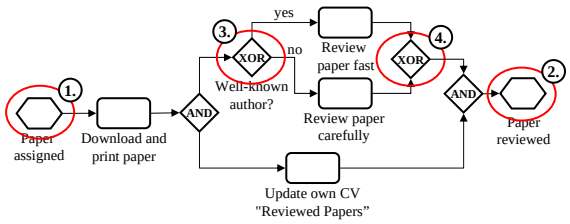

Fig. 6. "Review Paper" Model after four morphing steps

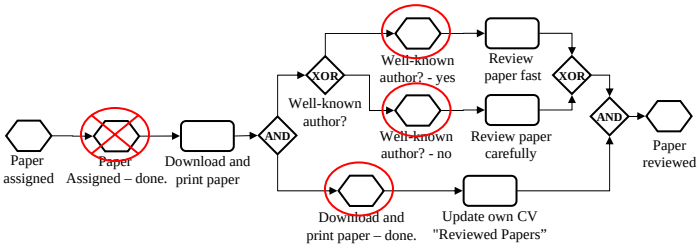

Fig. 7. "Review Paper" Model after applying the insertEvents() method

5. Finally the events which are mandatory in EPCs have to be inserted. The method `insertEvents()` inserts an event in front of each function. This could lead (as in this case) to an unnecessary event if there is no preceding event (see Fig. 7 Event "Paper Assigned - done"). This event is redundant and makes the model invalid concerning the syntax of EPCs where events and functions must alter during the process model. Therefore the method `insertEvents()` calls a helper method which searches for such interfering events and deletes them. The name of a newly inserted event is generated depending on the preceding node and edge. In case the preceding edge has a name value it will be added to the name of the preceding node, like "Well-known author - yes", if not, then the fix term "- done" is added to the preceding node name.

After applying these methods the "Review Paper" process model is ready to be transformed into the target BPML, in this case EPCs. This is done by using the *IntL to EPC* adapter.

3.6 Morphing Methods Overview

The table 2 shows the methods needed in the possible transformation scenarios using the four BPMLs, ADONIS®, EPC, AD, BPMN. It provides a good overview and makes the reuse of each method visible.

Table 2. The use of methods in the different model transformation scenarios. In this table the abbreviations are as follows: EP = EPC, AD = AD, BP = BPMN, AO = ADONIS®.

Method / Source Language	ADONIS®			EPC			AD			BPMN		
Target Language	EP	AD	BP	AO	AD	BP	AO	EP	BP	AO	EP	AD
convertStartToEvent()	x							x			x	
convertEndToEvent()	x							x			x	
makeLOsUnambig()	x	x	x				x	x	x			
insertEvents()	x							x			x	
makeSplitsExplict()							x	x	x			
makeJoinsExplict()	x	x	x				x	x	x			
removeEvents()				x	x	x						
makeLOKindExplicit()				x	x	x						
makeLOsImplicit()		x		x	x		x			x		x
annotateGlobalEnds()	x		x					x	x			
convertEventToStart()				x	x	x						
convertEventToEnd()				x	x	x						
mergeMultipleStarts()				x	x		x	x	x	x		x

The first four of the eleven methods implemented in the GeMM framework for the participating BPMLs have directly been introduced during the example. So let us have a look at the remaining seven.

The method `makeSplitsExplicit()` works similar to `makeJoinsExplicit()`. Also the graph and pattern component is used to detect the implicit split element and then a new explicit split element is created and integrated. To remove events from a model the method `removeEvents()` can be used. The method `makeLOKindExplicit()` is used to be able to differ between an splitting and a merging logical operator. To make logical operators

implicit, in case of transforming in the direction of ADONIS® the method `makeLOsImplicit()` has to be used. This method can be seen as reverse method to `makeJoinsExplict()`. The method `annotateGlobalEnds()` is used to avoid loss of information regarding the different kinds of ends. As some BPMLs do not support the concept of global end, this method annotates the string "+ global" to the description attribute of an end. In case of transforming from EPCs it is necessary to convert the start and end events to start and end elements, this is reflected in the methods `convertEventToStart()` and `convertEventToEnd()`. The method `mergeMultipleStarts()` is used for different purposes. In the case of transforming models of BPMLs which have an AND semantics for multiple starts to a BPML which allows only one start element or has an XOR semantics for multiple start elements, the multiple start elements must be reduced to one start element and an AND-Split must be added after this element to connect it with the remaining process paths. A similar situation where the start elements in the source models have an XOR semantics can also be handled with this method. In this case the multiple start elements must be reduced to one start element and an XOR-Split must be added after this element to connect it with the remaining process paths. Note that nothing has to be done in case of transforming models from a BPML which allows only one start element.

The overview in Table 2 makes obvious, that the reuse of the most morphing methods depends on the BPML used as source or target language. The methods `removeEvents()`, `makeLOKindExplicit()`, `convertEventToStart()` and `convertEventToEnd()` for example are only used if EPCs is the source language. Whereas the methods `convertStartToEvent()`, `convertEndToEvent()` and `insertEvents()` are used when transforming to EPCs.

The degree of dissimilarity of the participating BPMLs regarding their support of the concepts of the domain is another observation that can be derived when looking at the numbers of methods needed for a transformation definition between two languages. The more different methods have to be applied, the higher are the differences.

4 Related Work

To the best of our knowledge there is no directly comparable approach to model morphing, which specifically supports horizontal model transformations in a distinct domain. Therefore we have decided to relate model morphing to existing general purpose model transformation approaches based on the work of Czarnecki and Helsen [7]. According to the feature model introduced in [7] the main characteristics of the model morphing approach are as follows.

- The integrated metamodel, the morphing methods and the adapters form the *transformation definition* part.
- Two kinds of *transformation rules* are used. Relational rules within the adapters implement exogenous transformation and the morphing methods implement endogenous transformations based on the integrated metamodel.

- In the transformation approaches considered in [7] intermediate *data structures* are playing a secondary role compared to the transformation rules. In contrast, the integrated metamodel plays the central role in the model morphing approach, whereas the morphing methods are defined based on the integrated metamodel.
- *Traceability* is not required in case of the model morphing approach. But it is implicitly covered in the morphing methods and could be extracted if necessary for a distinct application scenario.
- *Hybrid approaches* as characterized in [7] combine different transformation techniques within one transformation language. Whereas model morphing is a technology spanning and at the same time technology independent approach. That means the advantages of different transformation techniques can be combined. In our realization we used the relational approach and direct manipulation.

In the following we discuss related work regarding the integrated metamodel and the reuse of the methods.

The definition of an integrated metamodel is not a particular new approach, much work has been done in the 90s in the area of integrated metamodels. However the focus of this work is different. COMan [12] for example integrates an object-oriented application and a relational database, providing persistence for complex objects and object-oriented manipulation of relational data. This approach is dedicated to support business reengineering. In [9] an integrated environment for method engineering has been introduced. The aim of this work was to integrate different design notations to support metamodeling. Another relation can be observed to the area of schema integration. In case of schema integration, local data source are integrated to one global view as for example in [10] or [22]. This global view can then be used as a unified representation of the local data sources. This makes it possible sending one query to the global view instead of sending many queries to each local schema. In contrary to the different works stated above, our integrated metamodel specially focuses on the support of the transformation process, thus is an important component of our framework rather than a stand-alone product.

In the INTEROP [1] project the Unified Enterprise Modelling Language (UEML) has been developed [2]. The UEML provides core modeling methodology elements and is intended for exchanging information between enterprise modeling tools. UEML provides common semantic definitions - an intersection - of modeling constructs defined in an abstract UEML meta-metamodel. This is in contrast to our integrated metamodel that forms an aggregation of all concrete metamodel elements of all participating BPMLs.

Regarding modularization and reuse, the work in [15] analyzes rule based transformation languages for the support of creating modular transformation definitions. Assuming that transformations are built on the base of source and target metamodels, modularization of the transformation is derived from the modularization of these metamodels. The authors conclude that existing rule

based languages provide support for modularization and integration mechanisms, i.e., rule inheritance and rule calls. However, this does not meet the requirements for reusability and adaptability of transformation definitions. Modularization in the our approach is achieved by decomposing the transformation into adapters and morphing methods. The modularization of the morphing methods does not depend on any modularization of the metamodels, rather morphing methods encapsulate the solution algorithms for non-trivial model transformation problems. Due to fact that there are many similar transformation scenarios between the languages belonging to a specific domain, this kind of modularization facilitates exploiting the reuse potential.

5 Conclusion and Outlook

In this paper we presented model morphing - a new approach to tackle model transformation issues which we found to be difficult using existing model transformation techniques. We used this approach for defining transformations between different BPM languages. It should be noted, that model morphing is generally intended to define horizontal model transformations in a given domain where many different modeling languages are used to capture the same or similar concepts.

Inverse to the presentation in this paper the development of the approach started with the implementation of the transformation in the area of business process modeling domain. From that it has been generalized to the generic model morphing framework and further abstracted to the model morphing approach.

Thus, future work includes the application of the approach and the framework to a different domain to validate our hypothesis regarding the applicability of the model morphing approach to different domains. The framework will be further elaborated regarding the treatment of attributes, bi-directionality in form of inverse morphing methods and test mechanisms for validation. Furthermore an evaluation regarding the use of other transformation languages than ATL for the adapter layer has to be done.

As the construction of the framework requires a lot of expert knowledge and implementation skills it is not efficient to use the framework with only a small number of languages (2-3 languages). Therefore we have to evaluate the minimum number of participating languages where the construction of such a framework pays off.

References

1. *INTEROP Project.* http://interop-noe.org, (Last accessed: 2007-08-28), 2007.
2. *UEML 2.1 - Unified Enterprise Modelling Language.* http://www.ueml.org/, (Last accessed: 2007-08-28), 2007.
3. B. Axenath, E. Kindler, and V. Rubin. An Open and Formalism Independent Meta-Model for Business Processes. In *Proceedings of the Workshop on Business Process Reference Models 2005 (BPRM 2005), Nancy, France*, pages 45–59, 2005.

4. J. Bézivin, F. Jouault, and D. Touzet. An Introduction to the ATLAS Model Management Architecture. Technical report, LINA, 2005.
5. BOC. ADONIS 3.7 - User Manual III: ADONIS Standard Modeling Method. BOC Ltd., 2005.
6. F. Budinsky, S. A. Brodsky, and E. Merks. *Eclipse Modeling Framework*. Pearson Education, 2003.
7. K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, pages 621–645, 2006.
8. I. Eclipse Foundation. *Eclipse Modeling Framework (EMF)*. http://www.eclipse.org/modeling/emf/?project=emf.
9. J. C. Grundy and J. R. Venable. Towards an integrated environment for method engineering. In *Proceedings of the IFIP TC8, WG8.1/8.2 working conference on method engineering on Method engineering : principles of method construction and tool support*, pages 45–62, London, UK, UK, 1996. Chapman & Hall, Ltd.
10. G. Hu. Global schema as an inversed view of local schemas for integration. *sera*, 0:206–212, 2006.
11. S. Jablonski, M. Boehm, and W. Schulze. *Workflow-Management. Entwicklung von Anwendungen und Systemen*. dpunkt.verlag, 1999.
12. G. Kappel, S. Preishuber, E. Proell, S. Rausch-Schott, W. Retschitzegger, R. Wagner, and C. Gierlinger". COMan - coexistence of object-oriented and relational technology. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 259–277, 1994.
13. G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". Technical report, Institut für Wirtschaftsinformatik Universität Saarbrücken.
14. W. Kocay and D. L. Kreher. *Graphs, Algorithms and Optimization*. Chapman & Hall/CRC, 2004.
15. I. Kurtev, K. van den Berg, and F. Jouault. Evaluation of rule-based modularization in model transformation languages illustrated with atl. In *21st Annual ACM Symposium on Applied Computing (SAC2006), Bourgogne University, Dijon, France*, pages 1202–1209. ACM, April 2006.
16. J. Miller and J. Mukerji. *MDA Guide*. Object Management Group, version 1.0.1 (omg/03-06-01) edition.
17. M. Murzek and G. Kramler. Business process model transformation issues. In *Proceedings of the 9th International Conference on Enterprise Information Systems ICEIS 2007*, 2007.
18. OMG. *MOF QVT Final Adopted Specification*. Object Management Group.
19. OMG. *Business Process Modeling Notation Specification*. Object Management Group, http://www.bpmn.org/, February 2006.
20. OMG. *UML 2.1 Superstructure Specification*. Object Management Group, http://www.omg.org/docs/ptc/06-04-02.pdf, April 2006.
21. R. Sedgewick and M. Schidlowsky. *Algorithms in Java, Part 5: Graph Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
22. S. Sundaresan and G. Hu. Schema integration of distributed databases using hypergraph data model. In *IRI*, pages 548–553, 2005.
23. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*.
24. W. M. P. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. The MIT Press, January 2002.