2007

Facilitate Modelling Using Method Integration: An Approach Using Mappings and Integration Rules

S. Zivkovic  
BOC Information Systems GmbH, srdjan.zivkovic@boc-eu.com

H. Kuhn  
BOC Information Systems GmbH, harald.kuehn@boc-eu.com

Dimitris Karagiannis  
University of Vienna, dke@dke.univie.ac.at

Follow this and additional works at: http://aisel.aisnet.org/ecis2007

Recommended Citation
http://aisel.aisnet.org/ecis2007/122

This material is brought to you by the European Conference on Information Systems (ECIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in ECIS 2007 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
FACILITATE MODELLING USING METHOD INTEGRATION:

AN APPROACH USING MAPPINGS AND INTEGRATION RULES

Srdjan Zivkovic, BOC Information Systems GmbH, Wipplingerstr. 1, 1010 Vienna, Austria, srdjan.zivkovic@boc-eu.com
Harald Kühn, BOC Information Systems GmbH, Wipplingerstr. 1, 1010 Vienna, Austria, harald.kuehn@boc-eu.com
Dimitris Karagiannis, University of Vienna, Department of Knowledge Engineering, Brünnerstr. 72, 1210 Vienna, Austria, dk@dke.univie.ac.at

Abstract

Conceptual modelling is one of the core disciplines within business informatics. Often, different metamodels have to be integrated to provide project specific or domain specific modelling solutions. This is a challenging task due to the heterogeneity of the modelling languages being assembled. This paper presents an approach for integrating heterogeneous modelling languages in the domain of enterprise modelling. It extends the Enterprise Model Integration approach (EMI) by introducing two key concepts: mappings and rules for metamodel integration. For both mappings and integration rules language definitions are provided. Furthermore, a catalogue of generic integration rules is introduced to provide a set of solutions for recurring metamodel integration problems. The applicability of the presented approach is shown using the case of integrating the Business Process Modelling Notation (BPMN) metamodel with the metamodel for organizational modelling of the ADONIS BPMS method.

Keywords: Method Integration, Metamodelling, Mappings, Integration Rules, Case Study.
INTRODUCTION

Models facilitate the communication between people by concentrating on the essentials of a problem under consideration. Conceptual modelling is accepted as one of the core disciplines within business informatics (Buhl et al. 2005). Models in business informatics are used to describe, analyse and improve the relevant aspects of an enterprise or public organisation such as the strategy using balanced scorecard models, the products and services using product models, the work procedures and workflows using business process models, the working environment using organizational structure models, or the information systems and IT infrastructure using architecture models, application landscape models, components models etc. In practice, different types of such models (metamodels) often have to be assembled to project specific modelling solutions, i.e. different modelling languages have to be integrated (Kühn & Karagiannis 2005).

Modelling frameworks are used to structure different kinds of models and guide the selection of the appropriate modelling languages for a given problem domain. Some of such frameworks are ARIS (Scheer 1992), E-BPMS (Bayer et al. 2001), MEMO (Frank 2002), TOGAF (2006) and Zachman Framework (Zachman 1987). Method Engineering focuses on procedures, principles, languages and tools to support the development of methods (Brinkkemper 1998). One area in the domain of Method Engineering is Situational Method Engineering (SME), which concentrates on the creation of problem specific methods, depending on the problem under consideration and on the goal of the corresponding modelling project. A special aspect of SME is the reuse and assembly of existing method fragments (method chunks) to create domain specific solutions (Ralytė et al. 2006). Typically, a method or its fragments consist of a product part which describes artefacts used in the method, and a process part which defines the procedure how artefacts are used to produce specific method output. According to the Enterprise Model Integration approach (EMI), a method is a triple of a modelling language (product part), a modelling procedure (process part), and additional mechanisms & algorithms (Kühn et al. 2003, Karagiannis & Kühn 2002). Within a method, modelling languages are used by means of mechanisms & algorithms and according to a defined modelling procedure. Metamodelling approaches are seen as a convenient way to describe the elements of a method.

The focus of this work is to facilitate modelling based on the idea of Situational Method Engineering. To achieve this goal, mechanisms for flexible integration of methods and/or method fragments are needed. We see metamodel integration as a crucial issue to be tackled in this task. Therefore an approach to metamodel integration is introduced. It extends the EMI approach by introducing two key concepts: mappings and rules for metamodel integration. The EMI approach was successfully applied in various modelling projects in research and industry (Braun & Winter 2005, Moser & Bayer 2005).

The remainder of the paper is structured as follows: chapter 2 provides a classification of metamodel heterogeneity and introduces an approach to metamodel integration. In chapter 3, as first, definitions of mappings and integration rules for metamodel integration are presented. Then, a catalogue of integration rules is introduced by showing three generic integration rules. Chapter 4 gives an example how to use the presented mappings and integration rules in a practical case: the integration of the Business Process Modelling Notation (OMG BPMN 2006) with concepts of organizational modelling (BOC BPMS 2006). Chapter 5 provides an overview of related work. Chapter 6 summarizes the paper and gives an outlook to future work.

2 METAMODEL INTEGRATION

In this section we explain the underlying approach of metamodel integration. First, we discuss the metamodel integration problem and provide the corresponding classification of metamodel heterogeneity (Section 2.1). We then introduce the approach of metamodel integration in order to deal with the raised metamodel integration issues (Section 2.2).
2.1 Classification of Metamodel Heterogeneity

The fundamental integration problem among metamodels emerges when we try to join together vertically and/or horizontally different metamodels. Metamodels are (1) vertically different, when they vary in the level of details they describe, (2) horizontally different, when their concepts on the same abstraction level describe different aspects of the system or the same aspect in a different way and (3) both vertically and horizontally different, when they show characteristics of the previous two. No matter what kind of integration orientation is considered, there is a need to overcome syntactical, structural and semantic discrepancy of metamodels, in order to join together their concepts. This categorization stems from the classification of information heterogeneity (Sheth 1999) and can be applied in the modelling context at each level of the modelling hierarchy (Karagiannis & Kühn 2002). In the following, we concentrate on the heterogeneity on the metamodel level (Figure 1).

**Figure 1. Perspectives in Model Heterogeneity**

Syntactical heterogeneity represents the difference in formats intended for the serialization of metamodels. Two metamodelling platforms can base their serialization mechanisms on different proprietary formats or even paradigms e.g. having diverse relational, object oriented or XML based schemas. One approach to overcome this level of heterogeneity in the course of integration is to agree on standard paradigms and formats such as XMI (OMG XMI 2005), as well as to provide specific mappings in the case of mismatch with the agreed standards.

Structural heterogeneity can be expressed through representational and schematic heterogeneity. Metamodels are represented using different metamodeling languages, i.e. metamodels (meta² models), each of them showing difference in its expressive power of available modelling primitives (classes, attributes, supported relationship types etc.). Even when agreed on the common meta² model, metamodels vary schematically when the same concepts being described are modelled in a different way (thus having different conceptual schemas). There are two primary reasons for schematic conflicts: equal concepts are modelled either with different modelling primitives or with different number of primitives. For example, in one metamodel the concept “Performer” is defined as a simple attribute of a class, whereas the same concept in another metamodel is modelled with two classes: “Worker” and its generalisation “Processor”.

Semantic heterogeneity subsumes differences in the meaning of the considered metamodel concepts. Concepts coming from different metamodels can use the same linguistic terms to describe different concepts or use different terms to describe the same concept etc. Recalling the categorisation of ontology-level mismatches (Klein 2001), we distinguish semantic mismatches between metamodels by defining the possible inter-relations between metamodel concepts such as semantic equivalence, semantic relation and non-relation. Semantically equivalent concepts hold the same meaning, even if different terms are used such as synonyms. When not equivalent, concepts are related through semantic relation types such as “is-a”, “has-a”, “type-of” and “associate”. Unrelated concepts are completely orthogonal in the meaning, but can be described with same terms such as homonyms.
Metamodel heterogeneity imposes a layered approach for its resolution. Once the syntactical conflicts among metamodels are eliminated, i.e. having a common exchange format such as XML, metamodels are compared in order to discover and reconcile structural and semantic discrepancies. In this paper, we concentrate on horizontal, structural and semantic differences between metamodels and provide an approach for their integration.

2.2 Metamodel Integration Approach

The fundamental idea of the introduced approach is based on the use of concept mappings and integration rules. We divide the “what” from the “how” in the course of integration, by separating the mapping specification from the rule based integration definition. Dealing with metamodel heterogeneity starts with the modelling of the possible correspondences between modelling concepts using concept mappings. Once the mappings are modelled answering the “what”, they form the basis for answering the “how”, for the definition of appropriate integration rules. A rule defines steps how two or more related concepts can be integrated in the target metamodel based on the structural and semantic constraints of the corresponding mapping. The rule output is then the integrated structure of the source concepts.

Figure 2. Architecture of the metamodel integration approach

Metamodelling architectures, such as MEMO (Frank 2002) or MOF (OMG MOF 2004), provide a layered approach for flexible language definition. We use and extend this idea by introducing two new languages, particularly a mapping and a rule definition language, as means of integrating existing metamodels (Figure 2). An important part of the architecture is the meta-meta model (meta² model). Meta² models provide metamodelling constructs for describing metamodels. We deliberately agreed on the use of a common meta² model, particularly the ADONIS Meta² Model (Karagiannis & Künn 2002), in order to avoid metamodel representational conflicts mentioned before. If additional meta² models are used, appropriate mappings can be defined from ADONIS Meta² Model to the other meta² models such as MOF, thus reconciling potential conflicts. This can be done by applying one of the model transformation techniques classified in the work of Czarnecki and Helsen (2003) but on the metamodel level. The meta² model is used for description of source metamodels as well as for the check of the conformity of the target metamodel. Furthermore, to treat mappings and rules as models, specific mapping and rule definition languages have to be defined. Finally, at the first level of the metamodelling hierarchy, a mapping model and an integration model are specified by conforming to mapping and rule definition languages. A mapping model, containing concept mappings, correlates two source metamodels. Referring to mappings, concrete integration rules are defined, which form the basis for the creation of an integrated target metamodel.

The separation of the mapping specification from the integration definition is crucial in order to support overall modularity, adaptability and reusability. Furthermore, independently modelled mappings, as a fundamental abstraction, could be used as a starting point not only for metamodel integration, but also to support other generic model management operations such as model transformation, model merging, model difference etc. (Bernstein 2003, Bezivin 2005).
3 METAMODEL INTEGRATION DEFINITION: MAPPINGS AND INTEGRATION RULES

Mappings and integration rules represent core elements for the integration definition. To model them, we need both mapping and rule definition language (Section 3.1, 3.2). Furthermore, the typical resolution strategy in terms of rules can be discovered for each type of structural-semantic conflict identified by a mapping. We describe these strategies as generic integration rules and collect them in a rule catalogue, thus providing reusable knowledge for the recurring metamodel integration scenarios.

3.1 Mapping Definition

To introduce the syntax (=abstract syntax) and semantics of the mapping definition language, we discuss the basic notion of mapping. The mapping represents a structural and semantic correspondence between concepts of two metamodels. One mapping takes at least one concept (class, attribute or relationship) from each of the source metamodels and relates them, in order to denote the nature of the appearing structural and semantic conflicts (see Section 2.1). The semantics are taken as the main dimension to distinguish different types of mappings. The structural dimension expresses mapping variants. The main semantic mapping subtypes are: equivalence, relation (generalisation, aggregation, composition, association, classification) and non-relation. Taking into account the structural dimension, each of the semantic mapping types differs in the mapping structure which describes a specific combination of concept types related and in the mapping cardinality which describes the number of concept instances being related. We distinguish the following mapping variants: 1) mapping structure: class-to-class (C2C), attribute-to-attribute (A2A), relationship-to-relationship (R2R), attribute-to-class (A2C), attribute-to-relationship (A2R), relationship-to-class (R2C), and 2) mapping cardinality: one-to-one (1-1), one-to-many (1-N), many-to-many (M-N). The metamodel of the mapping modelling language is depicted in the Figure 3. For the sake of brevity, mapping variants are excluded.

Two concepts cannot be related with more than one mapping, whereas one concept can be mapped to many others. In addition, cycle and transitivity rules for respective mappings have to be properly followed. The cross-relationship-type implications can also be implied on mappings, as stated by Pottinger and Bernstein (2003).

The notation (=concrete syntax) of the mapping definition language is based on the extension of UML class diagrams. The concepts are represented by the class symbols using appropriate stereotypes to distinguish classes from attributes and relationships. In addition, separate symbols, each with at least two connecting arrows, are introduced to depict different kinds of mappings. Figure 4 shows an excerpt of a mapping model conformant to the introduced language.
Here, the mapping of type *generalisation (1:N, A2C)* is used to define the relation between concepts “Performers”, “Role” and “Actor”.

### 3.2 Integration Rule Definition

The mapping definition describes “what”, i.e. which concepts have to be integrated. The “how” is answered within integration rule definition. The abstract syntax of the rule definition language introduces the notion of integration points and integration rules, both on the basis of the previously defined mappings. *Integration points* are used to separate which of the identified mappings are of importance for the underlying integration definition. Delaying this integration specific decision to the later phase, we contributed to an integration independent definition of concept mappings. Next, the *integration rules* are specified for each of the mappings embraced by integration points. Following the basic notion of a rule as a pair of the condition and its respective action, we take mapping (holder of structural-semantic conflicts of the concept relation) as a rule *condition* and define the conflict resolution strategy as an appropriate rule *action*. The action represents the core algorithm which specify how source concepts are transformed into an integrated concept structure i.e. target metamodel fragment. Therefore, actions can be specified using a text-based imperative or declarative programming (scripting) language. Particularly, defining actions as transformations which transform one or more source concepts into one or more target concepts, we can refer to model transformation languages such as OMG QVT (2005), as an applicable mechanism to describe rule actions. Finally, execution of one rule can imply execution of another one. This is defined through the rule linkage. Similar to the mapping definition, we summarize the syntax of the rule definition language using the metamodel depicted in Figure 5.

The notation of the rule definition language reuses the mapping modelling notation and extends it for the integration specific details, in particular for depicting integration points and rules. Integration points are represented with circles which enclose mappings (see Zivkovic (2006) for detailed description). On the other side, the representation of the concrete rule actions depends on the underlying notation of the rule implementation language.

### 3.3 A Catalogue of Integration Rules

Specification of the integration rules is a metamodel specific task influenced by the nature of the integration conflict being resolved. However, generic rules can be discovered as solutions for
recurring integration problems. The first step in the definition of generic rules is to identify repeating conflicting situations. As mappings are essentially conflict description holders, the basic idea is to reuse already introduced classification of the mapping types, in order to identify typical integration problems. Thus, for each of the mapping type - equivalence, relation (generalisation, aggregation, composition, association, classification) excluding non-relation types - the solution strategy is defined in terms of generic rules, which are then sorted in a rule catalogue. Collected as rule catalogue, such rules can be reused in a number of typical integration scenarios.

The categorisation of rules follows the one defined for mappings. We distinguish two main rule categories: alignment rules and connection rules. Alignment rules address the situation when two concepts show semantic equivalence among each other. Those concepts can be then merged into the one concept, mapped in order to preserve their previous structure or abstracted with more general concept. As the names suggest, we define general rules such as merge, map and abstract. Connection rules integrate concepts according to the relation mapping type. Following its subtypes, we define general connection rules such as generalize/specialize, aggregate, compose, associate and classify. Alignment rules cannot be used for the resolution of relation mappings and vice versa, since this leads to conceptual and semantic contradiction. Both alignment and connection rules are simple rules. In addition, they can be combined, thus allowing the composition of complex rules.

Beside the semantic dimension of the mapping conflicts, the structural dimension has to be also considered when defining rules. We use this second dimension to distinguish the rule variants for each of the alignment and connection rules. For that purpose, we define specific rules such as merge A2C, aggregate R2C or generalize R2R etc (see Section 3.1 for abbrev.). As a rule of thumb, rules are named with verbs to denote their functional nature. Each generic rule description begins with the problem and goal statement. This is followed by a template-based textual and graphical rule description conformant to previously syntax introduced in Section 3.2. Textual part summarizes rule building blocks. Two graphics illustrate on the one hand the addressed fragment of a mapping model with source concepts as the rule input, and on the other hand the integrated target structure as the rule output. Intentionally, rule action definitions are sketched using natural text, in order to preserve simplicity and neutrality of specific implementation language. In this paper, the following integration rules are introduced: the merge rule and the generalize rule as simple, and the embed rule, as the example of one complex rule. Both the rule description format and the extended list of identified integration rules can be found in Zivkovic (2006).

The Merge Rule: Two or more concepts from the source metamodels are semantically equivalent. To avoid the concept duplicity, these source concepts have to be merged into one integrated concept in the target metamodel (Table 1 illustrates the Merge C2C rule).

Following variants of the Merge rule are identified: C2C, A2A, A2C, R2R and R2C.

<table>
<thead>
<tr>
<th>Rule Name: Merge C2C</th>
<th>Rule action:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Details (Rule Condition):</td>
<td>1. One of the source class concepts need to be declared as primary concept.</td>
</tr>
<tr>
<td>Semantics: Equivalence</td>
<td>2. The primary class is directly transferred to a target metamodel (TMM) along with other connected concepts.</td>
</tr>
<tr>
<td>Structure: C2C</td>
<td>3. All attribute concepts and relationship concepts are detached from the overlapping class concepts and assigned to the primary class concept.</td>
</tr>
<tr>
<td>Cardinality: 1:1, 1:n, m(1xn)</td>
<td><strong>Meta model Constraints</strong></td>
</tr>
<tr>
<td>Source Concepts: * - primary concept, [ ] – connected concept</td>
<td><strong>Target Concepts</strong></td>
</tr>
<tr>
<td>Left/Right: A*, [C] / B, [D], [E]</td>
<td>A, [C], [D], [E]</td>
</tr>
<tr>
<td>Mapping Model Fragment (Rule Input):</td>
<td>Integrated Target Model Fragment (Rule Result):</td>
</tr>
<tr>
<td><strong>Class</strong> A*</td>
<td><strong>Class</strong> A</td>
</tr>
<tr>
<td><strong>Attribute</strong> C</td>
<td><strong>Attribute</strong> C</td>
</tr>
<tr>
<td><strong>Relation</strong> E</td>
<td><strong>Relation</strong> E</td>
</tr>
</tbody>
</table>

*Table 1: The Merge Rule*
The Generalize Rule: One or more concepts of the left source metamodel are specialisations of the concept in the right source metamodel (and vice versa for the generalisation). The concepts from both metamodels have to be included in the target metamodel and integrated by a generalisation relationship.

Beside the trivial case of generalisation of two classes, the *generalize* rule can be applied in the following variants: A2C, R2R and R2C. Table 2 introduces the *Generalize A2C* rule.

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Generalize A2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Details</td>
<td>Generalisation</td>
</tr>
<tr>
<td>Semantics</td>
<td>Generalisation</td>
</tr>
<tr>
<td>Structure</td>
<td>A2C, C2A</td>
</tr>
<tr>
<td>Cardinality</td>
<td>1:1, 1:n</td>
</tr>
<tr>
<td>Meta² model Constraints:</td>
<td>* - primary concept, [ ] – connected concept, ‘– new concept</td>
</tr>
<tr>
<td>Source Concepts</td>
<td>A / B, [C]</td>
</tr>
<tr>
<td>Target Concepts</td>
<td>1) A, [C]; 2) A, [C], B’</td>
</tr>
<tr>
<td>Mapping Model Fragment (Rule Input):</td>
<td>Integrated Target Model Fragment (Rule Result):</td>
</tr>
</tbody>
</table>

![Diagram of the Generalize A2C rule](image)

*Table 2: The Generalize Rule*

The Embed Rule: Two concepts of the left source metamodel are related as parent and child (either with generalisation or with aggregation/composition). The concept in the right model represents, on the one hand the child of the parent concept from the left metamodel, and on the other hand the parent of the child concept from the left metamodel. All concepts have to be included in the target model reflecting the new concept constellation and with preserved semantics – the concept from the right metamodel has to be embedded between the two concepts from the right metamodel.

The complex *embed* rule imposes the transitivity of the relation mappings (generalisation, aggregation, composition), in order to preserve the semantics between concepts. This rule has following variants: C2C, A2C and R2C. Table 3 shows the *embed C2C* rule using the generalisation mapping type.

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Embed C2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Details</td>
<td>Generalisation: C gen. A, B gen. C)</td>
</tr>
<tr>
<td>Semantics</td>
<td>Generalisation: C gen. A, B gen. C)</td>
</tr>
<tr>
<td>Structure</td>
<td>C2C</td>
</tr>
<tr>
<td>Cardinality</td>
<td>simple rules dependent.</td>
</tr>
<tr>
<td>Meta² model Constraints:</td>
<td>concept A generalize B (left SMM)</td>
</tr>
<tr>
<td>Source Concepts</td>
<td>A / B, C</td>
</tr>
<tr>
<td>Target Concepts</td>
<td>A, B, C</td>
</tr>
<tr>
<td>Mapping Model Fragment (Rule Input):</td>
<td>Integrated Target Model Fragment (Rule Result):</td>
</tr>
</tbody>
</table>

![Diagram of the Embed C2C rule](image)

*Table 3: The Embed Rule*
4 INTEGRATION OF BPMN AND ORGANIZATIONAL MODELLING: A USE CASE

The Business Process Modelling Notation (BPMN) gets increasingly more attention in business process modelling and SOA projects in industry (OMG BPMN 2006). Nevertheless, as the current BPMN specification does not sufficiently describe the organizational and resource part of a business process modelling language, BPMN misses sophisticated concepts for organizational modelling. Organizational modelling concepts are essential in business process modelling and analysis projects, e.g. using capacity-based process simulation mechanisms for resource planning. In the following case, we apply the presented metamodel integration approach to create a domain specific modelling language by integrating the BPMN metamodel with the working environment metamodel of the ADONIS BPMS method for organizational modelling (BOC BPMS 2006). Here, a metamodel of an extract of BPMN is introduced, due to the fact that the current specification of the BPMN does not define a formal metamodel of the BPMN language.

The procedure model of the metamodel integration starts with the problem and the goal definition followed by the selection of the appropriate source metamodels. Then, the modelling of concept mappings and the specification of integration rules takes place. In the first, source metamodels are represented as concept (object) models using a UML class diagram notation extended with stereotypes to distinguish between classes, relationships and attributes. As a next step, the semantic correspondences are modelled by connecting the corresponding concepts using suitable mapping types (as defined in Section 3.1). The mapping model is the result of this phase. Next, in the rule definition phase, mappings are assigned to the integration points. Figure 6a) presents an excerpt of the mapping model and identified integration points. For the sake of brevity, we model relationship concepts without explicit stereotypes, as they are not affected by presented integration scenario.

![Figure 6. a) Mapping model and integration points, b) Revisited mapping model](image-url)

Three concept mappings are modelled (CM1, CM2, CM3) along with three integration points (IP1, IP2 and IP3). Next, the integration rules are selected for each defined mapping (see Section 3.2). Based on the structural and semantic conditions of the mappings, following generic rules are selected (as introduced in Section 3.3), forming together the integration definition:

IP1: The generalize A2C rule is applied on the mapping CM1. The attribute concept “Performer” will generalize the class concepts “Organisation unit”, “Actor” and “Role”.

IP2: The merge C2C rule, applied on the mapping CM2, will integrate equivalent class concepts “Role” in both source metamodels. Here, the decision which of the concepts would be the primary during the merge is irrelevant.

IP3: Similar to IP2, based on the CM3, the merge C2C rule merges the class concept “Entity” with the class concept “Organisation unit”. The primary concept is the “Organisation unit”, thus including the structure and semantics of the concepts being merged.
Following the rule based integration definition, the target model is created. All other source concepts, which are not affected by the integration points, are transferred to the target model. The new metamodel is now checked for its conformance to the meta’ model. Eventually, new semantic correspondences between concepts and/or inconsistencies with the meta’ model can arise due to the newly created concepts. In our case, a new “Performer” class is created. Consequently, multiple generalisations were found for the concepts “Organisation unit” and “Role”. If the new concepts are created and/or inconsistencies identified, the integration process suggests repeating the process by returning to mapping modelling phase. Therefore, two new mappings (CM4, CM5) and corresponding integration points (IP4, IP5) are additionally defined. Figure 6b) shows the fragment of the target metamodel after applying the rules followed by modelling of newly identified mappings.

IP4: In the new concept constellation, the “Participant” presents the child of the “Performer” concept, as well as the parent of the concepts “Role” and “Organisation unit”. As the latter two are direct children of the “Performer” class, this concept structure implies the use of the complex embed C2C rule, to embed “Participant” between “Performer” and its children concepts.

IP5: Based on the mapping CM5, the merge A2C rule is selected and applied. The concept class “Performer” is marked as primary. To obtain the semantics of the merged attribute, the same named aggregation relationship from the attributes’ container class to the “Performer” is created.

If no other new concepts are created and no inconsistencies are found, the integration process is complete resulting in an integrated metamodel as depicted in Figure 7.

![Figure 7. Integrated metamodel](image)

## 5 RELATED WORK

Method Engineering concentrates on procedures, principles, languages and tools to support the development of methods (Brinkkemper et al. 1998). In Situational Method Engineering existing method fragments (method chunks) are assembled to project specific or domain specific methods (Ralyté & Rolland 2001). To represent the method elements, often metamodelling approaches are used. A metamodel is a structural concept description, which can be compared to a database schema, an UML class diagram or ontology. Thus, we refer primarily to existing integration approaches from the fields of databases (Batini Lenzerini & Navathe 1986), model-driven engineering (Bezivin 2005) and ontology engineering (Noy 2004). Discovering of mappings (semi-)automatically is the essential integration task and an active research issue common to these communities (Rahm & Bernstein 2001, Kalfoglou & Schorlemmer 2005). Once the mappings are discovered, they need to be specified, in order to be used in the transformation or integration tasks. For instance, meta-ontologies for mapping definition are used to map and transform different ontologies (Maedche et al. 2002, Omelayenko et al. 2002). Based on mappings, knowledge-based merge operations are defined to support semi-automatic ontology merging and alignment (Noy & Musen 2003). Similarly, the mappings form the basis for approaches to schema integration and are seen as an essential concept in general model management (Bernstein 2003). The use of mappings, as a prior step to the definition of model transformations, has recently also gained attention in the field of model-driven engineering (Lopez et al. 2005).
6 SUMMARY AND OUTLOOK

In this paper we presented an extension of the Enterprise Model Integration approach (Kühn et al. 2003, Kühn & Karagiannis 2005). This extension adds two new concepts to EMI: mappings and integration rules for metamodel integration. A classification of metamodel heterogeneity was provided, in order to clarify the problem domain being addressed. Then, we introduced the metamodel integration approach, dividing it into a mapping definition and a rule definition. For both parts, definition languages were specified using metamodels. Furthermore, a set of generic integration rules was presented to be reused for recurring integration problems. To demonstrate the applicability of the approach, selected rules were applied in a practical case: the integration of the BPMN metamodel with a metamodel for organizational modelling of the ADONIS BPMS method.

The outcome of having the metamodel integration mechanism is the better support for the assembly of the method chunks in the course of the creation of domain specific modelling languages (DSML). As a consequence, we can achieve increased quality of the DSML by assembling and reusing quality-approved method chunks as well as shorter time-to-market by having significantly faster method definition and integration lifecycles. Based on our experiences in business-oriented modelling and modelling tool development in research and industry, we are convinced that domain specific modelling languages noticeably facilitate the modelling task, thus resulting in improved quality and the better acceptance of solutions.

One aspect of our future work will focus on automation improvements of the underlying approach, in order to provide better support for seamless method integration.

References