

A Vision for a Semantic Infrastructure Supporting Model-based Tool Integration

G. Kappel, G. Kramler

Business Informatics Group,
Vienna University of Technology
{gerti|kramler}@big.tuwien.ac.at

**E. Kapsammer, T. Reiter,
W. Retschitzegger**

Dept. of Information Systems,
Johannes Kepler University Linz
{ek|tr|wr}@ifs.uni-linz.ac.at

W. Schwinger

Dept. of Telecooperation,
Johannes Kepler University Linz
wieland.schwinger@jku.ac.at

Abstract

With the rise of model-driven software development, more and more development tasks are being performed on models. Seamless exchange of models among different modeling tools increasingly becomes a crucial prerequisite for effective software development processes. Due to lack of interoperability, however, it is often difficult to use tools in combination, thus the potential of model-driven software development cannot be fully utilized. To tackle this problem, our main contribution is ModelCVS, a sketch of a system aiming at model-based tool integration. ModelCVS enables transparent transformation of models between different tools' languages and exchange formats, as well as versioning exploiting the rich syntax and semantics of models, thus going beyond existing low-level model transformation approaches. For this, ModelCVS utilizes semantic technologies and integration patterns, allowing integration at the metamodel level. To foster reuse, a tool integration knowledge base captures essential knowledge about modeling languages and tools in terms of ontologies.

Keywords: model transformation, tool integration ontologies and patterns, metamodels, semantic versioning

1 Introduction

The shift from code-centric to model-centric software development places models as first-class entities in model-driven development processes, to exploit the higher level of abstraction, the richness of visualization, and the power of expressiveness, as compared to general-purpose programming language code. A rich variety of tools is available supporting different tasks, such as model creation, model simulation, model checking, and code generation. Consequently the exchange of models among different modeling tools becomes an important prerequisite for effective software development processes. Due to a lack of interoperability, however, it is often difficult to use tools in combination, thus the potential of model-driven software development cannot be fully exploited – unless some way of integrating the variety of existing modeling tools can be found. What is demanded, is *model-based tool integration*, facilitating any tool appropriate for the modeling task at hand. Although one could want complete tool integration, i.e., integration also addressing processes, user interfaces, etc., complete a-posteriori integration of modeling tools is very expensive in terms of effort and scalability

compared to its benefits, and therefore out of scope of this paper.

Integration of modeling tools in terms of model exchange poses several challenges [37]. First, there is *heterogeneity* in textual representation, syntax, semantics, and scope of modeling languages and exchange formats used by different tools. Detecting and resolving these heterogeneities is a matter of both, size of modeling language and subtleties of syntactic and semantic differences. Second, *implementation* of an integration solution, i.e., basically a program that takes models in one tool's format and transforms it into another tool's format and vice versa, is a cumbersome and error-prone task. Although there are specific technologies emerging that can be used solving this task, e.g., in the context of OMG's *model-driven architecture (MDA)*¹ specific model transformation languages (QVT) and tools are being developed, these are not tailored for the integration task, and furthermore require specific skills. Third, *inconsistency* in the handling of models becomes an issue when the development process proceeds in parallel branches such that different tools concurrently modify a model and model versions must be merged. Concurrent development arises in any development team and must be correctly dealt with. Finally, *repetitive effort* occurs when tools are updated by new versions or when tools similar to already known ones need to be integrated. Although during integration of a set of tools a huge amount of integration knowledge will build up, that knowledge is not captured explicitly in a form that facilitates re-use and automation support when integrating new tools or new tool versions.

To address the above mentioned issues and based on experiences gained in various integration scenarios [17], [22], [25], [36], [38], we are currently realizing *ModelCVS*, which - as the name suggests - functions similar to a traditional CVS server. It enables concurrent development by storing and versioning software artifacts that clients can access by a check-in/check-out mechanism. To adequately support the model-driven software development process, ModelCVS enables tool integration through transparent transformation of models between different tools' modeling languages, as well as versioning capabilities exploiting the rich syntax and

¹ <http://www.omg.org/docs/omg/03-06-01.pdf>

semantics of models. Furthermore, a knowledge base will store machine-readable, tool integration relevant information to minimize repetitive effort and partly automate the integration process.

For presenting ModelCVS, Section 2 will introduce a running example, which will be referred to throughout the paper. Section 3 lays out the core concepts of ModelCVS, while Section 4 proposes an architecture for the system. An overview of related work is given in Section 5, followed by concluding remarks Section 6.

2 Motivating Example

To illustrate the specific challenges we want to tackle, we consider a real-world scenario as encountered in a project, which involves a partner of Computer Associates (CA) and the Austrian Ministry of Defence. This scenario also serves as a running example throughout this paper and assumes the integration of three tools, CA's CASE tool *AllFusion Gen* (*Gen* for short)², the UML tool Rational Software Modeler³, and the Oracle BPEL Process Manager⁴. *Gen* is a legacy tool under which many existing applications have been developed. UML should be employed for new projects, to link up with current technologies. And BPEL (Business Process Execution Language for Web Services)⁵ is required for developing certain web-enabled workflow applications. The UML and BPEL tools are stand-alone tools, with integration support restricted to file exchange using some particular file format, i.e., XMI (XML Metadata Interchange)⁶ for UML and XML for BPEL. *Gen* is actually a suite of tools covering a wide range of tasks, following a common modeling paradigm. However, *Gen* is not open in a sense that it could readily be integrated with external tools. Hence, without proper infrastructure support, integration of these tools poses severe problems as introduced above, which are very costly to solve.

Different format, representation, scope, syntax, and semantics. First of all, the model *exchange formats* of these tools are different. The *differences in representation* – textual data by *Gen*, XMI by the UML tool, and XML by the BPEL tool – are the least problem, since specific tool adaptors can cope with that. A bigger problem, however, is *difference in scope*. *Gen* supports a variety of modeling domains, ranging from database via GUI to definition of functions. UML also has a rather broad scope, which is a subset of *Gen*'s. BPEL, in contrary, has a very limited scope focusing on process modeling, which is related to *Gen*'s process model and UML's activity diagram. Therefore, it is not possible to simply take a *Gen* model and directly translate it to UML or BPEL as only parts of it can be translated. Conversely, to allow for a translation back to *Gen*, precautions need to be taken to enable reassembly of any changed parts with the overall *Gen* model. No less of a problem are the *differences in syntax and semantics*. E.g., the control flow primitives of UML activity diagrams [19] and BPEL are somewhat

different, although they express the same concepts, e.g., parallelism. In some cases, however, there are also differences in expressiveness that cannot be translated. An integration infrastructure has to deal with that, too.

Scalability problems for large metamodels. The *metamodels* especially of *Gen* and UML are *very large and complex*. For instance, *Gen*'s metamodel comprises more than 800 classes and the metamodel of UML2 more than 260 classes. Even if specific implementation technology for model transformations is used, e.g., the forthcoming *QVT (Query/Views/Transformations)-standard* [33], it is clear that implementing a transformation for *Gen* and UML will require a lot of effort. Hence, the problem is not just to implement a single translation, but to also deal with the scalability problem. If BPEL is added to the tool chain, two new translations have to be implemented. If even more tools need to be integrated, simple point-to-point integration quickly comes to its limits and the need for more powerful architectures arises.

Conflicting modifications. When, in the course of concurrent development, changes to a model are merged, much care has to be taken to keep the model consistent. As we have seen above, if a model is translated from *Gen* to BPEL, only some part of it can be translated, thus the BPEL developer may not be aware of all implications that any changes to the BPEL-relevant part may have on the overall model. The situation is even exacerbated when models are modified concurrently. Since we want to enable *collaborative development* and flexible working processes, we have to deal with the facts that experts working on some specific part, e.g., the BPEL part, use very specific tools and that development in related parts may proceed concurrently. Although the task of *merging concurrent modifications* is not genuine to model-based tool integration and is addressed by existing versioning tools, in our case we cannot rely on individual tools to help detecting and resolving potential inconsistencies, since a change performed in one tool may propagate to parts of the model that are outside that tool's scope. Therefore, appropriate infrastructure support is required.

Different versions of metamodels. Finally, let's consider the process of updating a tool to a new version with an updated metamodel. If not already supported by the tool, the update process involves implementing translators for migrating existing models to the new version and possibly back again. Less obviously, also all of the existing integration specifications need to be updated to the new version, and the tool will certainly not support this task. Lot of repetitive effort will be required unless it is possible to automatically migrate existing integration specifications. A similar situation and potential of re-use occurs when tools supporting equal modeling languages have to be integrated. Typically these tools, although supporting the same modeling language, interpret and realize the associated metamodels in different ways, in case that the semantics of the metamodel is not clearly defined. In this case, reuse could also be supported through higher-level integration knowledge, thus reducing the manual integration effort to validation, precision, and completion.

² <http://www3.ca.com/Solutions/Product.asp?ID=256>

³ <http://www-306.ibm.com/software/awdtools/modeler/swmodeler/>

⁴ <http://www.oracle.com/technology/bpel/>

⁵ <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>

⁶ <http://www.omg.org/technology/documents/formal/xmi.htm>

3 Layered Approach to Tool Integration

To address the challenges identified for providing interoperability between tools, the approach taken to the realization of ModelCVS (cf. Fig. 1) is separated into three distinct conceptual layers that enable to integrate models produced by adjacent modeling tools.

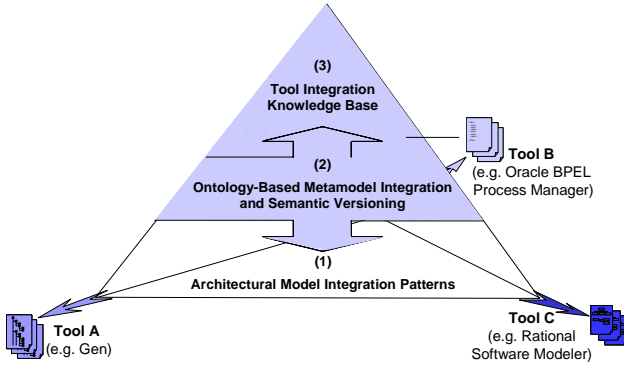


Figure 1: Layered Tool Integration Approach

The bottom layer (1) is formed by *architectural model integration patterns* (*integration patterns* for short) that ensure openness, scalability, and evolvability of a tool integration solution. Further elaborated on in subsection 3.1, these will serve as a basis to define specific bridging tasks and to develop appropriate *bridging operators* that support the identified integration patterns. The second layer (2) deals with the use of *semantic technologies* in the form of *ontologies* for the integration of tool metamodels as well as for the realization of semantically aware model versioning mechanisms. Subsection 3.2 addresses the integration problem at the semantic level using ontologies in more detail and shows how automation support can be achieved. Top layer (3) aims at providing *reuse capabilities* in the form of a *tool integration knowledge base* for ModelCVS' semantic infrastructure. As further detailed in subsection 3.3, the knowledge base enhances support for ontology-based metamodel bridging, as well as improved detection of versioning conflicts.

3.1 Patterns for Model-based Tool Integration

The basis for our solution to model-based tool integration is a set of integration patterns that define requirements and working context for the *bridging language*, which contains *bridging operators* that specifically support the identified integration patterns at a suitable abstraction level, and hence can be more efficiently used than, e.g., generic model transformation languages [36]. By finally deriving *model transformation code* to enforce specific bridging semantics on models, the bridging language is made executable. The four proposed integration patterns cover various situations relevant for model-based tool integration. Addressing openness, scalability, and evolvability requirements, these patterns are elaborated in the following paragraphs.

Metamodel translation. The basic case of tool integration occurs when two different tools' modeling languages conceptually overlap to a large extent. This means, that both modeling languages cover the same or very similar domains, in a way that semantically

equivalent concepts can be identified in either metamodel and models can be *translated* correspondingly. As an example, we refer to two modelers jointly modeling a workflow: One of the modelers employs a dedicated BPEL modeling tool, whereas the other colleague makes use of UML activity diagrams. Both modelers are able to transparently check-out versions of the latest model, edit it, and check it in again without having to deal with modeling languages other than their own, as the language heterogeneity between modeling languages is implicitly taken care of through translation by ModelCVS.

Variations of this pattern address *directionality* and *completeness* of translation. A translation may be bidirectional, allowing two-way transformations between metamodels. In case a tool, for instance a code generator, is purely consuming and not producing models, unidirectional translations suffice. In case modeling languages do not entirely overlap, meaning that some concepts expressible in one modeling language cannot be expressed in another, a translation may be lossy. A solution to solve this problem is to explicitly store information that would get lost in the course of a transformation and to reincorporate it when performing the roundtrip [7]. A further variation, which is advisable in case multiple tools with similar domain have to be integrated, is to construct a so-called *pivot metamodel*, which can be seen as representing a universal language covering a certain domain. In practice, however, such a universal language encompassing all possible concepts that can occur in a certain domain is hard to find. Nevertheless, finding a pivot metamodel for a specific enough modeling domain can be feasible, yielding the advantage of reducing the amount of mappings required when translating between *n-many* tools from $n*(n-1)$ to *n*.

Figure 2 shows the translation approach involving the process metamodel of Gen (MM_{Gen}), UML's activity diagram metamodel (MM_{UML-AD}), and BPEL's metamodel (MM_{BPEL}).

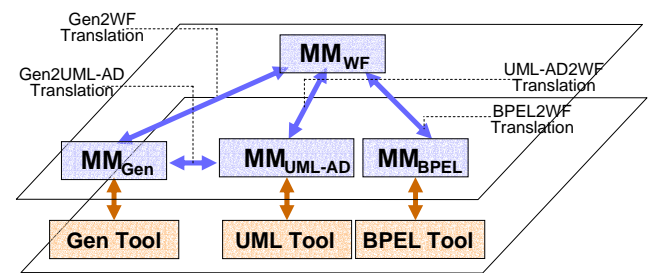


Figure 2: Metamodel Translation

The domain common to all three could be described in a generic, tool independent workflow metamodel (MM_{WF}), which serves as a pivot facilitating tool integration in a scalable way. As starting point, let's assume that a *Gen2UML-AD* translation already existed and that for integration of further metamodels like MM_{BPEL} , the establishment of a pivot metamodel was chosen. Then a specific requirement on bridging operators resulting from this scenario is re-usability of the existing bridge *Gen2UML-AD* for construction of the pivot metamodel and the translations *Gen2WF* and *UML-AD2WF*. Now the

pivot metamodel MM_{WF} can be used in order to generate a translation to MM_{BPEL} , namely $BPEL2WF$.

Metamodel alignment. The *alignment* pattern deals with interrelating rather than translating models. This requirement occurs when a system to be modeled crosses several domains, and several modeling languages or better to say modeling tools, tailored to specific domains, participate in modeling that system. Although these domains are typically very different, they will overlap to some extent, making it necessary to integrate these domains to cohesively represent the entire system's domain. As modeling languages in this case do not cover same or similar domains, the focus is shifted from a complete translation of concepts onto an *alignment* of concepts, manifesting in the creation of relationships that *enforce certain constraints or alignment rules* imposed on the integrated domains.

The example scenario depicted in Figure 3 shows the alignment of two different tool metamodels (MM_{GUI} and MM_{UML-CD}), provided by a tool for modeling GUIs and a UML tool, which are used jointly to model a single system.

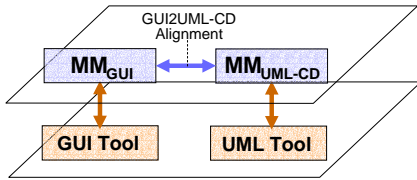


Figure 3: Metamodel Alignment

It is important to note, that these two metamodels do not cover same or similar domains. It would therefore not make sense to find a mapping which would translate a UML class diagram of a system into a GUI, as the GUI design would rather be undertaken independently. Depending on the underlying system, however, a specific overlap is necessary to integrate the two domains. As an example similar to the model-view-control paradigm, the tool metamodels are aligned (*GUI2UML-CD*) to establish a behavior that transparently sets the labels of GUI components to the value of a certain attribute of a model element in a UML class diagram. To furthermore avoid inconsistencies, model elements representing GUI components should be transparently deleted if a corresponding model element in the UML model is deleted, which also has to be defined as an alignment rule.

Metamodel modularization. The modularization pattern addresses the scalability issue of two related integration scenarios. On the one hand, to fulfill the scalability requirement, the effectiveness of a tool integration process may not be affected by the *size of the metamodels* involved. Hence, a model-based tool integration approach must allow to deal with large, monolithic tool metamodels in a manageable way. As an example, the integration of two large tool metamodels, like those of UML and Gen, has to be supported in a way that keeps the integration task comprehensible. On the other hand, scalability is required when it comes to the integration of *tools with a varying scope*, regarding the domain specificity of the underlying modeling languages. As an

example, it should be possible to integrate a UML tool with a BPEL tool. Thereby, the domain specific BPEL tool will conceptually overlap with the domain covered by the UML tool to a certain extent, only. Nevertheless, the integration of the BPEL metamodel with the overlapping part of the UML metamodel should not become unwieldy. To keep the integration of large metamodels with varying scopes manageable, *modularization* enables the decomposition of these metamodels according to certain concerns, resulting in smaller metamodels, so-called *metamodel fragments*, each expressing a certain *aspect* of the entire metamodel. Analogous to the decomposition of a metamodel, models conforming to such a metamodel are modularized accordingly to allow model exchange in a scalable way.

The example depicted in Figure 4 shows the integration of tools with differing scopes using modularization. The top section of the figure shows the Gen metamodel (MM_{Gen}) modularized into several smaller metamodel fragments representing more specific domains (MM_{GenGUI} , MM_{GenWF} , $MM_{GenClasses}$, and $MM_{GenStates}$). As shown, the metamodel fragments may overlap each other, which can result in interdependencies that shall be taken care of in a transparent way as described in the *alignment* example. The bottom left part of the figure shows the integration of domain specific GUI and BPEL modeling tools, which are directly mapped to metamodel fragments of the Gen tool. Similar to the modularization of MM_{Gen} , the bottom right part of the figure illustrates a UML tool's metamodel (MM_{UML}) being modularized (MM_{UML-AD} , MM_{UML-CD} , and MM_{UML-SM}). The integration of large tools is made possible in a scalable way, as the metamodel fragments of either tool covering semantically equal domains are mapped onto each other instead of mapping the original huge metamodels.

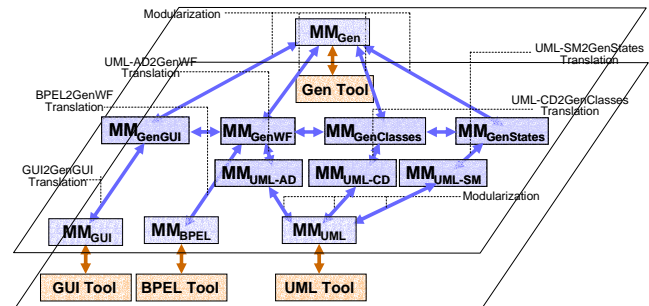


Figure 4: Metamodel Modularization

At check-out time, models conforming to metamodel fragments have to be reassembled. This implies that links between model elements that have been cut off during the modularization phase have to be re-established. The rules specifying how the various models should be reassembled have to be derived from the applied bridging operators. To enable reassembly, in certain cases information about linked model elements must be explicitly stored during the modularization phase.

Metamodel versioning. Tool metamodels may need to change if a new version of a tool becomes available. To ensure the *evolvability* requirement by not rendering existing assets unusable, it is necessary to migrate existing models towards the new metamodel.

Furthermore, in case different tool versions remain in use at the same time, it has to be possible to access models using different versions of that metamodel. As an example, a UML 1.4 compliant modeling tool may be replaced with a UML 2.0 modeling tool. Therefore, models compliant to UML 1.4 have to be migrated to the current UML 2.0 metamodel. However, a code generator taking UML 1.4 models as input should still remain in use. Hence, addressing the requirement for *evolvability* can be associated with the need for so-called *metamodel versioning*. Metamodel versioning includes keeping track of different versions of tool metamodels and migrating models towards newer versions of tool metamodels. Through defining translations between versions of a tool metamodel, various versions of tools can remain in use. Different to the general translation case, the typically rather small difference between metamodel versions can be exploited. Furthermore, also existing metamodel bridges must be taken care of by providing migration support for bridges, too.

Figure 5 illustrates the required migrations facilitating metamodel versioning when a new tool version *UML2* and a corresponding tool metamodel version for UML2 activity diagrams $MM_{UML2-AD}$ is introduced into an existing tool chain.

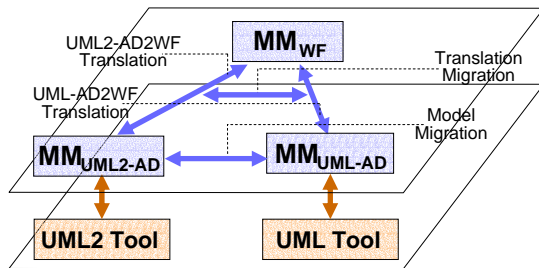


Figure 5: Metamodel Versioning

What needs to be done is (1) to define a model migration bridge from the old version of the metamodel (MM_{UML-AD}) to the new version ($MM_{UML2-AD}$), and migrate the existing models accordingly, and (2) to define a new version of the translation to the pivot metamodel (*UML2-AD2WF*), assuming that the existing pivot metamodel MM_{WF} is not affected by the changes.

3.2 ModelCVS Semantic Infrastructure

In the following, two core functionalities of ModelCVS are laid out. Both are founded on the use of ontologies to express the semantics of modeling languages. We believe that in doing so, semantic technologies can yield significant benefits for effectively driving a model-based tool integration solution as envisioned with ModelCVS.

3.2.1 Ontology-based Metamodel Integration

As a strictly manual bridging specification can become a laborious and error prone task, ModelCVS offers automation support to do so. The following paragraphs describe a sequence of steps showing how ModelCVS' semantic infrastructure can be utilized. As done before, our example refers to the metamodels of BPEL and UML Activity Diagrams (UML-AD) to be integrated. Details on Figure 6, which generally depicts our architecture used

for ontology-based metamodel integration, will be given throughout the running example.

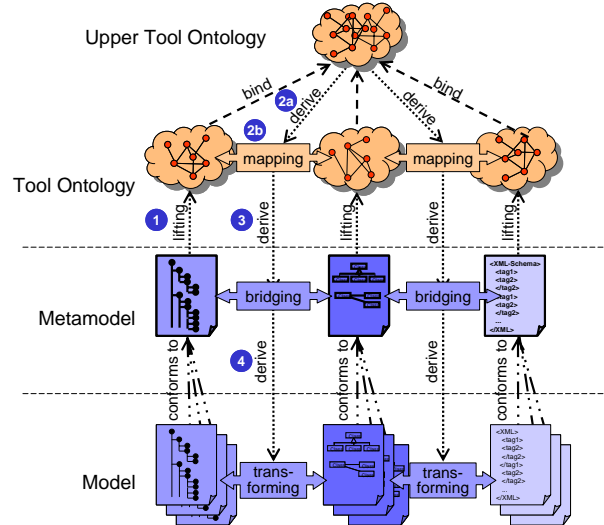


Figure 6: Ontology-based Metamodel Integration

(1) **Metamodel lifting.** The creation of an ontology from some kind of metadata like an XML schema [12] or a DB schema [43] is generally referred to as *lifting*. Metamodel lifting in particular encompasses a mapping of elements in the metamodel to concepts in the ontology, thereby performing a step of abstraction and semantical enrichment such that the ontology explicitly expresses the semantics of the modeling concepts whose syntax is defined by the metamodel. Automatic as well as semi-automatic approaches to lifting have already been presented in literature (cf. Section 5). For a more elaborated description of ModelCVS' metamodel lifting functionalities we kindly refer the reader to a technical report⁷. In our case, a generic solution for lifting arbitrary MOF models (tool metamodels) to so-called *tool ontologies* can partly automate the lifting process. However, the entailment of specific semantics for newly lifted ontologies, naturally requires user intervention. Referring to our example, this would mean to lift both the BPEL and the UML-AD metamodel resulting in the respective tool ontologies. Once these are at hand, ModelCVS' semantic infrastructure offers two ways to support a user in finding semantic correspondences:

(2a) **Upper tool ontology.** The establishment of mappings by means of an *upper tool ontology* involves explicit mapping of each tool ontology to a common upper tool ontology. Thereby it is necessary that the chosen upper tool ontology covers the domains of both tool ontologies appropriately. From the upper ontology, mappings between tool ontologies can be deduced. Continuing our running example, we assume a generic 'Workflow' ontology as a common upper ontology. As an example, we can imagine to map the concept of BPEL 'Activity' and the concept of a UML-AD 'ActionNode' both onto the same, semantically equal concept in the upper tool ontology, e.g., a concept called 'Action'. From the two mappings between tool and upper tool ontology we could for instance deduce an "equivalentClass"

⁷ ftp://ftp.ifs.uni-linz.ac.at/pub/publications/2005/0705.pdf

relationship directly between the concepts of ‘Activity’ and ‘ActionNode’.

(2b) Heuristic mappings are based on finding structural and linguistic similarities in ontologies. The estimation of naming similarity need not only be orthographically based. The possible utilization of lexical reference systems (e.g., [14]) allows to identify and relate names in question as for instance being synonyms, homonyms, antonyms and the like. In that sense, a heuristic could come to the conclusion that the concepts of ‘ActionNode’ and ‘Activity’ are semantically equivalent, as the substring ‘Action’ and the string ‘Activity’ are identified as synonyms. Furthermore, the results of heuristic matching techniques can be greatly enhanced when incorporating instance data into the ontology matching process [20] (instance-based matching), which could be accomplished by populating tool ontologies with data of a common reference example.

Although both of the above mapping methods can alleviate the burden when creating a mapping, a user is still needed to check the appropriateness of a proposed mapping and to eventually give it a finishing touch. Furthermore, it lies in the responsibility of the user to choose an appropriate method to support the mapping process.

(3) Derivation of bridging. Once a mapping between tool ontologies exists, the next logical step is to derive bridging operators to express the desired integration behavior on the metamodel level. In a derived bridge between metamodels, depending on the integration pattern in use, namely *translation*, *alignment*, *modularization*, or *metamodel versioning*, this semantic correspondence can be expressed by certain metamodel bridging operators accordingly. In case of alignment, a bridging operator might denote the propagation of a certain attribute value, whereas in the modularization case, a bridging operator could denote that two model elements should be merged into one at check-out. Getting back to our example, the translation pattern will be the most appropriate, as both metamodels cover a largely similar domain. Hence, our example’s ‘equivalentClass’ relationship on the ontology level would be derived in a bridging operator relating the metamodel elements that initially got lifted to the concepts of ‘Activity’ and ‘ActionNode’ accordingly.

(4) Derivation of transformation. In the context of a *translation* from BPEL to UML, this relationship between metamodel elements could in turn be derived into QVT code causing the creation of a new model element of type ‘ActionNode’ for every model element of type ‘Activity’ at execution time.

3.2.2 Semantic Versioning

Ensuring consistency during the concurrent development of models requires automated detection of merge conflicts. Conflict detection will be performed at both the syntactic level, based upon a models’ graph structure and on metamodel constraints, and the semantic level, based upon the meaning of (syntactic) model elements. Essentially, semantic conflict detection also takes into

account semantic changes to a model element that are implied by changes to other model elements but do not manifest in syntactical changes of that particular model element. As an example of a semantic merge conflict which can arise in merging UML models, consider Figure 7, which shows models *Model₂* and *Model₃* copied from common ancestor *Model₁*. In *Model₂*, operation *a* in class *A* has been modified. Note, that class *B* is also affected by this change as *B* inherits that operation from *A*. In *Model₃*, class *B* has been modified to include an operation *a*, which overrides operation *a* as inherited from class *A*. When merging the concurrent changes from *Model₂* and *Model₃* into *Model₄* and considering only structural changes, no conflict can be detected. However, when taking into account the meaning of the generalization relationship, it becomes obvious that the two changes are in conflict with each other.

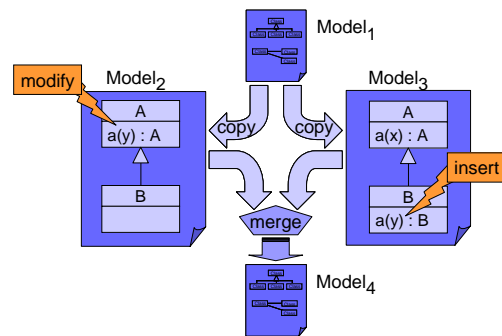


Figure 7: Semantic Merge Conflict

To enable automatic identification of merge conflicts, knowledge about such semantic merge conflicts are captured in the knowledge base (cf. section 3.3.) by enhancing the definition of language concept semantics as relevant to conflict detection, e.g., rules defining the semantics of generalization, i.e., inheritance of features with the ability to redefine inherited features. The semantic conflict detection mechanism employs these rules to deduce the semantic changes made to models and compute potential conflicts based on that. As conflict detection on the ontology level may suffer from poor performance, we aim at automatic generation of executable conflict detection programs operating on models directly. For instance, one could envision to realize the inheritance example as shown above by defining a derived attribute for inherited features using OCL and by extending the syntactic conflict checks to that derived attribute.

3.3 Knowledge Base for Tool Integration

As described in the previous paragraphs, ModelCVS’ semantic infrastructure makes use of ontologies for means of metamodel integration and semantic versioning by relying on *tool ontologies*. Just like metamodels, these tool ontologies represent valuable assets in terms of conceptualizing a domain.

Hence, similar to a class library of a programming language, it is intended to foster reuse capabilities of ontological knowledge concerning the field of tool integration by building up a so-called *tool integration knowledge base*. This knowledge base is made up of tool

ontologies (i.e. products of liftings) capturing knowledge about modeling languages, and thus foster immediate reuse capabilities. Concerning the running example, tool ontologies for Gen, UML, and BPEL would fall into this category. As one can see, in the same way as tool metamodels may either represent conceptual modeling languages (e.g., UML) or domain-specific languages (e.g., BPEL), tool ontologies will also vary in their domain specificity accordingly. Therefore, similar as more specific classes in a class hierarchy of a programming language reuse concepts of more general classes, a hierarchical structure of ontologies is to be imposed that enables reuse of semantic concepts for tool ontologies. For instance, a user entailing specific semantics during the lifting process - usually by manually editing the resulting ontology - can reuse concepts in the tool integration knowledge base by establishing subsumption relationships to concepts in the respective tool ontology.

Thus, apart from domain-specific tool ontologies, the resulting knowledge base will also comprise so-called *foundational ontologies* in a hierarchical order providing reusable semantics (cf. Fig 8). For instance, the BPEL ontology can reuse concepts from a foundational ‘Workflow’ ontology. Furthermore, such a ‘Workflow’ ontology can also play the role of an upper tool ontology (cf. Section 3.2.1) as in the BPEL/UML-AD example described earlier.

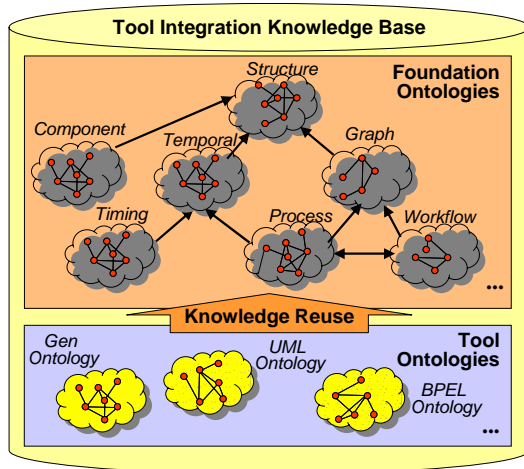


Figure 8: Reuse of Integration Knowledge

Furthermore, the ontologies within the proposed tool integration knowledge base will be populated with *specific instance data*, stemming from reference examples of case studies. These reference examples contained in the knowledge base enable the semi-automatic mapping with newly created tool ontologies that are as well populated with instance data from a suitable reference model. Thus, the process of specifying semantics for tool ontologies can be enhanced considerably. The reference models have to be made up such that they produce satisfying results with respect to enhance ModelCVS’ matching and reuse capabilities.

4 Architecture

As can be seen in Figure 9, the proposed architecture for ModelCVS is organized into three major components.

First, a *Technological Framework* provides the actual tool integration services and comprises among others, a repository supporting semantic versioning and transparent model transformation. It is supported by *Tool Adapters*, i.e., external components that mediate between proprietary tool interfaces and ModelCVS. Second, the *Metamodel Bridging Toolkit* provides support for defining bridges as to realize integration patterns, manually or automatically. Third, the *Ontology Toolkit* supports ontology-based metamodel integration in terms of lifting, mapping, and editing capabilities. In the following we will elaborate ModelCVS’ components in more detail.

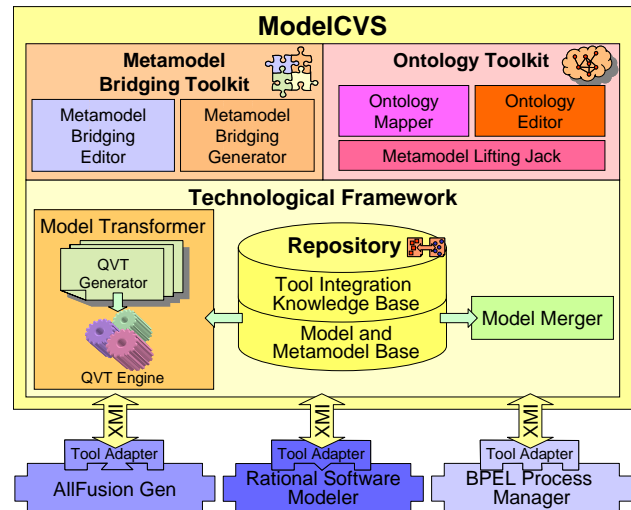


Figure 9: ModelCVS Architecture

Technological Framework. The *Technological Framework* performs the actual tool integration, based on the configurations defined using the *Metamodel Bridging Toolkit* and the *Ontology Toolkit*. Its main component is the *Repository* which provides persistent storage and versioning of complex artefacts. The *Repository* is divided into two parts. First, the *Model and Metamodel Base* is dedicated to artefacts of the model and metamodel level, comprising, e.g., models, metamodels, and bridging definitions. Second, the *Tool Integration Knowledge Base* contains the ontology level artefacts such as tool and foundational ontologies, as well as associated mappings and liftings. Concerning the *Repository*, we make use of a *versioning system*, e.g., Subversion⁸, as the repository’s back-end, providing persistence and basic versioning capabilities. As a front-end, an existing *MOF-repository*, e.g., MDR⁹, along with an existing *ontology repository*, e.g., Sesame¹⁰, can provide access interfaces for their respective clients.

The *Model Transformer* plugs into the *Repository* to provide model transformation capabilities as required for the various tasks defined by the integration patterns. A *QVT Engine* implementing model transformation as requested by QVT [33] will be used, such as ATL¹¹ or MTF¹². The metamodel bridges that are specified in a

⁸ <http://subversion.tigris.org/>

⁹ <http://mdr.netbeans.org/>

¹⁰ <http://www.openrdf.org/>

¹¹ <http://www.eclipse.org/gmt/>

¹² <http://www.alphaworks.ibm.com/tech/mtf>

high-level language (cf. Section 3.1) using the *Metamodel Bridging Toolkit* have to be translated into that transformation language. A *QVT Generator* will perform this compilation task. The *Model Merger* also plugs into the *Repository* to provide *syntactic* and *semantic merging* thus enhancing the *textual merging* capabilities already provided by the repository back-end. Appropriate concepts and reasoning tasks, as defined with respect to the tool integration knowledge base (cf. Section 3.3), have to be developed for the semantic merging capabilities. Since conflict detection is a time-critical function that has to be performed during each check-in, the *Model Merger* implementation will use precompiled conflict detection programs, derived from metamodel-level rules, such as OCL constraints.

Tools interact with the *Repository* using some *access protocol* to perform operations such as browse or model check-out. Preferably an already existing access protocol such as CVS or WebDAV¹³ can be reused for this purpose. Regarding the *data format* used for exchanging models, XMI is a natural candidate as it is based on MOF, and supported by many tools, particularly UML tools. The choice of XMI does not substantially restrict the kinds of modeling tools which can be integrated, since XMI can represent any model that exhibits a graph-based structure. *Tool adaptors* are a practical necessity, since it cannot be assumed that all tools to be integrated in a tool chain support the access protocol and data format of ModelCVS.

Metamodel Bridging Toolkit. This component provides all functionalities dealing with the handling of metamodels and especially the creation of metamodel bridges according to the integration patterns, i.e., *translation, alignment, modularization, and metamodel versioning*. A *Bridging Editor* for the bridging language can for instance be implemented by reusing a generic mapping tool like the Atlas Model Weaver [5] that can be customized to accommodate the specific concepts of the bridging language. The *Bridging Generator* makes use of any mappings created at the ontology level to automatically derive bridges between metamodels. Automatically generated bridges will have to be reviewed and refined by the user, using the *Metamodel Bridging Editor*.

Ontology Toolkit. Finally, the *Ontology Toolkit* provides the means for metamodel lifting as well as mapping and editing of ontologies. Its key component is the *Metamodel Lifting Jack*, which provides means for the creation of an ontology from a metamodel through *lifting*. The lifting mechanism can be built on experiences gained from lifters working with database or XML schemata. To facilitate the lifting implementation, we aim on defining a concise mapping from the MOF 2.0 meta-metamodel onto an ontology language definition, such that any MOF compatible modeling language can be lifted and expressed in an ontology language like OWL for instance, simplifying the further process of semantic enrichment. To actually manipulate and make use of the resulting ontologies further, tools like Protégé, the

Eclipse plug-in Semantic Web Development Environment (SWeDe)¹⁴, the JENA API¹⁵ as well as several specialized inference engines like F-OWL¹⁶ can be used, contributing to the *Ontology Mapper* and the *Ontology Editor*.

5 Related Work

Tool Integration. Regarding the Stoneman Model¹⁷, Brown [6] categorized tool integration into a conceptual (“what is integration?”) and a mechanical level (“how to provide integration?”). Regarding the conceptual level, Wasserman [44] is regarded as the first author who has suggested a categorization to describe the integration of tools from a *functional point of view* comprising integration in terms of *platforms, GUIs, data, control, and processes*. Research efforts at the mechanical level of tool integration include (1) a series of *standardization efforts* and *middleware services* like CAIS [31], PCTE [2], CDIF [13], CORBA¹⁸, and OMG’s recent RFP OTIF¹⁹ (open tool integration framework) to support tool interoperability, (2) *architecture models, infrastructures, and tool suites* like the ECMA toaster model [11], the ToolBus architecture [4], and finally (3) basic *tool integration mechanisms* such as data sharing, data linkage, data interchange, and message passing [37]. Some of these efforts were often grounded in large initiatives but have not been widely accepted, such as the PCTE with its somewhat heavyweight architecture or CDIF, which in the meanwhile has been replaced by MOF and XMI, for example. Despite of all these important efforts, tool integration is still a challenge, leading most often to strongly technology-dependent, hand-crafted solutions that suffer from high maintenance overheads and most importantly, *poor scalability*.

Model transformation languages. Existing approaches in this area having been either submitted to OMG’s QVT request for proposals or being already part of existing MDA tools ranging from *algorithmic* and *imperative approaches*, via *graph-transformation-based approaches* to *template rule-driven*, and *hybrid approaches* [9]. Tratt *et al.* [41], e.g., provide an extensible, imperative model transformation language with some rule-based elements for pattern matching purposes, whereas Becker *et al.* [3] use purely rule-based mechanisms based on graph-transformations. BOTL²⁰ allows the definition of modular, rule-based transformations, with independent rules for sets of metamodel elements. Based on these several kinds of QVT-like transformation language proposals, infrastructures and frameworks have been built for tool integration (cf. the special issue of SoSym on model-based tool integration [37]). For example, *WOTIF (Web-based open tool integration framework)*²¹ uses a graph-transformation mechanism and realizes different tool integration patterns, but requires that every client tool supports certain APIs for installing plugins, which is in

¹³ <http://www.webdav.org/>

¹⁴ <http://owl-eclipse.projects.semwebcentral.org/>

¹⁵ <http://jena.sourceforge.net/>

¹⁶ <http://fowl.sourceforge.net/>

¹⁷ <http://www.adahome.com/History/Stoneman/stoneint.htm>

¹⁸ <http://www.omg.org/corba>

¹⁹ <http://www.omg.org/docs/mic/04-08-01.pdf>

²⁰ <http://www4.in.tum.de/~marschal/botl/>

²¹ http://escher.isis.vanderbilt.edu/tools/get_tool?WOTIF

contrast to our approach. *GeneralStore* [35] being in fact a MOF-based repository, allows bi-directional transformations between models, but uses XSLT or ad-hoc approaches for model transformation, only. Finally, although *MDDi (Model-driven Development Integration Project of Eclipse)*²² is still in its drafting phase, it provides some interesting ideas for model integration in terms of a bus architecture similar to AMMA [TODO].

Although, QVT-like model transformation languages are a cornerstone also of our vision, existing proposals are too generic and lack appropriate abstraction mechanisms for different kinds of *model integration patterns*, which are highly needed in practice and well-known from other research areas such as *federated and multi database systems* [39], *megaprogramming* [45] and *web service composition* [24]. Such integration patterns (cf. Section 3) would require a series of basic model transformations which will simply not scale up when manually specified for complex models.

Integration patterns and bridging operators. There are only few related approaches providing abstraction mechanisms in terms of, e.g., high-level bridging operators or modularization techniques in the areas of *model management and model integration*. For instance Rondo [27] provides high-level operations facilitating the integration of relational and XML schemata. In the modeling realm, Clarke [8] and Straw et al. [40] introduce *Model Composition Semantics* and *Model Composition Directives* respectively, which represent *composition mechanisms* for UML class diagrams. Both approaches are fit to UML models only, and do not immediately provide an appropriate abstraction as would be required for the integration patterns identified above (c.f. Section 3). Furthermore, ideas from the area of *aspect-orientated modeling* dealing with modularization of cross-cutting-concerns and the weaving of aspects are relevant to the definition of bridging operators for our integration patterns. In this respect, *C-SAW* by Gray et al. [15] which is a so called cross-cutting-concern weaver, is of interest. However, it lacks support for abstract enough integration mechanisms and is based on a metamodel different from MOF, making the approach not immediately applicable for us.

Ontology-based Integration. As could be seen at the remarkable Dagstuhl workshop on semantic interoperability and integration in 2004, ontologies have become very popular to facilitate various semantic integration tasks, as one can heavily rely on the high expressive power of ontology languages and on reasoning techniques accordingly. As our approach utilizes ontologies as a base mechanism to semantically enrich tool metamodels, although concepts from related work in the area of *lifting metadata to ontologies* - for instance the OntoLIFT prototype [43] for database schemata and the automatic mechanism introduced by Ferdinand et al. to lift XML schemata - are of relevance to our approach, they are not immediately reusable in our metamodel-centric context.

As ModelCVS performs tool metamodel integration on basis of semantics covered by tool ontologies, *integrating these individual tool ontologies* is an issue. The central burden making ontology integration a rather comprehensive challenge are heterogeneity issues that have to be coped with [23], which are similar to heterogeneities in database research [39]. Thus, our approach has to deal with different forms of *heterogeneity*, establish a certain *ontology integration architecture*, and provide appropriate mechanisms for *mapping discovery, representation and reasoning* [30]. Although having different goals in mind since we use ontologies as a basic vehicle for the integration of tool metamodels, we can benefit from a large body of literature which can provide useful input for our approach. For a comprehensive overview of this active research area compare, e.g., [1], [21], and [30].

Concerning architectures for ontology integration, one can basically distinguish three alternatives (cf. e.g., [30]): (1) a *direct mapping* between ontologies, (2) an *indirect mapping* via a *common, shared ontology* also called *upper ontology* (sometimes also referred to as *toplevel, or reference ontology*), e.g., the Standard Upper Merged Ontology (SUMO) [28] and DOLCE [14] and (3) a mapping based on a *library of already mapped ontologies* [42]. We intend to use a *hybrid approach*, involving all three architectures in order to ensure a balance between reuse capabilities and induced overhead. Based on a certain ontology integration architecture, mappings between ontologies have to be established, i.e., similar concepts have to be related to each other. Mapping discovery techniques as employed for instance by Chimaera [26], PROMPT [29], KRAFT [32] and PUZZLE [20], deal with finding such correspondences between ontologies. This can be done either in a *fully manual way* or by utilizing *heuristic-based or machine learning techniques* that use various characteristics of ontologies, such as their schemata (*schema-based matching*), their instances (*instance-based matching*) as well as *lexical reference systems* [34], [10], [30].

6 Concluding Remarks

Currently our focus lies on the realization of components constituting the Technological Framework (cf. Fig 9). Furthermore, we are in the initial stages of developing an appropriate bridging language and concepts for the implementation of ontology-based integration. We are aware that a successful realization of a system like ModelCVS as laid out in this paper faces a number of issues mainly concern technological feasibility and practical applicability issues of the final result:

Incompatible standards. At the time of writing it is a fact that the interchange of models via XMI still poses a practical problem, which stems from *incompatible XMI output* produced by different modeling tools. Nevertheless has XMI become a widely adopted standard by most modeling tools and it can be expected that tool vendors will eventually converge on producing interchangeable XMI serializations. Furthermore, although MOF is a widely accepted standard, several interpretations - and resulting from that - different

²² <http://www.eclipse.org/proposals/eclipse-mddi/>

implementations in terms of model repositories exist. For seamless interchange of tool metamodels with ModelCVS, strict adherence to a common standard like MOF is necessary. In general, however, the problems with incompatible XMI files and differing meta-model standards are issues which can be solved with the construction of specific tool adapters.

Quality of integration. Considering the fact that we use an ontology rather than a mapping to some *semantic domain* [17] to denote the semantics of a modeling language, this is reasonable since ontologies have been developed as a means for integration, whereas semantic domains are more appropriate for reasoning about intrinsic properties of a model. Furthermore, it is often difficult or even impossible to define a mapping from a modeling language to a semantic domain, as is the case with UML [17]. The consequences of not using a semantic domain are that a mapping between ontologies and therefore a derived bridging between metamodels may not be precise enough as to ensure exact equivalence of models – a property that would be important if executable code should be generated from models. Ontologies can, however, be used to explicitly keep track of the quality of a mapping, i.e., whether a mapping is precise or not, and which caveats have to be considered. Therefore, the knowledge base and bridging operators should support this kind of quality control.

Integration overhead. Considering the manual effort involved in lifting a metamodel, the question arises whether that effort pays off by the improved support in defining metamodel bridges and in semantic versioning. We assume that moving to the more abstract semantic level becomes beneficial especially if a metamodel is large and complex, as is the case, e.g., in our case study with more than 800 classes of Gen. The ontology will express semantics of concepts and consequently integration mappings much more concisely, thus helping to keep mappings comprehensible and manageable. Another net benefit resulting from lifting metamodels is to build up a comprehensive tool integration knowledge base containing readily reusable semantic definitions. In practice, however, if the tool integration knowledge base may not be rich enough to support a certain integration problem, most likely a pragmatic approach aiming at a short term solution rather than investing in longer term benefits of reuse will be chosen. However, the prospect of Internet scale reuse by publishing tool integration knowledge bases and especially metamodel liftings, economy of scale will be an additional motivating factor. Nevertheless, the design of the semantic infrastructure will be such that lifting is optional or that it is possible to lift just core concepts of a metamodel.

References

- [1] V. Alexiev, et al. (eds.): Information Integration with Ontologies – Experiences from an Industrial Showcase, Wiley, 2005.
- [2] M. J. Anderson, B. D. Bird: An evaluation of PCTE as a portable tool platform, Proc. of the Software Engineering Environments Conference, July 1993.
- [3] S. Becker et al.: Model-Based A-Posteriori Integration of Engineering Tools for Incremental Development Processes, Journal on Software and Systems Modeling (SoSym), Springer, 4(2), May 2005.
- [4] J. A. Bergstra, P. Klint: The Discrete Time ToolBus - a software coordination architecture. Coordination Models and Language, LNCS# 1061, 1996.
- [5] J. Bézivin et al.: First Experiments with a ModelWeaver, OOPSLA & GPCE Workshop, Vancouver, Oct. 2004
- [6] A. W. Brown, P. H. Feiler, K. C. Wallnau: Past and future models of CASE integration, Proc. of the 5th Int. Workshop on Computer-Aided Software Engineering, IEEE, July 1992.
- [7] T.-P. Chang, R. Hull: Using Witness Generators to Support Bi-directional Update Between Object-Based Databases, ACM Symposium on Principles of Database Systems, May 1995
- [8] S. Clarke: Extending standard UML with model composition semantics, Science of Computer Programming, Elsevier Science, 44(1), July 2002.
- [9] K. Czarniecki, S. Helsen: Classification of Model Transformation Approaches, OOPSLA Workshop on Generative techniques in the context of MDA, Oct. 2003.
- [10] A. Doan, et al.: Introduction to the Special Issue on Semantic Integration, SIGMOD Record, 33(4), Dec. 2004
- [11] A. Earl: Principles of a Reference Model for Computer Aided Software Engineering Environments, Proc. of the Int. Workshop on Software engineering environments, Springer, USA, NY, 1989.
- [12] M. Ferdinand, et al.: Lifting XML Schema to OWL, 4th Int. Conf. on Web Engineering (ICWE), Munich, Germany, July, 2004.
- [13] R. G. Flatscher: Metamodeling in EIA/CDIF - meta-metamodel and metamodels, ACM Transactions on Modeling and Computer Simulation (TOMACS), 12(4), Oct. 2002.
- [14] A. Gangemi et al.: Sweetening wordnet with DOLCE, AI Magazine, 24(3), 2003.
- [15] J. Gray, T. Bapty, S. Neema, D. C. Schmidt, A. Gokhale, B. Natarajan: An Approach for Supporting Aspect-Oriented Domain Modeling, Generative Programming and Component Engineering (GPCE), Springer LNCS# 2830, Erfurt, Germany, Sept. 2003.
- [16] A. Y. Halevy, et al.: Enterprise Information Integration: Successes, Challenges and Controversies, Int. Conf. on Management of Data (SIGMOD), Baltimore, June 2005.
- [17] M. Haller, B. Pröll, W. Retschitzegger, A.M. Tjoa, R. Wagner, "Integrating Heterogeneous Tourism Information in TIScover - The MIRO-Web Approach", in Proc. of Information and Communication Technologies in Tourism (ENTER), Barcelona, Springer, April, 2000
- [18] D. Harel, B. Rumpe: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer, 64-72, October 2004.
- [19] M. Hitz, G. Kappel, E. Kapsammer, W. Retschitzegger: UML@Work", (in German), 3. Edition, dpunkt, July 2005.
- [20] J. Huang et al.: A Schema-Based Approach Combined with Inter-Ontology Reasoning to Construct Consensus Ontologies, 1st Int. Workshop on Contexts and Ontologies: Theory, Practice and Applications, July, 2005.
- [21] Y. Kalfoglou, M. Schorlemmer: Ontology Mapping: The State of the Art, Proc. of Dagstuhl Seminar on Semantic Interoperability and Integration 2005, Schloss Dagstuhl, Germany, 2005.
- [22] G. Kappel, E. Kapsammer, W. Retschitzegger: Integrating XML and Relational Database Systems, in WWW Journal, Kluwer Academic Publishers, June 2003.
- [23] M. Klein: Combining and relating ontologies: an analysis

- of problems and solutions, Workshop on Ontologies and Information Sharing (IJCAI), Seattle, USA, 2001.
- [24] J. Koehler, B. Srivastava: Web service composition: Current solutions and open problems, Proc. of the ICAPS, Workshop on Planning for Web Services, Trento, Italy, June 2003.
- [25] G. Kramler, E. Kapsammer, G. Kappel, W. Retschitzegger: Towards Using UML 2 for Modeling Web Service Collaboration Protocols", in Proc. of the Int. Conf. on Interoperability of Enterprise Software and Applications (INTEROP-ESA), Geneva, Switzerland, Feb. 2005.
- [26] D. L. McGuinness, et al.: An Environment for Merging and Testing Large Ontologies, 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR), USA, 2000.
- [27] S. Melnik, E. Rahm, P. A. Bernstein: Rondo: a programming platform for generic model management, ACM SIGMOD international conference on Management of data, ACM Press, New York, USA, June 2003.
- [28] I. Niles, A. Pease: Towards a standard upper ontology, 2nd International Conference on Formal Ontology in Information Systems, (FOIS), Maine, 2001.
- [29] N. Noy, M. A. Musen: The PROMPT suite: Interactive tools for ontology merging and mapping, International Journal of Human-Computer Studies, 59(6), 2003.
- [30] N. Noy: Semantic Integration: A Survey Of Ontology-Based Approaches, SIGMOD Record, 33(4), Dec. 2004.
- [31] P. A. Oberndorf: The Common Ada Programming Support Environment (APSE) Interface Set (CAIS), Software Engineering, IEEE Transactions, 14(6), June 1988.
- [32] A. D. Preece, et al.: KRAFT: an Agent Architecture for Knowledge Fusion, International Journal of Cooperative Information Systems, World Scientific Publishing, 2000.
- [33] QVT-Merge Group: Revised Submission for MOF 2.0; OMG Query/Views/Transformations RFP(ad/2002-04-10), Version 2.0, ad/2005-03-02, March 2005.
- [34] E. Rahm, P.A. Bernstein: A survey of approaches to automatic schema matching, VLDB Journal, 10(4), 2001
- [35] C. Reichmann, et al.: GeneralStore - a CASE-tool integration platform enabling model level coupling of heterogeneous designs for embedded electronic systems, Proc. of the 11th IEEE Int. Conf. on Engineering of Computer-Based Systems, May 2004.
- [36] T. Reiter, E. Kapsammer, W. Retschitzegger, W. Schwinger: Model Integration Through Mega Operations, Proc. of the Int. Workshop on Model-driven Web Engineering (MDWE), Sydney, Australia, July 2005
- [37] A. Schürr, H. Dörr: Introduction to the special SoSym section on model-based tool integration, Journal on Software and Systems Modeling (SoSym), Springer, 4(2), May, 2005.
- [38] M. Schrefl, M. Bernauer, E. Kapsammer, B. Pröll, W. Retschitzegger, Th. Thalhammer: Self-Maintaining Web Pages, in the Int. Journal of Information Systems (IS), Vol. 28/8, Elsevier Science Ltd., 2003.
- [39] A. P. Shet, J. A. Larson: Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases, ACM Computing Surveys, 22(3), Sep. 1990.
- [40] G. Straw et al.: Model Composition Directives, Proc. of the 7th UML Conference, Lisbon, Portugal, Oct. 2004.
- [41] L. Tratt: Model transformations and tool integration, Journal on Software & Systems Modeling (SoSym), Springer, 4(2), 2005.
- [42] M. Uschold et al.: Ontologies and Semantics for Seamless Connectivity, SIGMOD Record, 33(4), Dec. 2004.
- [43] R. Volz, D. Oberle, S. Staab, R. Studer: OntoLIFT Prototype, IST Project 2001-33052 WonderWeb, Deliverable 11, 2003.
- [44] A.I. Wasserman: Tool integration in software engineering environments, Proc. of the Int. Workshop on Software engineering environments, Springer, New York, USA, 1989.
- [45] G. Wiederhold, P. Wegner, S. Ceri: Toward Megaprogramming, Communications of the ACM, Nov. 1992.