On Models and Ontologies -
A Layered Approach for Model-based Tool Integration

Gerti Kappel[1], Elisabeth Kapsammer[2], Horst Kargl[1], Gerhard Kramler[1],
Thomas Reiter[2], Werner Retschitzegger[2], Wieland Schwinger[3] and Manuel Wimmer[1]

[1] Business Informatics Group, Vienna University of Technology
{gerti, kargl, kramler, wimmer}@big.tuwien.ac.at
[2] Information Systems Group, Johannes Kepler University Linz
{ek, tr, wr}@ifs.uni-linz.ac.at
[3] Dept. of Telecooperation, Johannes Kepler University Linz
wieland.schwinger@jku.ac.at

Abstract. The exchange of models among different modeling tools ever more 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. This paper proposes *ModelCVS,* a system which enables tool integration through transparent transformation of models between different tools' modeling languages expressed as MOF-based metamodels. ModelCVS provides versioning capabilities exploiting the rich syntax and semantics of models. Concurrent development is enabled by storing and versioning software artifacts that clients can access by a check-in/check-out mechanism, similar to a traditional CVS server. Semantic technologies in terms of ontologies are used together with a knowledge base to store machine-readable, tool integration relevant information, thus allowing to minimize repetitive effort and partly automate the integration process.

1 Introduction

The shift from code-centric to model-centric software development places models as first-class entities in software development processes. 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. To illustrate the specific challenges we want to tackle in this paper, we consider a real-world scenario encountered in a project, which involves a partner of Computer Associates (CA) and the Austrian Ministry of Defense. 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), the UML tool Rational Software Modeler, and the Oracle BPEL Process Manager. Covering a wide range of modeling tasks, Gen is a CASE tool supporting a proprietary modeling language, with which many existing applications have been developed. UML should be employed for new projects to link up with current technologies, and finally BPEL is required for developing certain web-enabled workflow applications. Without proper infrastructure support, integration of these tools poses severe problems as discussed in the following.

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 the 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 [12] and BPEL are somewhat different, although they express the same concepts. Furthermore, the *metamodels* 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 transformation is used, e.g., the forthcoming *QVT (Query/ Views/Transformations)-standard* [22], it is clear that implementing a transformation for Gen and UML will require a lot of effort. Hence, the problem is not only to implement a single translation, but to deal also with scalability problems. 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 transformation architectures arises.

Considering these challenges and based on experiences gained in various integration scenarios [11], [15], [19], [24], [26], we are currently realizing *ModelCVS,* a system which enables tool integration through transparent transformation of models between different tools' modeling languages expressed as MOF-based metamodels. In addition, ModelCVS will support versioning capabilities exploiting the rich syntax and semantics of models. It enables concurrent development by storing and versioning software artifacts that clients can access by a check-in/check-out mechanism, similar to a traditional CVS server. Semantic technologies in terms of ontologies are used together with a knowledge base to store machine-readable, integration relevant information, thus allowing to minimize repetitive effort and partly automate the integration process.

The remainder of the paper is structured as follows. Section 2 lays out the core concepts of ModelCVS while Section 3 proposes the system's architecture together with a simple

example demonstrating a prototypical implementation. An overview of related work is given in Section 4, followed by concluding remarks in Section 5.

2 Layered Approach to Tool Integration

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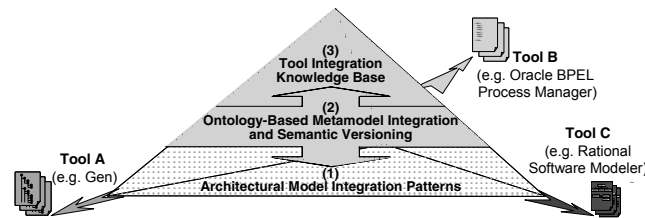

Fig. 1. Layered Tool Integration Approach

The bottom layer (1) is formed by *architectural model integration patterns* that ensure openness, scalability, and evolvability of our solution. Further elaborated on in subsection 2.1, these will serve as a basis to define specific bridging tasks and to develop appropriate *bridging operators*. 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 semantic versioning capabilities. The topic of semantic versioning, however, will not be further expanded in this paper (cf. [16]). Subsection 2.2 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*, which enhances support for metamodel bridging (cf. subsection 2.3).

2.1 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 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 [24]. By finally deriving *model transformation code* to enforce specific bridging semantics on models, the bridging language is made executable. For reasons of brevity we resort to only elaborating on two proposed integration patterns, namely *translation* and *modularization*, dealing with openness and scalability issues. Other patterns relevant for model-based tool integration include the *alignment* of models, which allows to keep models of conceptually disparate metamodels synchronized, as

well as *metamodel versioning* aiming at supporting the evolution of metamodels. For a description of these patterns we refer to [16].

Metamodel translation. The basic case of tool integration occurs when two different tools' modeling languages 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 so that models can be *translated* accordingly. As an example, we refer to the joint modeling of 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 translated 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. A further variation, which is advisable in case multiple tools with similar domains 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, allowing to reduce the amount of mappings required when translating between *n-many* tools from $O(n^2)$ to $O(n)$. 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}).
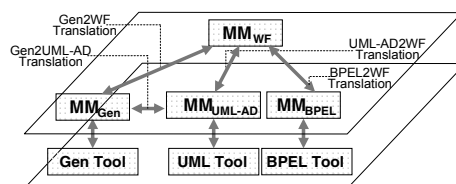

Fig. 2. Metamodel Translation

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, lets 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

14

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 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
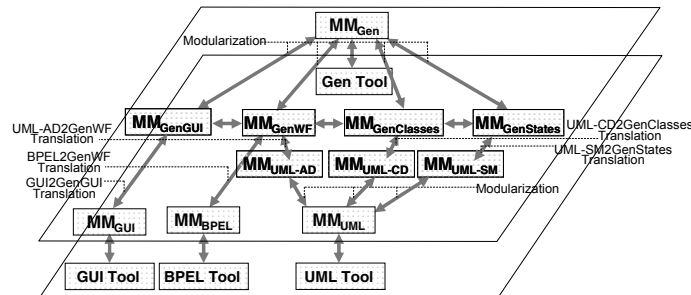

Fig. 3. Metamodel Modularization

The example depicted in Figure 3 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 fragments representing more specific domains (MM_{GenGUI}, MM_{GenWF}, $MM_{GenClasses}$, and $MM_{GenStates}$). As shown, the 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 in [16]. The bottom left part of the figure shows the integration of domain specific GUI and BPEL modeling tools, whose metamodels 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. At check-out time, models conforming to fragments have to be reassembled. This implies that links between model elements that have been cut off during modularization have to be re-established. The rules specifying how the models should be reassembled have to be derived from the applied bridging operators. To enable reassembly, information about linked model elements must be explicitly stored.

2.2 ModelCVS Semantic Infrastructure

In the following, the core functionalities of ModelCVS are laid out, which 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.

As a strictly manual bridging specification can become an 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. For the sake of simplicity, in the following our running example focuses on the metamodels of BPEL and UML Activity Diagrams to be integrated, only. Details on Figure 4, which generally depicts our setup used for ontology-based metamodel integration, are given throughout the next subsections.

(1) Metamodel lifting. The creation of an ontology from some kind of metadata like an XML schema [8] or a DB schema [30] 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 report [16]. 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.

(2) Ontology-level integration. The use of ontologies is based on the assumption that integration on the ontology layer is more easy to understand and can be automated to a greater extent. Lifting different metamodels' elements to concepts of some common ontology provides the first step of integration by establishing a common terminology. Thereby, it is necessary that the chosen generic ontology covers the domains of both tool ontologies appropriately. Furthermore, based on defined relations between concepts in the ontology, relations between the concepts of specific tools can be deduced, e.g.,

equivalence, subsumption, or substitutability. Continuing our example, we assume a generic *Workflow* ontology as the common upper ontology. As an example, we can imagine to map all of BPEL's control flow constructs onto the semantically appropriate classes in the *Workflow* ontology. Analogously we proceed with mapping the UML-AD metamodel onto the Workflow ontology. From the two mappings between tool and Workflow ontologies we employ structural reasoning to deduce relationships between ontology classes representing the control flow constructs of BPEL and ontology classes representing the UML-AD metamodel elements.

(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, semantic correspondence can be expressed by certain metamodel bridging operators accordingly. In case of a translation, a bridging operator might denote the creation of a new target model element for every encountered source model element, 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 the ActivityDiagram and the BPEL metamodels cover a largely similar domain. Hence, a relationship on the ontology level between 'equivalent' classes would be derived into a bridging operator relating the metamodel elements that initially got lifted to the respective ontology classes.
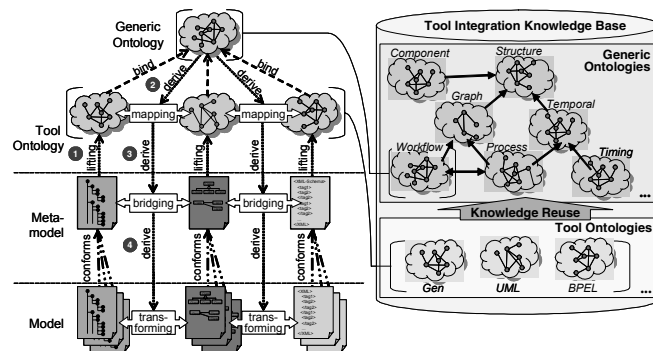

Fig. 4. Ontology-based Metamodel Integration

(4) Derivation of transformation. After bridging operators between metamodels are established, a code generation step results in QVT eventually representing executable transformations. In the context of a *translation* from BPEL to UML at execution time, this would basically result in code querying the source model and populating the target model with new elements appropriately.

17

2.3 Knowledge Base for Tool Integration

As described in the previous paragraphs, ModelCVS' semantic infrastructure makes use of ontologies for means of the integration of metamodels 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 specific tool ontologies, the resulting knowledge base will also comprise so-called *generic ontologies* in a hierarchical order providing reusable semantics (cf. Figure 4). For instance, the BPEL and the UML ActivityDiagram ontology can reuse concepts from the generic 'Workflow' ontology, which in turn can play a role in integrating these tool ontologies, as described in the BPEL to UML-AD example (cf. Section 2.2) earlier. Furthermore, the ontologies within the proposed tool integration knowledge base will be populated with *specific instance data* from reference examples of case studies. These 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.

3 Architecture and Prototype Implementation

As can be seen in Figure 5, 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 and lay out some of the design decisions taken during the realization of our prototype, whose functionality is detailed in a simple example.
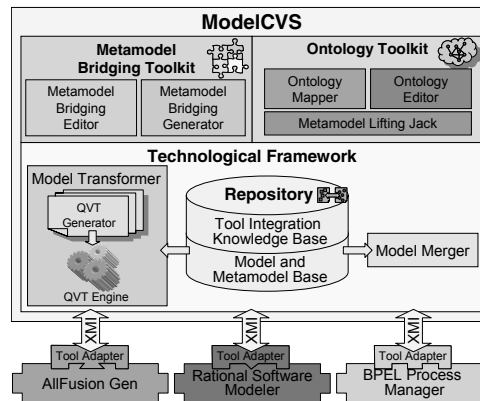

Fig. 5. ModelCVS Architecture

3.1 ModelCVS Architectural Components

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 *generic ontologies*, as well as associated mappings and liftings. Concerning the repository for the *model and metamodel base*, our prototype relies on the Eclipse Modeling Framework (www.eclipse.org./emf) and Subversion (subversion.tigris.org) for persistence and versioning capabilities, along an ontology repository for hosting *the tool integration knowledge base*. The ontology repository is the only component among those depicted in Figure 5 for which no prototypical solution exists, as for the moment an evaluation of viable solutions is still ongoing we simply store ontologies in the filesystem. The *Model Transformer* plugs into the repository and provides model transformation capabilities as required for the various tasks defined by the integration patterns. The prototype currently realizes model transformations with ATL [13] as a *QVT Engine* [22]. The metamodel bridges that are specified in a high-level language (cf. Section 2.1) using the *Metamodel Bridging Toolkit* have to be translated into that transformation language. A *QVT Generator,* prototypically realized through a template based approach, will perform this compilation task. The *Model Merger* also plugs into the *Repository* to provide merge conflict detection for models, based on the versioning capabilities provided by the repository back-end. *Tool adaptors* are a practical necessity,

since it cannot be assumed that all tools to be integrated in a tool chain support the *data format* of ModelCVS. XMI is a natural candidate for exchanging models, as it is based on MOF, and supported by many tools, particularly UML tools.

Metamodel Bridging Toolkit. This component provides all functionalities dealing with the handling of metamodels and especially the creation of metamodel bridges according to the various integration patterns. A *Bridging Editor* for the bridging language can for instance be implemented by reusing a generic mapping tool like the Atlas Model Weaver [4] that can be customized to accommodate the specific concepts of the bridging language, as was done in our prototype implementation. The *Bridging Generator* makes use of any mappings created at the ontology level to automatically derive bridges between metamodels. These automatically generated bridges usually 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*. Our prototypical lifting solution is able to map EMF's Ecore metamodels onto OWL ontologies, enabling the further process of semantic enrichment. To actually manipulate and make use of the resulting ontologies further, tools like Protégé (protege.stanford.edu), the JENA API (jena.sourceforge.net) as well as several specialized inference engines like F-OWL (fowl.sourceforge.net) can be used, contributing to the *Ontology Mapper* and the *Ontology Editor*. Our prototype is currently built on IBM's Integrated Ontology Development Toolkit (IODT)[1].

3.2 ModelCVS Integration Example

To exemplify the above described functionalities and demonstrate the feasibility of our prototype, Figure 6 shows two ontologies that have been lifted from a simple UML *ActivityDiagram* and a BPEL *Process* metamodel.

For reasons of simplicity we use a 'minimized' BPEL metamodel, in which we assume that all activities reside in one *Flow* activity with *Links* excplicitly defined between the contained *Activities*. Thus, our simple BPEL metamodel is made unambiguous, as "human-friendly" control structures are omitted without loosing semantics. Similarly, a simplified subset of the UML 1.4 metamodel is employed, as it is used in the UML profile for Automated Business Processes[2]. After the automated lifting step, subsumption relationships are being established to a generic upper ontology conceptualizing structural notions such as 'connection' and 'containment'. Note that only a few semantic enrichments relevant for the example are shown in Figure 6.

---

[1] http://www.alphaworks.ibm.com/tech/semanticstk
[2] http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/
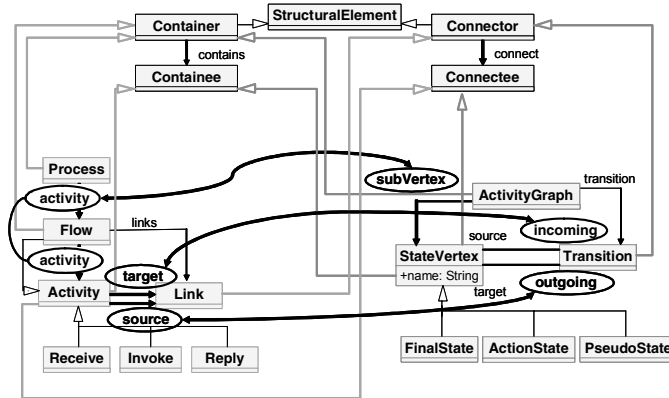
20

Fig. 6. Structural Reasoning for Mapping Tool Ontologies

Based upon the mapping towards the common upper ontology, which for instance assumes that an *ActivityGraph* is a *Container* as it is made up of *Transitions* and *StateVertices*, or that a *Reply, Receive* and *Invoke* are *Containees* due to the atomicitiy of primitive BPEL activities, structural reasoning can yield 'semantically equivalent' (or at least conceptually related) classes and properties. For instance, since *Process* and *Flow* subsume *Container* and *Activity* subsumes *Containee* (which means *Flow* is a *Containee* as well by inheritance), one can reason that the properties *activity* and *contains* linking the respective classes are equivalent, too. Analogously we proceed with the Activity Diagram ontology. Finally, we can, e.g., deduce that *subVertex* and *activity* are 'equivalent' properties. As another example, one sees that both Activity and *StateVertex* are *Connectees*, and *Link* and *Transition* are *Connectors*. Hence, one can see *target* and *source* are 'equivalent' to *incoming* and *outgoing* respectively. In this case structural reasoning on the ontology level was able to resolve semantic heterogeneity that a name matching heuristic would have not found, namely that the *source* and *target* properties contained in both ontologies do not(!) carry the same semantics. Momentarily, the ModelCVS prototype is able to carry out TBox reasoning on ontologies as described above to find equivalent classes and properties.

In a next step, depending on a certain integration pattern, ontological mapping information is used to derive bridgings between metamodels. Figure 7 shows a screenshot of our prototypical bridging editor, with a specific bridging model being edited, that consist of three different kinds of operators 'translating' the BPEL and the UML Activity Diagram metamodel: *ExclusiveEquivalence*, *SharedEquivalence,* and *GeneralizedEquivalence*. These operators should not be considered complete, but nevertheless allow to illustrate the use of bridging operators derived from ontology mappings for the purpose of this example.
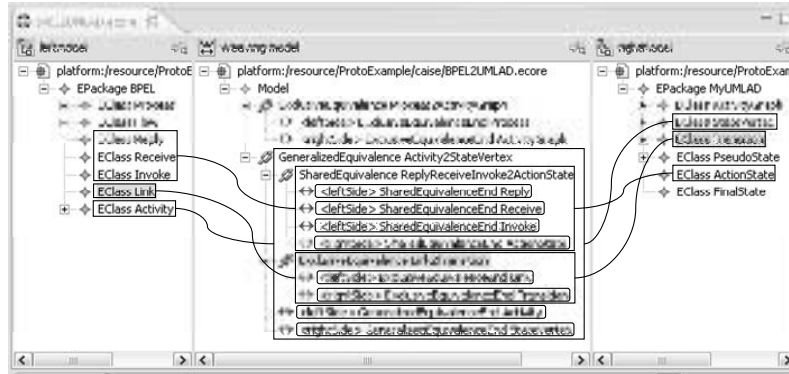
21

Fig. 7. Screenshot of Bridging Editor

In case an ontology mapping yields a one-to-one relation between classes, as it is the case with *Link* and *Transition*, or with *Process* and *ActivityGraph*, one can deduct that each class will have one counterpart, and no semantics is being lost when translating. However, in case of *Reply*, *Receive,* and *Invoke* mapping to *ActionState*, we identify a *SharedEquivalence*. This means, that multiple classes are being mapped on a single class, only, which would result in loss of semantics. Thus, additional information is introduced to the target model element, for instance in the form of an identifying attribute value or by a stereotyped class. A *GeneralizedEquivalence* indicates that their subclasses are involved in other, more specialized bridgings.

The described example shows that a mapping between ontologies yields a conceptual mapping, which has to be further refined by bridging operators on the metamodel level, allowing to make design decisions (*SharedEquivalence* using stereotypes, attribute, etc.) about the implementation of the translation. Utilizing a bridging model, our prototype then makes use of a template mechanism to create ATL code that finally implements the mapping. Although code generation is not complete, as for instance certain queries or helper functions require manual refinement, overall, a substantial amount of work can be avoided compared to traditional transformation development.

4 Related Work

This section gives an overview of related work that we deem relevant to ours. These fields encompass work on tool integration, which is helpful to put our goals in context with past efforts, model transformation languages building our system's backbone, and work on integrating heterogeneous data in terms of models and ontologies.

Tool Integration. Brown [5] categorized tool integration into a conceptual ("what is integration?") and a mechanical level ("how to provide integration?"). Regarding the conceptual level, Wasserman [31] first 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* and *middleware efforts* like, CDIF [9] and OMG's recent RFP OTIF[3] (open tool integration framework) and (2) *infrastructures* like the ToolBus architecture [3]. Some of these efforts were often grounded in large initiatives but have not been widely accepted, such as 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* [7]. *Tratt et al.* [29], e.g., provide an extensible, imperative model transformation language with some rule-based elements for pattern matching purposes. With ATL [13] *Bezivin et al.* have developed a hybrid (declarative/imperative) transformation language in response to QVT built upon EMF, making it especially applicable in context of Eclipse development. Such is the Eclipse based MTF[4] by IBM, which with a purely declarative transformation definition style might be harder to practically apply than ATL, for instance. BOTL[5] 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 [25]. For example, *WOTIF (Web-based open tool integration framework)*[6] 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 contrast to our approach. Finally, although *MDDi* (Model-driven Development Integration Project of Eclipse)[7] *is* still in its drafting phase, it provides some interesting ideas for model integration in terms of a bus architecture similar to AMMA[8]. 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* [27] and *web service composition* [18]. Such integration patterns (cf. Section 3) would require a series of basic model transformations which will not scale up when manually specified for complex models.

Integration patterns and bridging operators. There are only few related approaches (cf. e.g., [23]) providing abstraction mechanisms in terms of, e.g., high-level bridging

---

[3] http://www.omg.org/docs/mic/04-08-01.pdf
[4] http://www.alphaworks.ibm.com/tech/mtf
[5] http://www4.in.tum.de/~marschal/botl/
[6] http://escher.isis.vanderbilt.edu/tools/get_tool?WOTIF
[7] http://www.eclipse.org/proposals/eclipse-mddi/
[8] http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/

operators or modularization techniques in the areas of *model management and model integration*. For instance Rondo [20] provides high-level operations facilitating the integration of relational and XML schemata. Another interesting approach from the database community is MDM and its successor ModelGen [2], which to the best of our knowledge caters for the translation integration pattern, only. However, we believe that another M3 level other than MOF and the 'supermodel' approach may work well for integrating database models, we also believe that for general modeling languages their approach may not scale adequately. Nevertheless, the ideas behind the 'axiomatic' approach in which model transformations are derived from set of predefined rules may prove valuable to us with respect to the QVT generation.

In the modeling realm, Clarke [6] and Straw et al. [28] 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 2). 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. [10] *which* is a so called cross-cutting-concern weaver, is of interest. However, it lacks support for abstract integration mechanisms and is based on a meta-metamodel different from MOF, making the approach not immediately applicable for us.

Ontology-based Integration. Although concepts from related work in the area of *lifting metadata to ontologies* are of relevance to our approach, tools like the OntoLIFT prototype [30] for database schemata or the automatic mechanism introduced by Ferdinand et al. to lift XML schemata, 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 [17], which are similar to heterogeneities in database research [27]. 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* [21]. 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], [14], and [21].

5 Concluding Remarks

Currently, an early prototype of the proposed system exists, that already allows to carry out a comprehensive range of intended functionality. Besides further developing the existing implementation, our focus lies on extending bridging languages 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 concerning technological feasibility and practical applicability 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 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-metamodel 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* [11] 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 [11]. 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 metamodel lifting, 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 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 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.

References

[1] V. Alexiev et al. (eds.): Information Integration with Ontologies – Experiences from an Industrial Showcase, Wiley, 2005.

[2] P. Atzeni, P. Cappellari, P. Bernstein: A multilevel dictionary for model management, Int. Conf. on Conceptual Modeling (ER), Klagenfurt, Nov. 2005.

[3] J. A. Bergstra, P. Klint: The Discrete Time ToolBus - a software coordination architecture. Coordination Languages and Models, Springer LNCS 1061, 1996.

[4] J. Bézivin et al.: First Experiments with a ModelWeaver, OOPSLA & GPCE Workshop, 2004.

[5] W. Brown, P. H. Feiler, K. C. Wallnau: Past and future models of CASE integration, 5th Int. Workshop on Computer-Aided Software Engineering, IEEE, July 1992.

[6] S. Clarke: Extending standard UML with model composition semantics, Science of Computer Programming, Elsevier Science, 44(1), July 2002.

[7] K. Czarnecki, S. Helsen: Classification of Model Transformation Approaches, OOPSLA Workshop on Generative Techniques in the Context of MDA, Oct. 2003.

[8] M. Ferdinand et al.: Lifting XML Schema to OWL, 4th Int. Conf. on Web Engineering (ICWE), Munich, July 2004.

[9] R. G. Flatscher: Metamodeling in EIA/CDIF - meta-metamodel and metamodels, ACM Transactions on Modeling and Computer Simulation (TOMACS), 12(4), Oct. 2002.

[10] J. Gray et al.: An Approach for Supporting Aspect-Oriented Domain Modeling, Int. Conf. on Generative Programming and Component Engineering, Springer LNCS 2830, 2003.

[11] M. Haller, B. Pröll, W. Retschitzegger, A M. Tjoa, R. Wagner, "Integrating Heterogeneous Tourism Information in TIScover - The MIRO-Web Approach", Conf. on Information and Communication Technologies in Tourism (ENTER), Barcelona, 2000.

[12] M. Hitz, G. Kappel, E. Kapsammer, W. Retschitzegger: UML@Work (3.ed., in German), dpunkt, July 2005.

[13] F. Jouault, I. Kurtev: Transforming Models with ATL, Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica, 2005.

[14] Y. Kalfoglou, M. Schorlemmer: Ontology Mapping - The State of the Art, Dagstuhl Seminar on Semantic Interoperability and Integration, 2005.

[15] G. Kappel, E. Kapsammer, W. Retschitzegger: Integrating XML and Relational Database Systems, WWW Journal, 7(4), Kluwer Academic Publishers, Dec. 2004.

[16] G. Kappel, G. Kramler, E. Kapsammer, T. Reiter, W. Retschitzegger, W. Schwinger: ModelCVS - A Semantic Infrastructure for Model-based Tool Integration, Technical Report, ftp://ftp.ifs.uni-linz.ac.at/pub/publications/2005/0705.pdf, 2005.

[17] M. Klein: Combining and relating ontologies: an analysis of problems and solutions, Workshop on Ontologies and Information Sharing (IJCAI), Seattle, 2001.

[18] J. Koehler, B. Srivastava: Web service composition: Current solutions and open problems, ICAPS Workshop on Planning for Web Services, Italy, June 2003.

[19] G. Kramler, E. Kapsammer, G. Kappel, W. Retschitzegger: Towards Using UML 2 for Modeling Web Service Collaboration Protocols, Int. Conf. on Interoperability of Enterprise Software and Applications, Geneva, Feb. 2005.

[20] S. Melnik, E. Rahm, P. A. Bernstein: Rondo - a programming platform for generic model management, ACM SIGMOD Int. Conf. on Management of Data, New York, June 2003.

[21] N. Noy: Semantic Integration - A Survey Of Ontology-Based Approaches, SIGMOD Record, 33(4), Dec. 2004.

[22] 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.

[23] E. Rahm, P.A. Bernstein: A survey of approaches to automatic schema matching, VLDB Journal, 10(4), 2001.

[24] T. Reiter, E. Kapsammer, W. Retschitzegger, W. Schwinger: Model Integration Through Mega Operations, Proc. of the Int. Ws. on Model-Driven Web Engineering (MDWE), Sydney, Australia, July 2005.

[25] A. Schürr, H. Dörr: Introduction to the special SoSym section on model-based tool integration, Journal on Software and Systems Modeling, Springer, 4(2), May 2005.

[26] M. Schrefl, M. Bernauer, E. Kapsammer, B. Pröll, W. Retschitzegger, Th. Thalhammer: Self-Maintaining Web Pages, Information Systems, 28(8), Elsevier, 2003.

[27] A. Shet, J. A. Larson: Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases, ACM Computing Surveys, 22(3), Sep. 1990.

[28] G. Straw et al.: Model Composition Directives, 7th UML Conference, Lisbon, 2004.

[29] L. Tratt: Model transformations and tool integration, Journal on Software & Systems Modeling (SoSym), Springer, 4(2), 2005.

[30] R. Volz et al.: OntoLIFT Prototype, IST Project 2001-33052 WonderWeb, 2003.

[31] A.I. Wasserman: Tool integration in software engineering environments, Int. Workshop on Software Engineering Environments, Springer, New York, 1989.