ModelCVS
A Semantic Infrastructure for Model-based Tool Integration

G. Kappel, G. Kramler
Business Informatics Group, Vienna University of Technology

E. Kapsammer, T. Reiter, W. Retschitzegger
Dept. of Information Systems, Johannes Kepler University Linz

W. Schwinger
Dept. of Telecooperation, Johannes Kepler University Linz

Abstract
With the rise of model-driven software development, more and more development tasks are being performed on models. A rich variety of modeling tools is available supporting different tasks, such as model creation, model simulation, model checking, and code generation. 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 – unless we find some scalable way of integration.
We are aiming at providing a semantic infrastructure for model-based tool integration, enabling to facilitate any tool appropriate for the modeling task at hand. The key innovations provided are a set of scalable architectural model integration patterns supported by a high-level metamodel integration language, thus going beyond existing low-level model transformation approaches. Ontology-based metamodel integration considerably lowers the manual effort required for tool integration, enabling a novel synergic use of technologies from the model engineering and ontology engineering domains. An open knowledge base for tool integration captures essential knowledge about modeling languages and tools in terms of ontologies, fostering reuse within and beyond the scope of this project.
These innovations will be realized within the ModelCVS prototype and case study. The core of the system will be based on a versioning system such as CVS, thus providing a loosely-coupled and well-proofed integration architecture. Transparent transformation of models between different tools' languages and exchange formats, as well as versioning capabilities exploiting the rich syntax and semantics of models represent the key functionalities of ModelCVS. In this way, ModelCVS will serve as both, a research vehicle and testbed for exploring applications of semantic technologies in model-based tool integration and a prototype for a succeeding industrial product.

1 Introduction

Motivation for model-based tool integration. Seamless exchange of models among different modeling tools becomes an important prerequisite for effective software development processes. With the rise of *model-driven software development*, more and more development tasks are being performed on models, 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. 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 we find some way of integrating the variety of existing modeling tools. What we are looking for is *model-based tool integration*, enabling to facilitate 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 post-

hoc integration of modeling tools is very expensive in terms of effort and scalability compared to its benefits, and therefore out of scope of this project.

Problems of model-based tool integration. Integration of modeling tools in terms of model exchange poses several difficult problems, resulting in high effort and costs.

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 and tools are being developed, these are not tailored for the integration task, and furthermore require specific skills. Similarly, tools from the *enterprise information/application integration (EII/EAI)* markets [26] are not appropriate for handling data as complex as models typically are.

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.

Semantic technologies for model-based tool integration. We believe that *semantic technologies* can improve model-based tool integration in multiple ways. Schema matching and ontology mapping solutions can be adapted to the modeling domain to tackle the heterogeneity problem. Research results from ontology mapping can serve as a basis for developing concepts and operators for specifying model transformations at a higher level of abstraction. Advanced model merging techniques can be developed based on semantically enriched descriptions of modeling languages. And finally, ontologies can be used to build a knowledge base capturing essential tool integration experience.

Our approach – ModelCVS. Therefore, we propose to build *ModelCVS*, a *semantic infrastructure* that serves as both a research vehicle and testbed for exploring applications of semantic technologies in model-based tool integration and a prototype for a succeeding industrial product. The core of the system will be based on a versioning system like the *concurrent versioning system (CVS)*[2], thus providing a loosely-coupled and well-proofed integration architecture. Transparent transformation of models between different tools' languages and exchange formats, as well as versioning capabilities exploiting the rich syntax and semantics of models enhance the system's core.

To keep the system evolvable, a scalable architecture for realizing tool integration is provided that minimizes the effort necessary for integrating new tools while maximizing reuse of integration knowledge. Integration is specified both at syntactic and semantic levels. The *syntactic level* deals with metamodels which define the structures and datatypes of models, whereas the semantic level uses ontologies which describe the semantics of modeling concepts. *Semantic level* integration and reuse of integration knowledge will be specifically supported through dedicated component of the semantic infrastructure that facilitates lifting metamodels to the semantic level, finding mappings between metamodels to be integrated, and finding semantic merge conflicts. Another dedicated component, the knowledge base, will comprise generic modeling concepts, selected important modeling domains, e.g., workflow, as well as reference examples.

[1] http://www.omg.org/docs/omg/03-06-01.pdf
[2] https://www.cvshome.org/

Example

Overview of the case study. To exemplify the complexity of model-based tool integration and to point out the specific challenges we want to tackle, we consider a real-world scenario[3] that deals with the integration of three tools, the CASE tool *AllFusion Gen* (*Gen* for short)[4], the UML tool Rational Software Modeler[5], and the Oracle BPEL Process Manager[6]. In this scenario, 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)[7] 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)[8] for UML and XML for BPEL. Gen is actually a well integrated suite of tools covering a wide range of tasks, following a common modeling paradigm. The integration capabilities of Gen, however, are not open for external tools. 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 quite 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 there are tools to cope with that. A big 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 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 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.

Large and complex metamodels. The *metamodels* especially of Gen and UML are *very large and complex*. For instance, Gen's metamodel comprises more that 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* [48], it is clear that implementing a transformation for Gen and UML will require a lot of effort and not least due to the overall complexity will be a very error-prone process. Part of the problem is here, that existing *implementation technologies* such as object-oriented programming languages but also emerging model transformation languages are *not optimized* for the problem at hand. 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*

[3] This scenario is envisioned as case study to evaluate the results of this project. The tools have been selected according to the requirements of the demonstrator of our project and to cover a broad range of integration issues.
[4] http://www3.ca.com/Solutions/Product.asp?ID=256
[5] http://www-306.ibm.com/software/awdtools/modeler/swmodeler/
[6] http://www.oracle.com/technology/bpel/
[7] http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
[8] http://www.omg.org/technology/documents/formal/xmi.htm

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 (which is, e.g., a major concern in the current UML 2.0 standard). Also in this case, reuse could be supported through higher-level integration knowledge, thus reducing the manual integration effort to validation, precision, and completion.

2 Research Goals

The main research goal that has to be achieved in the proposed project is the finding of novel methods and the development of new technologies for a model-based approach to tool integration. The innovation in our approach is clearly the employment of semantic technologies in the form of ontologies to facilitate tool integration. Apart from the basic requirement of providing interoperability between tools, our focus on model driven development rises new requirements, in the form of providing model integration capabilities supported by semantic technologies for scalability reasons, advanced versioning mechanisms for practical applicability, and the construction of a tool integration specific knowledge base for the effective reuse of concept semantics for modeling languages.

The unique character of the proposed project stems from being rooted in several traditionally disparate research fields such as ontology engineering, model driven development, versioning, and tool integration in general. Thus, the specific research goals associated with the proposed project as listed below, are expected to contribute to various research areas.

Motivated by this overall research focus, three specific research goals can be derived, which are followed in this project (cf. Figure 1-1).

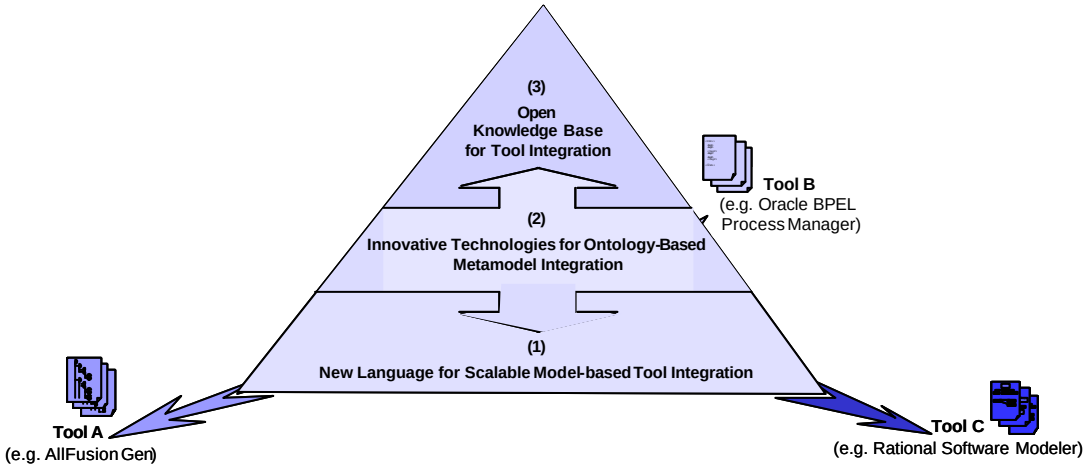

Figure 1-1: Research Goals of ModelCVS

(1) New Language for Scalable Model-Based Tool Integration

Metamodel bridging. Model-based tool integration comprises creating so-called *metamodel bridges* between the different tool metamodels (i.e., the metamodels of the modeling languages supported by the tools). These metamodel bridges define the model transformations facilitating transparent model translation. The main problems in creating such bridges arise due to metamodel heterogeneity in various aspects and due to the fact that existing implementation technologies are not exactly appropriate for the metamodel bridging task. While we do not attempt to fully *solve* metamodel heterogeneity in any case – for certain reasons heterogeneity is actually considered a necessity – we aim at providing improved technologies for *dealing* with metamodel heterogeneity in more efficient and evolvable ways.

Existing approaches. As already discussed, there exist specific languages for defining model transformations as required in the area of model-driven development. Requirements for such languages are, e.g., to transform complex high-level models into platform-specific models and ultimately into code. Although model transformation capabilities form the base of a model-based tool integration solution, generic model transformation languages operate at a very low level of abstraction, such that the specifics of tool integration are not explicitly supported (cf. Section 1.2). There also exist a wide range of EII/EAI tools, that support, e.g., conversions between different data formats (cf. Sections 5.2 and 5.4). These tools, however, do not focus, as already mentioned, on model-based tool integration and therefore are not able to deal with the complexities of metamodel bridging as envisioned in our approach.

Integration patterns and bridging operators. For these reasons, we aim at defining a language specifically tailored to metamodel bridging. We will identify *architectural model integration patterns* (*integration patterns* for short) that ensure openness, scalability, and evolvability of a tool integration solution. These will serve as basis to define specific bridging tasks and to develop appropriate *bridging operators* forming a metamodel bridging language that supports the identified integration patterns. Concerning these integration patterns and bridging operators, our research can build on a few closely related approaches in the areas of *model management* (e.g., [39]) *and model integration* (e.g., [6]) as well as in the area of *aspect-oriented modeling* (e.g., [54]) which can be used as a first starting point (for a detailed overview, cf. Section 1.2).

An initial set of integration patterns is proposed in Section 1.4, namely *translation* (i.e., bridging syntactic and semantic heterogeneity between largely overlapping tool metamodels), *alignment* (i.e., bridging cross-cutting concerns of partly overlapping tool metamodels), *modularization* (i.e., decomposing monolithic tool metamodels as a pre-requisite for scalable bridging), and *versioning* (i.e., semantic-based migration of different versions of tool metamodels).

(2) Innovative Technologies for Ontology-Based Metamodel Integration

Ontologies for metamodel integration. The proposed project makes extensive use of semantic technologies for the integration of tool metamodels as well as for the realization of semantically aware model versioning mechanisms. We assume that addressing the integration problem at the semantic level using ontologies improves the quality of automation support that can be achieved. Given the fact that a huge amount of work already exists in the area of ontology integration, the question arises as how these research results can be employed for *ontology-based metamodel integration*.

The essential difference between metamodels and ontologies is that metamodels define the concepts of a modeling language in terms of their syntax, whereas ontologies focus on the semantics of concepts, disregarding syntactical concerns. Therefore, in order to harness the potential of ontologies for metamodel integration and semantic versioning, the difference in abstraction level and semantic expressiveness between metamodels and ontologies needs to be dealt with.

Metamodel lifting. In this respect, we aim to enable transitioning from the mostly syntactic metamodel level to the semantic ontology level in terms of a translation and subsequent

syntax abstraction and *semantic enrichment*, furtheron called *metamodel lifting*. The lifting process should result in a mapping between metamodel level and ontology level such that both levels can be used synergically.

Regarding utilization of the expressiveness and reasoning capabilities of the semantic level, we aim in particular at supporting the various integration tasks as outlined in goal (1). Therefore, existing work on lifting data sources to ontologies (e.g., [59]), integration of ontologies (e.g., [41]) as well as modularization (e.g., [56]), and versioning of ontologies (e.g., [32]) has to be considered and adapted to the specific requirements of metamodel integration in terms of integration patterns and bridging operators.

(3) Open Knowledge Base for Tool Integration

Reuse capabilities. The basic idea behind the semantic infrastructure in ModelCVS is to enrich metamodels with specific semantics. As suggested above, this can be achieved by deriving *tool ontologies* from tool metamodels, which provide proper semantics for modeling languages. The entailment of specific semantics through an enrichment of tool ontologies, however, shall be possible with reasonable effort. Therefore, a key requirement is to provide *reuse capabilities* for the process of defining specific semantics for a tool ontology.

Tool integration knowledge base. Hence, our research aims at constructing a *tool integration knowledge base* that, similar to a library, provides reusable concepts for the enrichment of individual tool ontologies. The knowledge base should enable semantic support for ontology-based metamodel bridging as discussed in goal (2), as well as improved detection of versioning conflicts as motivated by the introductory example. The knowledge base should furthermore be open for usage outside the scope of ModelCVS.

Content of the knowledge base. Specific research tasks comprise, first, identification of *generic, reusable concepts* and development of a *structure to organize the contents* of the knowledge base and to enable efficient reuse. Second, devising a set of *reference examples*, which will be the result of our case study, to populate the tool integration knowledge base with, to be used to enhance ModelCVS' matching and reuse capabilities. Third, defining *knowledge about semantic merging conflicts* as required for enhanced model versioning capabilities, i.e., automated identification and subsequent resolution of such conflicts. Fourth, *establishment* of *a public platform* enabling Internet-wide access and contributions to the knowledge base as to maximize reuse effects.

3 State of the Art

In the following, the state of the art is described with respect to our research goals outlined in Section 1.1. For this, in a first step, a brief overview on tool integration in general is given, followed by a discussion of more closely related approaches in the area of model-based tool integration and concluded finally, by a review of ontology research for integration purposes.

3.1 State of the Art in Tool Integration

Research in tool integration has been a "hot" topic since the Stoneman Model[9] was proposed at the end of the 70's and summarized by Brown [8] in two categories, the conceptual level ("what is integration?") and the mechanical level ("how do we provide integration?").

Conceptual level of integration. In general, commercial of the shelf (COTS) tools are meant to be integrated if they function coherently and effectively in an environment as a whole, as is the case in an integrated development environment (IDE). Wasserman [60] 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*. Other categorizations used for characterizing tool integration

[9] http://www.adahome.com/History/Stoneman/stoneint.htm

comprises *depth of integration*, varying from exchanging byte streams to semantics-preserving integration, and the *universal applicability* of the integration approach.

Mechanical level of integration. The research efforts at the mechanical level of tool integration include (1) a series of *standardization efforts* and *middleware services* like CAIS [45], PCTE [2], CDIF [20], CORBA[10], and OMG's recent RFP OTIF[11] (open tool integration framework) to support tool interoperability, (2) *architecture models*, *infrastructures*, and *tool suites* like the ECMA toaster model [17], the ToolBus architecture [5], and finally (3) basic *tool integration mechanisms* such as data sharing, data linkage, data interchange, and message passing [52].

Some of these efforts were often grounded in large initiatives but have not been widely accepted. The European standardization effort PCTE, e.g., supporting data integration by providing tools with a common repository and services to store, retrieve, and manipulate data was not widely adopted in industry, not least because of its heavyweight architecture and high usage costs. Another example, CDIF, a standard for model exchange has been in the meanwhile replaced by MOF and XMI (cf. Section 1.2.2). Regarding, e.g., tool suites, they are often incomplete with respect to the various development activities requiring tool support, and most often do not allow to select between "best of class" tools (apart from promising exceptions like Eclipse[12]) [52].

Despite of all these important efforts, tool integration is still a challenging task, leading most often to hand-crafted bilateral integration solutions [52]. These "solutions" suffer from high maintenance overheads not least in case of evolutions of the underlying data or tools themselves, are often strongly technology-dependent and, most importantly, *do not scale*. With the advent of *model-driven development (MDD)* and in particular the introduction of *MDA,* new possibilities have been opened up to cope with these challenges.

3.2 State of the Art Relevant for Model-Based Tool Integration

Model-driven development. The key idea of MDA is to focus on models instead of code as the major artefact in software development. This allows modeling tools to be integrated on basis of the metamodels of modeling languages supported by the tools (i.e., the tool metamodels), thus paving the way for another generation of *(meta)model-based tool integration approaches* and providing a basis to overcome the above mentioned limitations of existing integration approaches. For this, MDA includes a set of interrelated standards[13], comprising a language for metamodel definition (*Meta Object Facility – MOF*), and the MOF-compliant languages for constraint specification (*Object Constraint Language – OCL*), model transformation (*QVT*), and metadata interchange (*XML Metadata Interchange – XMI*).

Model transformation as key technology

Model transformation languages. Model transformation is one of the major building blocks in the context of model-based tool integration and a very active research area. Existing approaches in this area having been either submitted to OMG's QVT request for proposals or being already part of existing MDA tools range from *algorithmic* and *imperative approaches*, via *graph-transformation-based approaches* to *template rule-driven,* and *hybrid approaches* [14]. *Tratt et al.* [57], e.g., provide an extensible, imperative model transformation language with some rule-based elements for pattern matching purposes, whereas *Becker et al.* [4] use purely rule-based mechanisms based on graph-transformations and generates wrapper for tool integration following a kind of programming-by-example approach. Highly relevant for our approach seems to be transformation languages such as BOTL[14], which allows the definition of modular, rule-based transformations, with independent rules for sets of metamodel elements – a property important for realizing the versioning interaction pattern as introduced in Section 1.1.

[10] http://www.omg.org/corba
[11] http://www.omg.org/docs/mic/04-08-01.pdf
[12] http://www.eclipse.org
[13] http://www.omg.org/technology/documents/modeling_spec_catalog.htm
[14] http://www4.in.tum.de/~marschal/botl/

Infrastructures based on transformation languages. 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 [52]). For example, *WOTIF (Web-based open tool integration framework)*[15] uses a graph-transformation mechanism and realises different tool integration patterns (e.g., direct tool integration and integration via a common metamodel), but requires that every tool to be integrated supports certain APIs for installing plugins which is in contrast to our approach. *GeneralStore* [50] 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, *MDDi, (Model-driven Development Integration Project of Eclipse*)[16], although providing some interesting ideas for model integration in terms of a bus architecture similar to AMMA (cf. below) is still in its draft proposal phase.

Deficiencies of pure model transformations. Although, QVT-like model transformation languages are a cornerstone also for our project, 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* [53], *megaprogramming* [61] and *web service composition* [35]. Such integration patterns (cf. Section 1.4) would require a series of basic model transformations which will simply not scale up when manually specified for complex models.

Beyond pure model transformation – integration patterns and bridging operators

There are only few closely related approaches providing abstraction mechanisms in terms of, e.g., high-level bridging operators or modularisation techniques in the areas of *model management and model integration* as well as in the area of *aspect-oriented modeling* which are described in the following in more detail.

Rondo. Having a similar intent in mind, the *generic model management initiative* from Bernstein et al. [39] provides a prototypical implementation called *Rondo*, which aims at keeping the matching of large XML schemata scalable. An approach to matching is introduced that operates on fragments of a large schema to lower the complexity of matching tasks. Besides this modularisation, model management operators on relational and XML schemata are provided, comprising, e.g., the automatic derivation of semantic correspondences or differences, the merging of models, and the derivation of a mapping from other mappings.

Although set in the context of relational and XML schema matching, this idea seems to be transferable to tool metamodels. Nevertheless our approach is not only aimed at finding semantic correspondences between metadata, but also to support certain model integration patterns, keeping a later code-generation step in mind in terms of deriving appropriate model transformation programs thereof. Another difference is that our focus goes beyond integrating XML and database schemata, by allowing the integration of arbitrary MOF-models in the sense of MDA.

AMMA / AMW. The *ATLAS Model Weaver (AMW)* which is part of the *AMMA model engineering platform* (soon to be released under the *Eclipse GMT* project[17]) proposed by Bézivin et al. [6], allows to perform a weaving operation in terms of establishing semantic correspondences between two metamodels, which are stored in a weaving model. Model weaving seems to be – different to Rondo – a manual operation, requiring an explicit specification of appropriate semantics for correspondences.

Our approach, in contrast, aims at both, inferring semantics of correspondences from ontological knowledge and providing a predefined set of higher-level bridging operators. In addition, ModelCVS extends the notion of weaving from an activity that solely establishes correspondences between metamodels, to a mechanism that interprets operators specified between metamodel elements and carries out transformation programs accordingly.

[15] http://escher.isis.vanderbilt.edu/tools/get_tool?WOTIF
[16] http://www.eclipse.org/proposals/eclipse-mddi/
[17] http://www.eclipse.org/gmt/

Aspect-orientation. The research efforts associated with *aspect-orientation* also deal with modularization in terms of factoring out cross cutting concerns into modules called *aspects*. This idea manifests in *aspect-oriented programming languages* [33], but also in *aspect-oriented modeling*, which allows to modularize cross-cutting-concerns in an implementation independent manner (cf. the approaches below).

Our approach focuses on tool integration, meaning that metamodels are, e.g., decomposed according to certain concerns they cover. *Weaving* as in aspect-orientation can be compared in our approach to the re-assemblage of models after modularization. In a tool integration setting, one can assume modularization to take place by detecting join points, e.g., in the form of meta-associations and point-cuts, e.g., in the form of links between model elements, to offer automatic support for a future re-assemblage. Most of the following approaches more or less use ideas from aspect-orientation for model integration purposes.

Model Composition Semantics. Clarke [12] introduces a *composition mechanism* for UML class diagrams, representing different separated concerns. Overlapping concepts are identified in these models and thus merged as specified by a composition relationship, following so-called *merge* and *override* strategies. Based on this basic integration behavior, *composition patterns* [11] are introduced as an extension to UML templates.

This approach focuses on UML models, only, and does not allow, e.g., the deletion of obsolete model elements after an integration is performed, as required for our approach. In addition, we focus on the derivation of model transformation programs during the integration stage, which are capable of automatically performing, e.g., the merging of models.

Model Composition Directives. Based on [12], Straw et al. [54] propose so called *composition directives* for composing UML class diagrams. These basically include name rewriting, adding, and deleting of model elements, change of references, and control of execution order. Inspired by aspect-oriented programming, so-called primary models are composed with aspect models, which represent a crosscutting concern to be interwoven.

Although composition directives are comparable to our envisioned model bridging operators, their primary focus seems to be on model weaving but not on meta-model weaving. We believe that our metamodel bridging operators could in turn be transformed into composition directives at the model level. Since we avoid an ad-hoc integration of models, with our approach, licit integrated models can be generated, only.

GME. The *Generic Modeling Environment (GME)* proposed by Karsai et al. [31] is a modeling and metamodeling toolkit based on UML notation and a GME specific meta metamodel. GME allows for the composition of metamodels similar to our approach. The composition mechanisms comprise an *equivalence operator* creating a union of two model elements, similar to the *merge* semantics in [12] and two different inheritance operators, realizing implementation inheritance and interface inheritance.

Differerent to our approach is that GME is not based on the MOF standard. Furthermore, our approach goes beyond the functionalities for metamodel composition in GME by supporting different model integration patterns, not just composition of metamodels.

C-SAW. C-SAW, developed as a plug-in for GME by Gray et al. [23] is a so called cross-cutting-concern weaver. Aspects are specified using the Embedded Constraint Language (ECL), a OCL superset, additionally providing imperative constructs for model manipulation.

The transformation capabilities of ECL are, however, limited to models of the same metamodel, it lacks support for abstract integration mechanisms and is, instead of MOF, based on a meta-metamodel specific to GME, making the approach not applicable for us.

Domain Composition Approach. Estublier et al. [18] propose a *UML profile* allowing the composition of separately designed domain models, as required when facing the federation of immutable components off the shelf. UML associations and association classes are specialized by stereotypes to express feature correspondence and concept overlapping.

In principle, this approach is similar to our envisioned alignment interaction pattern, but does not support other interaction patterns as targeted in our project. In addition, only UML models are supported instead of arbitrary MOF-models.

Summary. Summarizing, although there are already few approaches targeting model-based tool integration from a meta-modeling point of view and providing some basic abstraction mechanisms in terms of modularization techniques and bridging operators, each of them suffers from certain deficiencies with respect to the focus of our project as outlined above. Nevertheless, several ideas and concepts of these approaches could be of high value for ModelCVS, which has to be investigated in-depth in the course of the project. It has to be noted however, to the best of our knowledge, none of these approaches uses ontologies to facilitate the semantic aspect of model-based tool integration, as done in our approach.

3.3 State of the Art Relevant for Ontology-Based Metamodel Integration

History of semantic integration. The research field mainly relevant for ontology-based metamodel integration is the broad area of *semantic integration*. The history of semantic integration goes back to the early 1980s, where Brodie et al. [7] addressed semantic relativism in data modeling, leading to a comprehensive taxonomy of semantic heterogeneities introduced by Shet et al. [53] in the early 1990s and an in-depth survey of automatic schema matching approaches in 2001, published by Rahm et al. [49]. Although the problem of semantic integration is tackled in various ways by different communities, as could be seen at the remarkable Dagstuhl workshop on semantic interoperability and integration in 2004[18], in recent years, ontologies became very popular to facilitate various semantic integration tasks. This is not least since, in comparison to other techniques, integration based on ontologies can rely heavily on the high expressive power of ontology languages and on appropriate reasoning techniques.

As already stated in Section 1.1, our approach will utilize ontologies as a base mechanism to semantically enrich tool metamodels, thus facilitating tool metamodel integration. In this respect related work in the area of *lifting metadata to ontologies*, *issues of integrating ontologies*, and the usage of *integration patterns for ontologies* is highly relevant for our approach, as discussed in the following.

Lifting Metamodels to Ontologies

A basic question to be investigated is the *derivation of ontologies* from the tool metamodels, often referred to as *lifting*. Few existing work, although approaching the lifting problem from somewhat different angles, could be used as starting point to resolve this research question.

OntoLIFT. Lifting is, e.g., dealt with in the WonderWeb project in terms of the OntoLIFT prototype [59], which helps to semi-automatically create ontologies from database schemata by using syntactical patterns as employed for mapping database schemata to ER models. Although these ontologies have to be further refined to infer specific semantics, OntoLIFT provides a useful entry point for the establishment of ontologies.

Ferdinand et al. Another approach from Ferdinand at al. [19] proposes an automatic mechanism to lift XML Schema to the *Web Ontology Language (OWL)*[19] via RDF and provide according mapping rules.

Although both approaches deal with the derivation of ontologies from structured sources, methods applied in these two approaches cannot be immediately reused, since ModelCVS requires the derivation of ontologies from metamodels. Further research has to be put into the question of how to facilitate the creation of ontologies from MOF-based metamodels.

ODM. A way to bridge between model engineering and ontology engineering could be the *Ontology Definition Metamodel (ODM)*[20], an upcoming OMG standard for the definition of ontologies in terms of MOF models.

Guizzardi et al. [24] provide an evaluation framework to estimate the appropriateness and the comprehensibility of a modeling language for describing concepts in terms of domain knowledge captured in an ontology. Such considerations are relevant in the context of

[18] http://www.dagstuhl.de/04391/
[19] http://www.w3.org/TR/owl-features/
[20] http://www.omg.org/cgi-bin/doc?ad/2003-03-40

ModelCVS to define ontologies for modeling languages or to estimate to what extent existing ontologies can be reused.

Basics of Integrating Ontologies

As ModelCVS is able to perform tool metamodel integration on basis of semantics covered by tool ontologies, these individual tool ontologies have to be integrated. The central burden making ontology integration a rather comprehensive challenge are heterogeneity issues that have to be coped with [34], which are similar to heterogeneities in database research [53]. 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* [44]. 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 may provide useful input for our approach. For a comprehensive overview about this active research area compare, e.g., [1], [30], and [44].

Ontology integration architecture. Concerning the architecture for ontology integration, one can basically distinguish three alternatives (cf. e.g., [44]): (1) a *direct mapping* between ontologies, (2) an *indirect mapping* via a *common, shared ontology* further on called *upper ontology* (sometimes also referred to as *toplevel, or reference ontology*), e.g., the Standard Upper Merged Ontology (SUMO) [42] and DOLCE [21] and (3) a mapping based on a *library of already mapped ontologies* [58]. This is again similar to database integration research, where peer-to-peer database systems are similar to the direct mapping approach, and federated database systems relying on a global schema are similar to the indirect mapping approach with the difference that an upper ontology is usually more general since it needs to encompass the top level for ontologies yet to be developed [44].

We intend to use a *hybrid approach*, involving all three architectures in order to ensure a balance between reuse capabilities, provided by upper ontologies as well as ontology libraries, and overhead induced, which can be reduced by using direct mappings for special, non-recurring mappings. For this, existing approaches as mentioned above can provide a valuable input, although they have to be adapted in order to deal with our special focus of deriving appropriate metamodel bridges.

Mapping discovery. 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* deal with finding such *correspondences* (also called *matches*) 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* [49], [15], [44].

It has to be emphasized, that it is not the intent of this project to develop yet another mapping discovery technique. Rather, it is foreseen to either use a single existing technique or a combination thereof which can be easily adapted to best fit our requirements. A selection of some of these approaches which may be (partly) useful for our purposes are sketched out in the following.

Chimaera. Chimaera [38] provides support for *ontology merging by interactively relating concepts* that are identical or related by subsumption or instance relationships. Further, it supports to *manipulate the ontologies* as to improve alignment by suggesting modifications.

PROMPT. PROMPT [43] supports *interactive, guided ontology merging*, starting from *linguistic* and *structural similarity* matches. Merge operations can be performed, and based on the results and potential conflicts arising from the merge (e.g., name conflicts, dangling references, or redundancies in class hierarchies), further operations are proposed.

KRAFT. KRAFT [47] supports the *finding of mappings* by *special mediator agents* which can be customized with respect to support particular ontologies as well as ontology languages. Although the approach provides great flexibility in supporting various mappings, the user is able to specify arbitrary mappings since the semantic of concepts is not regarded, thus risking wrong and even conflicting mappings. Within our approach it is of major importance

to guide the user and prevent useless mappings exploiting the provided semantics.

PUZZLE. The goal of PUZZLE [29] is to construct a *consensus* ontology, i.e., a common, shared ontology, from independently designed ontologies. Both, *linguistic* as well as *contextual features* of ontology concepts are considered, there is no need for a previous agreement on the semantics of the used terminology and WordNet is used to support, e.g., synonyms and homonyms. Reasoning rules are based on the relationships *subclass*, *superclass*, *equivalentclass*, and *sibling*, and on property lists of ontology concepts to find new relationships among concepts.

Representation of mappings. Having found appropriate mappings, they have to be properly represented in order to facilitate reasoning on mappings. Concerning the representation of mappings several approaches can be found in literature [44]. Note that also combinations thereof are possible. First, similar to traditional data integration, *views* can be used to describe mappings, e.g., between upper ontology and local ontologies, either using the *global-as-view (GAV)* or the *local-as-view (LAV)* approach, well known from database integration research [25] and used, e.g., within the OIS framework [9]. Second, mappings can be represented in terms of *bridging axioms* in first-order logic to express transformation rules, relating classes and properties of two ontologies, as it is done in the OntoMerge system [16]. Finally, mappings can be represented as instances in an *ontology of mappings*. The mapping ontology usually provides different ways of linking concepts from the source ontology to the target ontology, transformation rules to specify how values should be changed, and conditions and effects of such rules. Examples are the *Semantic Bridge Ontology* of the MAFRA framework [37] or the *mapping ontology* [13].

Within our project we will have to investigate the proposed alternatives to find an appropriate one, whereby specific mapping ontologies seem to provide a great potential for mapping representation as well as reasoning.

Reasoning with mappings. In general, reasoning aims at drawing a conclusion, e.g. to perform semantic integration tasks. In ModelCVS reasoning over ontology mappings is required to facilitate metamodel integration. Reasoning hardly depends on the underlying representation form [44]. In the OIS framework [9] mappings are expressed on basis of a GAV/LAV approach, using description logics and therefore a special description logics reasoner. PROMPT [43] takes a mapping ontology and automatically merges the corresponding ontologies based on the specified mapping. In case that we utilize a mapping ontology, corresponding existing approaches will be taken as base and adapted for our special purposes.

Model Integration Patterns and Ontologies

As our approach provides different model integration patterns such as alignment and modularization to allow metamodel integration in a scalable way, also the tool ontologies have to support these model integration patterns.

Klein, e.g., [34], suggests several *kinds of integration*, applied to ontologies as a whole, which are comparable to our integration patterns. In the Onion system [41], an *algebra for ontology composition* is proposed, supporting several operators, e.g., filter, extract, union, intersection, and difference. There already exist approaches in the field of ontology modularization, e.g., [55], [56] and [22], aiming at modularizing ontologies for the purposes of efficient reasoning, distribution, and maintainability.

In our approach, however, not an ontology is the target of integration patterns, but the tool metamodel it is associated with and as a consequence to that, finding ways for dealing with ontologies in the same semantics preserving way. Therefore, the approaches described above are not immediately reusable in our context, but could provide a useful starting point.

To support *ontology versioning*, Kauppinen et al. [32] define a so-called *change bridge ontology* that enables reasoning about an evolved ontology. The goal is not interoperability, as with ontology mapping in general, but rather to align the revisions of a single ontology in time. Important in our context are also maintenance and evolution techniques for mappings, as proposed, e.g., by Maedche et al. [36], providing a reusable ontology of semantic bridges.

Our interest is, in addition to relating versions of metamodels and corresponding ontologies, also in performing model migration, a capability that is not readily supported by ontology mappings. In the area of databases, schema evolution has been addressed in terms of schema change operations that also define corresponding updates to instances. Banerjee et al. [3] defines a comprehensive taxonomy of schema change operations. Using such operations, however, presupposes that a trace of change operations is available, which is typically not the case when a new version of a tool metamodel becomes available.

Summary. As outlined in this section, there is already a huge amount of work in the area of semantic integration dealing with ontologies, providing a proper basis for ModelCVS. There is, however, to the best of our knowledge, no literature available, dealing with the usage of ontologies for metamodel integration, thus leaving open research questions in several directions. We are sure that a combination of existing techniques in the area of model management and integration with semantic technologies in terms of ontologies will allow to fully exploit the potential of model-based tool integration in a semantic-preserving way, thus representing an example for future commercial products in this area.

4 Proposed Technical Solution

The solution description provided in this section is split into two subsections. The first subsection focuses on conceptual solutions and deals with the resolution of the research goals for ModelCVS established in Section 1.1. The second subsection puts a focus on the realization of ModelCVS from a technological point of view, describing a component-oriented system architecture.

4.1 Approaches taken for the Resolution of Research Goals

(1) New Language for Scalable Model-Based Tool Integration

The basis for our solution to this goal is a set of integration patterns that define requirements and working context for the bridging language to be developed. We propose four initial integration patterns that address openness, scalability, and evolvability covering various situations relevant for model-based tool integration. 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 [10].

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 1-2 shows the translation approach involving the process metamodel of Gen (MM_{Gen}), UML's activity diagram metamodel ($MM_{UML\text{-}AD}$), and BPEL's metamodel (MM_{BPEL}). 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 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.*

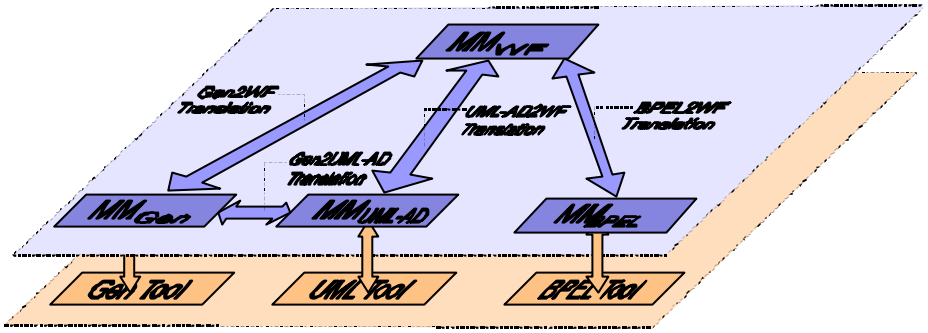

Figure 1-2: Translation using a pivot metamodel MM_{WF}

Metamodel alignment. The *alignment* pattern deals with interrelating rather than translating models. This requirement occurs when a system to be modeled cross-cuts 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 1-3 shows the alignment of two different tool metamodels (MM_{GUI} and $MM_{UML\text{-}CD}$), provided by a tool for modeling GUIs and a UML tool, which are used jointly to model a single system. 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.

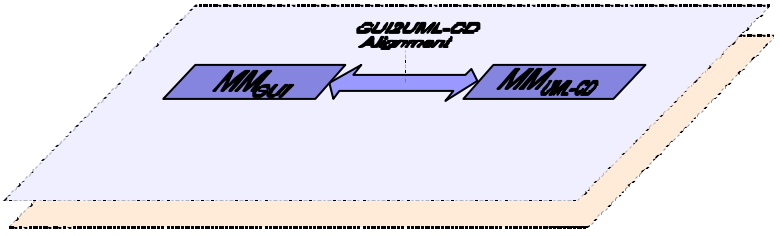

Figure 1-3: Alignment of two metamodels MM_{GUI} and $MM_{UML\text{-}CD}$

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 1-4 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 ($MM_{UML\text{-}AD}$, $MM_{UML\text{-}CD}$, and $MM_{UML\text{-}SM}$). 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.

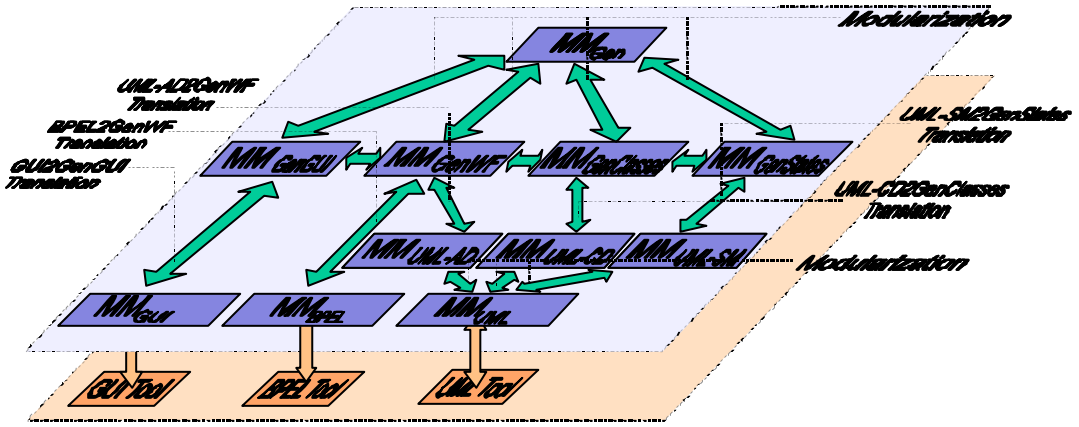

Figure 1-4: Modularization of Gen and UML metamodels facilitating scalable integration

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 1-5 illustrates the required migrations facilitating metamodel versioning when a new tool version *UML2* and a corresponding tool metamodel version for UML2 activity diagrams $MM_{UML2\text{-}AD}$ is introduced into an existing tool chain. What needs to be done is (1) to define a model migration bridge from the old version of the metamodel ($MM_{UML\text{-}AD}$) to the new version ($MM_{UML2\text{-}AD}$), 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.

Since the purpose of these patterns is to identify bridging tasks as specific as possible to enable definition of well-suited bridging operators, we aim to enrich these patterns and include additional ones in the course of the project.

Bridging operators. The bridging language containing bridging operators to be developed, has to specifically support the identified integration patterns at a suitable abstraction level and with appropriate range of functionality according to specific bridging tasks. Taking into account the peculiarities of specific tasks will enable us to develop a descriptive high-level language that can be more efficiently used than, e.g., generic model transformation languages. For elicitation of more specific requirements and identification of typical bridging problems, the integration patterns and exemplary tool metamodels from our case study will be investigated. The language must support evolvability in terms of bridge migration as required in the metamodel versioning pattern. Still, the language needs to be executable in that model transformation code can be generated out of it. With respect to modularization, for instance, transformation programs for the modularization of models, the alignment of overlaps in fragment models, as well as the automatic assembly of fragment models at check-out time will be derived. Definition and implementation of a mapping from the bridging language to an executable model transformation language will be the final task in resolving this research goal, serving as a proof of concept for the bridging language and as a component of the overall system architecture.

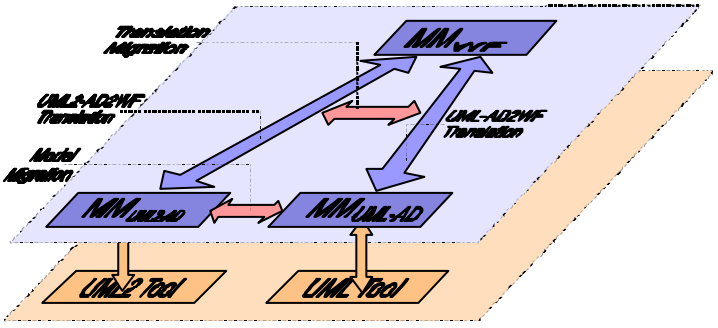

Figure 1-5: Versioning of UML metamodel and involved migrations

Achieving these goals can build on initial work already carried out by the project partners (cf., e.g., [51], [62] and Section 2) as well as on several branches of existing research in the areas of *model transformation languages*, *model management and integration*, *aspect-oriented modeling*, as well as *ontology modularization and versioning* (cf. Section 1.2).

(2) Innovative Technologies for Ontology-Based Metamodel Integration

Our approach to ontology-based metamodel integration basically comprises a technology for transitioning from the mostly syntactic metamodel level to the semantic level in terms of ontologies, i.e., metamodel *lifting*, and adaptations of existing research results in the area of ontology integration for the purposes of metamodel integration facilitated by the generation of bridges between metamodels as described above.

Metamodel lifting. The creation of an ontology from some kind of metadata like an XML schema [19] or a DB schema [59] 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 defines the semantics of the modeling concepts whose syntax is defined by the metamodel. A so-called *lifting mapping* keeps track of the relationship between syntax and semantics of modeling concepts.

Automatic as well as semi-automatic approaches to lifting have already been presented in literature (cf. Section 1.2). The challenge in all of these approaches is to define a kind of reverse engineering procedure which extracts the "pure" knowledge from the metadata, abstracting away application specific concerns and limitations of the implementation language. In case of lifting a relational schema, e.g., lifting equals reverse engineering from the relational schema to an ER model [59]. In that case, it can be performed semi-automatically since typically the well known patterns for mapping ER models to relations are used. Our case of metamodel lifting is more complex since, first, there are no patterns established for mapping certain language concepts to a metamodel, and second, we want to support any kind of MOF-based metamodel, thus there is a need for a generic solution.

To improve quality of automated mapping and semantic versioning, specific semantics expressed in terms of concepts defined in a so-called *tool integration knowledge base* should be reused. The relationship between lifted metamodel concepts and concepts in the tool integration knowledge base could be established through subsumption relationships, for instance. Since an ontology has to appropriately capture the semantics of the domain covered by the modeling language [24] which, however, is not appropriately captured by a tool metamodel, additional semantics have to be manually incorporated into the ontology.

We nevertheless envision a lifting process that proceeds semi-automatically, based on a number of options for automation support:

- A *lifting editor* can provide interactive guidance in the lifting process. Examples of interactive guidance for integration tasks can be found, e.g., in ontology integration tools such as [43]. An interactive lifting editor can be designed learning from such systems. A lifting guidance mechanism can exploit in particular several sources of knowledge, like information from the metamodel, e.g., generalizations and associations between metamodel concepts, or, can be based on already existing mappings from metamodel concepts to concepts in the tool integration knowledge base, such that related concepts and potential lifting mappings may be derived.
- A *mapping discovery* mechanism can be employed to automatically propose relationships between metamodel concepts and concepts from the tool integration knowledge base. Such a mapping can be based on heuristics and specific assumptions about the domain of the metamodel and corresponding domain specific ontologies can be made. Furthermore, to enable *instance-based matching*, it is necessary to use the tool to be integrated and model a reference example, which is specified as part of the tool integration knowledge base. Based on the gathered model, an automatic mapping of concepts from the tool metamodel to the knowledge base can be established.
- *Syntax patterns* can be used to help defining the mapping between semantic concepts and their syntactical representation in the metamodel, similar to the way patterns for

mapping ER models to relations are employed in [59]. As an example, lets consider two syntax patterns for representing a *directed association* (according to UML terminology) in a metamodel. In UML, this involves a model element of metaclass *Association*, two model elements of metaclass *Property*, and corresponding links between them. In contrast, the representation of a *reference* in MOF, which is semantically equivalent to the UML directed association, syntactically just consists of a link between two classes. Syntax patterns can be used for predefining mapping operations for lifting, as well as for interactive guidance and improved matching. Since currently no library of such patterns exists, we aim at creating one by identifying syntactical structures recurring in the metamodels of our case study.

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. Furthermore, with the prospect of Internet scale reuse by publishing 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.

Ontology integration architecture and mapping discovery. When it comes to establishing a bridge between two tool metamodels, ModelCVS offers support in the form of semantic technologies by performing a mapping of the respective tool ontologies. The resulting mapping between ontologies is used to derive a bridge between the underlying metamodels, as a strictly manual bridging specification can become an error prone and time consuming task.

ModelCVS supports a *hybrid integration architecture*. Basically mapping two tool ontologies is facilitated indirectly by a common *upper tool ontology* (cf. research goal (3)), which describes the domain covered by the two modeling languages, or directly by mapping them in a point to point manner, or using already *existing mappings* to deduce new ones. *Mapping discovery* can be supported by *heuristics* finding correspondences based on criteria such as name matching and structural equivalence or simply done manually. Mapping mechanisms can also operate directly on the metamodel level in case not all metamodel concepts have been lifted to the ontology level.

Although semantic technologies 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. To find a mapping between metamodels representing for instance different versions of a modeling language, heuristics based on structural and naming similarity may work well to establish a direct mapping, whereas the mapping of metamodels exposing very different structure and naming conventions may require the use of an upper ontology-based approach, combined with a manual mapping, for instance. Especially the mapping on basis of upper tool ontologies and heuristics, although providing several benefits, is a challenging task in the context of ModelCVS, because of several reasons.

Upper tool ontology. The establishment of mappings between tool metamodels by means of an *upper tool ontology* requires the tool metamodels to be semantically enriched, meaning a lifting onto tool ontologies exists. From each of these tool ontologies mappings are made onto an *upper tool ontology*. The explicit mapping to an upper tool ontology gives a user concise control in terms of which ontology concepts are related to each other. Utilizing this upper tool ontology, semantic correspondences between concepts in both tool ontologies can be deduced, of which subsequently a bridge between metamodels can be derived.

Figure 1-6 illustrates this approach by showing the lifting of metamodels to the ontology level which furthermore is associated with an upper ontology level. From the upper ontology, mappings between the ontologies are deduced which are used to derive bridges between

metamodels that in turn define a series of model transformation operations to ultimately carry out the necessary transformation operations on models to realize, e.g., translation.

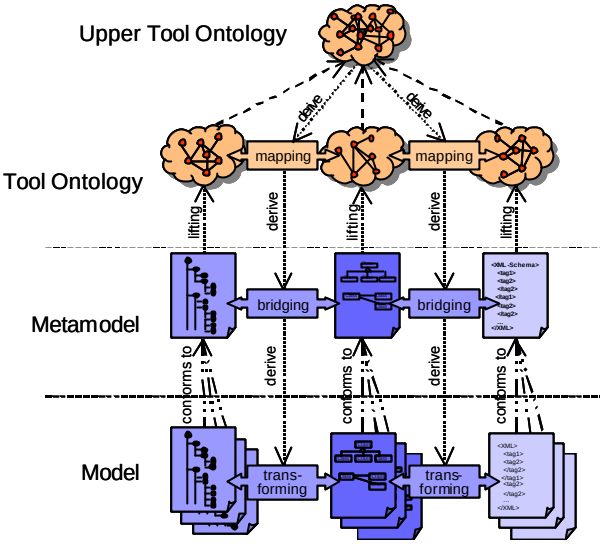

Figure 1-6: Using ontologies and upper ontologies for model integration

Heuristic mapping. 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., [21]) allows to identify and relate names in question as for instance being synonyms, homonyms, antonyms and the like. Furthermore, the results of heuristic matching techniques can be greatly enhanced when incorporating instance data into the ontology matching process [29] (cf. instance-based matching), which could be accomplished by populating both tool ontologies with data of a common reference example.

Once semantic correspondences between tool ontologies are established, a bridging between the underlying metamodels can be derived. As an example, we assume that a matching on the ontology level results in the finding of two semantically equivalent classes. 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 to be investigated. 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 metamodel elements should be merged into one at check-out.

(3) Open Knowledge Base for Tool Integration

Our approach to realizing a tool integration knowledge base comprises development of a range of ontologies capturing the semantics of modeling languages from typical domains. To foster efficient knowledge reuse among ontologies within the field of tool integration, a hierarchical structure comprising *tool ontologies*, *upper tool ontologies*, and *generic modeling ontologies* will be imposed. Furthermore, we will include instance data forming a reference example developed in the course of our case study, and define language semantics as relevant to enable semantic model merging. The resulting knowledge base will be published on an Internet platform, enabling Internet-wide reuse.

Tool ontologies. Bottom-up development of the knowledge base starts with modeling languages and corresponding metamodels from various domains. In particular, Gen, UML, and BPEL will be considered. According to the lifting mechanism described above, *tool ontologies* will be constructed for selected parts of these modeling languages. 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.

Tool independent ontologies. Tool independent ontologies exist independent of certain tool metamodels. In particular, an *upper tool ontology* facilitates the integration of several tools pertaining to a common modeling domain. For example, the tool ontologies of a BPEL tool, a UML activity diagram tool, and the Gen process modeling tool (cf. Translation in Section 1.4.1) all belong to a common modeling domain, which could be described by a tool independent *Workflow* ontology. An upper tool ontology can be constructed by generalizing concepts from various tool ontologies covering the same or similar domains, as well as by directly taking into account domain conceptualizations. Furthermore, generic modeling ontologies provide a reuse base of generic concepts for other ontologies. Generic modeling ontologies can be constructed by abstracting concepts common to different domains such as generalization and by reuse of existing foundational ontologies, e.g., UFO-A [24].

The resulting knowledge base will comprise a range of foundational and generic, as well as domain-specific ontologies, to facilitate reuse in specifying semantics for tool ontologies during lifting. Additional knowledge useful for our purposes, e.g., a lexical reference such as WordNet [21], will be kept external to the tool integration knowledge base. Figure 1-7 illustrates the relationships between tool metamodels, tool ontologies, and tool independent ontologies administered by the tool integration knowledge base. Tool ontologies, as created by lifting tool metamodels, reuse knowledge organized in specific modeling domains according to the respective scope of the tool. For instance, the UML ontology can reuse concepts from the *Workflow* domain. Furthermore, generic ontologies such as a *Process* ontology can facilitate mapping between related domains such as *Workflow* and *Sequence*, for instance to facilitate definition of an alignment between corresponding metamodel fragments of Gen and UML.

Ontology design. Design principles for the establishment of an ontological knowledge base for a certain field of domain can be founded upon existing work such as SUMO [42]. A structuring principle specifically relevant to our case of ontologies over metamodels and modeling languages is to separately consider both domain *conceptualizations*, and certain *representations* of domain conceptualizations as defined by modeling languages and metamodels [24]. Therefore, in addition to domain specificity, concept representation forms an additional structuring dimension. For example, in the *Process* domain ontology, representational variants may include network languages and algebraic (block-structured) languages, a variation that is also found, e.g., within UML activity diagrams [28].

Quality of integration. Considering the fact that we use an ontology rather than a mapping to some *semantic domain* [27] 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 [27]. 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. Furthermore, we aim at using concepts which already have a mapping to a semantic domain for building the upper and generic ontologies, to ensure precise understanding of these ontologies. Using semantic domains for integration purposes, however, is out of scope of this project.

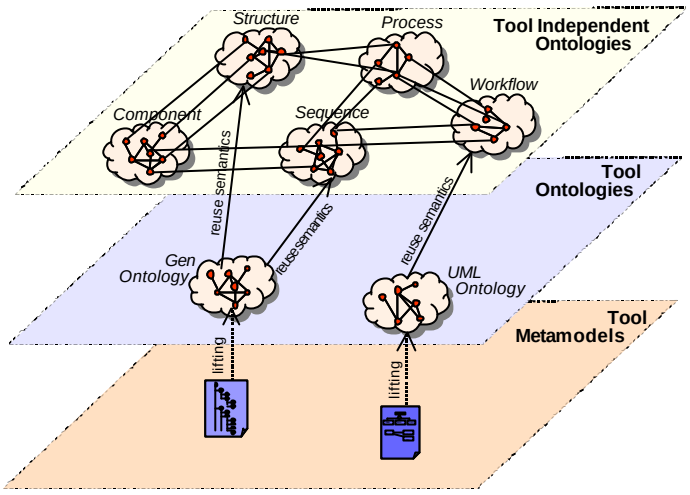

Figure 1-7: Reuse of tool independent ontologies

Reference example. The ontologies within the proposed tool integration knowledge base will be populated with specific instance data, stemming from reference examples of our case study. These reference examples contained in the knowledge base enable the semi-automatic mapping with newly created tool ontologies that are 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.

Semantic merging. To ensure consistency of concurrently developed models, automated detection of merge conflicts is required. Conflict detection will be performed at both syntactic and semantic levels. Syntactic conflict detection is based upon the graph structure of models, and on metamodels defining constraints on that structure. Semantic conflict detection is based upon the meaning of (syntactic) model elements. Essentially, semantic conflict detection improves upon syntactic detection by also taking 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 model element.

As an example of a semantic merge conflict which can arise in merging UML models, consider Figure 1-8, which shows models $Model_2$ and $Model_3$ copied from common ancestor $Model_1$. In $Model_2$, 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 $Model_3$, 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 $Model_2$ and $Model_3$ into $Model_4$ 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.

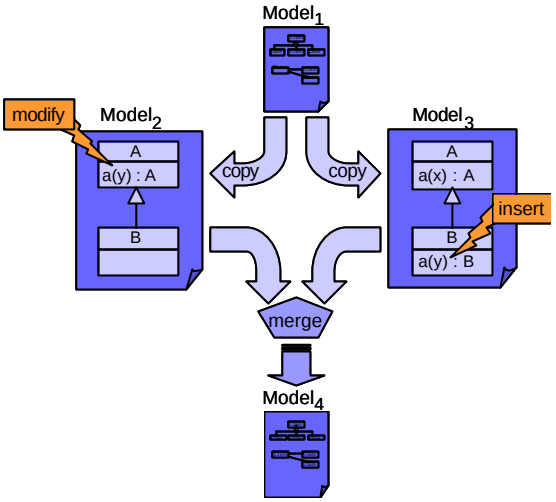

Figure 1-8: Example of a merge conflict caused by inheritance semantics

To enable automatic identification of merge conflicts, the knowledge base will capture both generic and domain-specific conflicts. Generic knowledge about conflicts will be based on generic language constructs, e.g, dependency or inheritance, and conflict detection rules based on them. Specific knowledge will take into account constructs of specific domains, and furthermore conflict patterns dealing with conflicts that arise from typical usages of a set of language constructs, e.g., workflow patterns. Patterns are an approach to declaratively cover certain kinds of conflicts which would otherwise require specification of appropriate algorithms such as a data flow analysis. The knowledge base will be built up using examples from literature (cf. [40]) as well as investigating selected modeling languages.

Knowledge about such semantic merge conflicts will be captured in the knowledge base 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. Because of this, semantic merge conflict detection becomes available as soon as a tool metamodel has been lifted to the semantic level.

Since the merge operation is performed during each check-in, runtime performance needs to be considered. Assuming that a solution based on lifting *metamodels* to the ontology level and performing online reasoning will not provide the required performance, we aim at automatic generation of executable conflict detection programs. Therefore we will investigate in automated translation of rules expressed at the ontology level to rules expressed at the metamodel level, such that they can be performed more efficiently. 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.

Open platform. An open, Internet-accessible platform containing the knowledge base will be provided to the community (cf. Section 7.2). The platform will not only be used to communicate the research results, but also enable community contributions to a growing knowledge base of ontologies and furthermore also tool metamodels and corresponding liftings, mappings, and bridgings, thus enabling reuse of integration solutions. Care has to be taken, however, not to violate copyrights regarding tool metamodels. Therefore, it will not be possible to publish the complete integration solution as developed within the case study. Nevertheless, a lifting and bridging of UML and BPEL metamodels as developed within the project can be safely published.

4.2 Proposed System Architecture

The proposed system *ModelCVS* establishes a semantic infrastructure for model-based tool integration incorporating the output of research goals (1-3) and putting them into a working environment that serves as both a testbed for evaluation of research results and a prototype for a succeeding industrial product. As can be seen in Figure 1-9, the proposed architecture of 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.

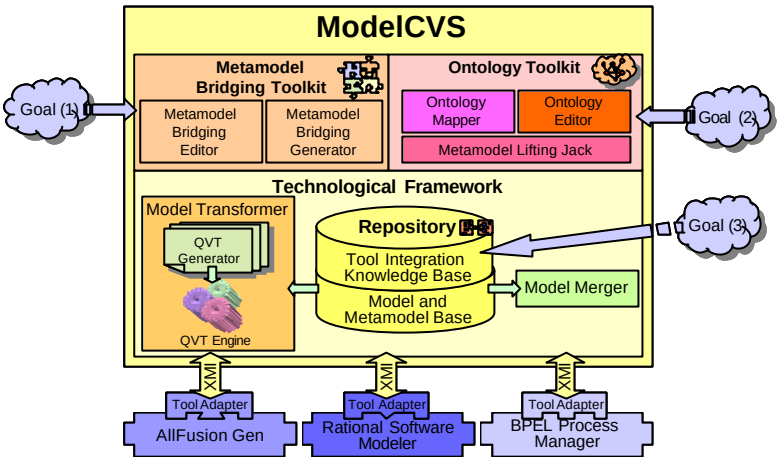

Figure 1-9: ModelCVS' system architecture and related research goals

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 sections. 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 as defined by research goals (2) and (3), i.e., tool ontologies, upper and generic ontologies as well as associated mappings and liftings. In building the *Repository*, existing technologies should be facilitated and integrated. It is planned to use a *versioning system*, e.g., CVS[21] or Subversion[22], as repository back-end, providing persistence and basic versioning capabilities. As front-ends, an existing *MOF-repository*, e.g., MDR[23], along with an existing *ontology repository*, e.g., Sesame[24], should be used as access layer. Thus, they 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 [48] will be used. Several implementations of QVT engines exist, e.g., ATL[25] and MTF[26]. However, as already mentioned, the standardization of a model transformation language is still work in progress. Therefore, available transformation engines and their respective transformation languages will be evaluated as to select one which provides suitable runtime performance and language expressiveness. The metamodel bridges that are specified in a high-level

[21] https://www.cvshome.org/
[22] http://subversion.tigris.org/
[23] http://mdr.netbeans.org/
[24] http://www.openrdf.org/
[25] http://www.eclipse.org/gmt/
[26] http://www.alphaworks.ibm.com/tech/mtf

language (cf. Section 1.4.1) using the *Metamodel Bridging Toolkit* have to be translated into that transformation language. A *QVT Generator* needs to be developed performing this compilation task. Note the bridges are specified independently of the actual transformation language to be used, therefore we are free to choose different transformation languages and engines for different kinds of integration patterns as to achieve optimal performance.

The *Model Merger* also plugs into the *Repository* to provide *syntactic* and *semantic merging* [40] thus enhancing the *textual merging* capabilities already provided by the repository back-end. While syntactic merging capabilities can be built based on existing research results [40], for the semantic merging capabilities appropriate concepts and reasoning tasks, as defined with respect to the tool integration knowledge base (cf. Section 1.4.3), have to be developed. 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 WebDAV[27] can be reused for this purpose. Requirements on the protocol include (a) support for the required repository operations (see Repository), (b) support for handling XMI data, (c) built-in support in existing modeling tools, and (d) platform independence. Existing protocols will be evaluated and either an appropriate protocol will be selected or a new one will be developed. 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. The purpose of a tool adaptor, thus, is to mediate between the tool and ModelCVS. A typical example is to convert between textual formats and XMI as is the case with Gen. Regarding the latter task, a prototypical implementation already exists developed by one of the partners [62]. A framework for tool adapter development will be created, along with concrete tool adapters for Gen and BPEL required for testing and for performing the case study.

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 versioning of metamodels. In particular, the *Bridging Editor* supports creation of bridges using the bridging language defined by research goal (1). Options that will be considered for implementing this editor are, (a) reuse of a generic mapping tool like the Atlas Model Weaver [6] that can be customized to accommodate the specific concepts of the bridging language, (b) definition of a textual syntax for the bridging language and reuse of a text editor, or (c) definition of a graphical syntax and editor, requiring that a suitable development environment for domain specific languages will already be available at the time of development. Furthermore, the *Bridging Generator* makes use of any mappings created at the ontology level to automatically derive bridges between metamodels, taking into account the ontology-level mappings, liftings, as well as any mappings between metamodel elements. 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 realizing research goal (2), i.e., 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,

[27] http://www.webdav.org/

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)[28], the JENA API[29] as well as several specialized inference engines like F-OWL[30] can be used, contributing to the *Ontology Mapper* and the *Ontology Editor*.

References

[1] V. Alexiev, M. Breu, J. de Ruujn, D. Fensel, R. Lara, H. Lausen (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] J. Banerjee, W. Kim, H.-J. Kim, H. F. Korth: Semantics and implementation of schema evolution in object-oriented databases. Proc. ACM SIGMOD 1987

[4] S. Becker et al.: Model-Based A-Posteriori Integration of Engineering Tools for Incremental Development Processes, Journal on Software and Systems Modeling (SoSym), Springer-Verlag, 4(2), May 2005.

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

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

[7] M. L. Brodie: On Modeling Behavioural Semantics of Databases, 7th International Conference on Very Large Data Bases, Cannes, France, Sept. 1981.

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

[9] C. Calvanese et al.: Ontology of integration and integration of ontologies, Description Logic Workshop, 2001.

[10] T.-P. Chang, R. Hull: Using Witness Generators to Support Bi-directional Update Between Object-Based Databases, ACM Symposium on Principles of Database Systems (PODS), May 1995

[11] S. Clarke, R. J. Walker: Composition Patterns: An Approach to Designing Reusable Aspects, Proc. of International Conference on Software Engineering (ICSE), Toronto, Canada, 2001.

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

[13] M. Crubezy, M. A. Musen: Ontologies in support of problem solving, Handbook on Ontologies, S. Staab, R. Studer (eds.), Springer, 2003.

[14] K. Czarnecki, S. Helsen: Classification of Model Transformation Approaches, OOPSLA '03 Workshop on Generative techniques in the context of MDA, Oct. 2003.

[15] A. Doan, et al.: Introduction to the Special Issue on Semantic Integration, SIGMOD Record, 33(4), Dec. 2004

[16] D. Dou, M. McDermmott, P. Qi: Ontology translation on the semantic web, International Conference on Ontologies, Databases and Applications of Semantics, 2003.

[17] A. Earl: Principles of a Reference Model for Computer Aided Software Engineering Environments, Proc. of the Int. Workshop on Software engineering environments, Springer-Verlag, New York, USA, 1989.

[18] J. Estublier, A. D. Ionita, G. Vega: A Domain Composition Approach, Proc. of the International Workshop on Applications of UML/MDA to Software Systems (UMSS), Las Vegas, USA, June 2005.

[19] M. Ferdinand, Ch. Zirpins, D. Trastour: Lifting XML Schema to OWL, 4th International Conference on Web Engineering (ICWE), Munich, Germany, July, 2004.

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

[21] A. Gangemi et al.: Sweetening wordnet with DOLCE, AI Magazine, 24(3), 2003.

[22] B. C. Grau, B. Parsia, E. Sirin, A. Kalyanpur: Modularizing OWL Ontologies, submitted to the 4th International Semantic Web Conference (ISWC-2005), Ireland, 2005.

[23] 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-Verlag LNCS 2830, Erfurt, Germany, Sept. 2003.

[24] G. Guizzardi, L. Ferreira Pires, M. van Sinderen: An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling Languages, ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, 2005.

[25] A. Y. Halevy: Data Integration: A Status Report, in: Datenbanksysteme für Business, Technologie und Web (BTW), Leipzig, February 2003,

[28] http://owl-eclipse.projects.semwebcentral.org/
[29] http://jena.sourceforge.net/
[30] http://fowl.sourceforge.net/

[26] A. Y. Halevy, et al.: Enterprise Information Integration: Successes, Challenges and Controversies, Int. Conf. on Management of Data (SIGMOD), Baltimore, June 2005.

[27] D. Harel, B. Rumpe: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer, 64-72, October 2004.

[28] M. Hitz, G. Kappel, E. Kapsammer, W. Retschitzegger, "UML@Work – Objektorientierte Modellierung mit UML2", 3. Auflage, dpunkt, July 2005.

[29] 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.

[30] 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.

[31] G. Karsai, M. Maroti, A. Ledeczi, J. Gray, J. Sztipanovits: Composition and Cloning in Modeling and Meta-Modeling Languages, IEEE Transactions on Control System Technology, special issue on Computer Automated Multi-Paradigm Modeling, March 2004.

[32] T. Kauppinen, E. Hyvönen: Bridging the Semantic Gap between Ontology Versions. STeP 2004

[33] G. Kiczales et al.: Aspect-Oriented Programming, Proc. of the European Conference on Object-Oriented Programming (ECOOP), Springer LNCS 1241, Finland, 1997.

[34] M. Klein: Combining and relating ontologies: an analysis of problems and solutions, Proc. of the Workshop on Ontologies and Information Sharing (IJCAI'01), Seattle, USA, 2001.

[35] 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.

[36] A. Maedche, S. Staab: Semi-Automatic Engineering of Ontologies from Text, 12th International Converence on Software Engineering and Knowledge Engineering (SEKE 2000), Chicago, July 2000.

[37] A. Maedche, B. Motik, N. Silva, R. Volz: MAFRA – a Mapping Framework for Distributed Ontologies, 13th European Conference on Knowledge Engineering and Knowledge Management, EKAQ, Spain, 2002.

[38] D. L. McGuinness, R. Fikes, J. Rice, S. Wilder: An Environment for Merging and Testing Large Ontologies, 7th Int. Conference on Principles of Knowledge Representation and Reasoning (KR 2000), USA, 2000.

[39] 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.

[40] T. Mens: A State-of-the-Art Survey on Software Merging. IEEE Transactions on Software Engineering, Vol. 28, No. 5, May 2002

[41] P. Mitra, G. Wiederhold, M. Kersten: A Graph-Oriented Model for Articulation of Ontology Interdependencies, Proc. of the 7th Conference on Extending Database Technology (EDBT2000), Konstanz, Germany, 2000.

[42] I. Niles, A. Pease: Towards a standard upper ontology, 2nd International Conference on Formal Ontology in Information Systems, (FOIS 2001), Maine, 2001.

[43] N. Noy, M. A. Musen: The PROMPT suite: Interactive tools for ontology merging and mapping, International Journal of Human-Computer Studies, 59(6), 2003.

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

[45] P. A. Oberndorf: The Common Ada Programming Support Environment (APSE) Interface Set (CAIS), Software Engineering, IEEE Transactions, 14(6), June 1988.

[46] R. Patnayakuni, A. Rai: Development Infrastructure Characteristics and Process Capability, Communications of the ACM (CACM), 45(4), April 2002.

[47] A. D. Preece, et al.: KRAFT: an Agent Architecture for Knowledge Fusion, International Journal of Cooperative Information Systems, World Scientific Publishing Company, 2000.

[48] 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.

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

[50] C. Reichmann, M. Kiihl, P. Graf, K. D. Muller Glaser: GeneralStore - a CASE-tool integration platform enabling model level coupling of heterogeneous designs for embedded electronic systems, Proc. of the 11th IEEE Int. Conference on Engineering of Computer-Based Systems, May 2004.

[51] T. Reiter, E. Kapsammer, W. Retschitzegger, W. Schwinger: Model Integration Through Mega Operations, Workshop on Model-driven Web Engineering (MDWE2005), Sydney, Australia, July 2005

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

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

[54] G. Straw et al.: Model Composition Directives, Proc. of the 7th UML Conference, Lisbon, Portugal, Oct. 2004.

[55] H. Stuckenschmidt, M. Klein: Integrity and Change in Modular Ontologies, International Joint Conference on Artificial Intelligence (IJCAI 03), Acapulco, Mexico, 2003.

[56] H. Stuckenschmidt: Modularization of Ontologies, WonderWeb: Ontology Infrastructure for the Semantic Web, IST Project 2001-33052 WonderWeb, Deliverable, 26.6.2003.

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

[58] M. Uschold et al.: Ontologies and Semantics for Seamless Connectivity, SIGMOD Record, 33(4), Dec. 2004.

[59] R. Volz, D. Oberle, S. Staab, R. Studer: OntoLIFT Prototype, IST Project 2001-33052 WonderWeb, Deliverable 11, http://wonderweb.semanticweb.org/deliverables/documents/D11.pdf, 2003.

[60] A. I. Wasserman: Tool integration in software engineering environments, Proc. of the international workshop on environments on Software engineering environments, Springer-Verlag, New York, USA, 1989.

[61] G. Wiederhold, P. Wegner, S. Ceri: Toward Megaprogramming, Communications of the ACM, Nov. 1992.

[62] M. Wimmer, G. Kramler: Bridging Grammarware and Modelware, Technical Report, Business Informatics Group, Vienna University of Technology, 2005