Towards a Semantic Infrastructure Supporting Model-based Tool Integration

G. Kramler, G. Kappel
Business Informatics Group
Vienna University of Technology
Favoritenstr. 9-17
A-1040 Wien, Austria
++43-1-58801-18804
{kramler,gerti}@big.tuwien.ac.at

T. Reiter, E. Kapsammer, W. Retschitzegger
Information Systems Group
Johannes Kepler University
Altenbergerstr. 69
A-4040 Linz
++43-732-2468-8880
{reiter, ek,werner}@ifs.uni-linz.ac.at

W. Schwinger
Dept. of Telecooperation
Johannes Kepler University
Altenbergerstr. 69
A-4040 Linz
++43-70-2468-9260
wieland@schwinger.at

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

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and techniques – *Computer-aided software engineering (CASE).*
D.2.12 [**Software Engineering**]: Interoperability – *data mapping.*

General Terms
Design, Experimentation, Languages.

1. INTRODUCTION
A rich variety of tools is available supporting different tasks, such as model creation, model simulation, model checking, and code generation. Consequently the exchange of models among different modeling tools becomes an important prerequisite for effective software development processes. Due to a lack of interoperability, however, it is often difficult to use tools in combination, thus the potential of model-driven development cannot be fully exploited. The problems to be dealt with in model-based tool integration are manifold, including differences in model data formats, e.g., relational databases vs. variants of XMI, differences in modeling scope, e.g., general-purpose UML vs. specific workflow languages, and differences in syntax and semantics of languages. Furthermore, practical tool integration needs to cope with large, complex, and evolving modeling languages, e.g. UML. Considering these problems and based on experiences gained in various integration scenarios, [10], [15], [16], we are currently realizing *ModelCVS*[1], a system which enables tool integration through transparent transformation of models between different tools' modeling languages expressed as MOF-based metamodels, as well as versioning capabilities exploiting the rich syntax and semantics of models. It enables concurrent development by storing and versioning software artifacts that clients can access by a check-in/check-out mechanism, similar to a traditional CVS server. This paper outlines the idea and concepts underlying ModelCVS – in particular a two-level approach separating syntactic and semantic issues –, the techniques we intend to use, and the research challenges we will be facing.

GaMMa'06, May 22, 2006, Shanghai, China.

2. LAYERED APPROACH TO TOOL INTEGRATION
To address the problems identified above for providing tool interoperability, the approach taken to the realization of ModelCVS is separated into two conceptual layers that enable to integrate models produced by adjacent modeling tools. The first layer is formed by *architectural model integration patterns* that ensure openness, scalability, and evolvability of a tool integration solution. Further elaborated on in subsection 2.1, these will serve as a basis to define specific bridging tasks and to develop appropriate *bridging operators* that support the identified integration patterns. On top of the first layer, which employs metamodeling technologies, the second layer deals with the use of *semantic technologies* in the form of *ontologies* for the integration of tool metamodels, as well as for semantic versioning capabilities for models. The topic of semantic versioning, however, will not be further expanded in this paper, as we exclusively focus on ModelCVS' integration capabilities and kindly refer the reader to [11]. Subsection 2.2 addresses the integration problem at the semantic level using ontologies in more detail and shows how automation support and reuse capabilities can be achieved. Note that the problem of differing data formats is not covered by this

[1] This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-810806.

approach. We assume a common data format for models, i.e., XMI based on MOF metamodels, and leave the task of interfacing particular modeling tools with so-called tool adaptors.

2.1 Model-based Tool Integration Patterns

The basis for our solution to model-based tool integration is a set of integration patterns that define requirements and working context for the *bridging language*. This language contains *bridging operators* that specifically support the identified integration patterns at a suitable abstraction level, and hence can be more efficiently used than, e.g., generic model transformation languages [15]. By finally deriving *model transformation code* to enforce specific bridging semantics on models, the bridging language is made executable. For reasons of brevity we resort to only elaborating on two proposed integration patterns, namely *translation* and *modularization*, dealing with openness and scalability issues. Other patterns (cf. [11]) address various special situations relevant for model-based tool integration. This includes the *alignment* of models, that allows to keep models of conceptually disparate metamodels synchronized, as well as *metamodel versioning* aiming at the evolution of metamodels.

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 so that models can be *translated* accordingly. 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 makes use of UML Activity Diagrams (UML-AD). 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.

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 should 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 metamodels, like those of UML and Computer Associates' CASE tool AllFusion 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 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. Hence, metamodel fragments are defined in terms of decomposition criteria as well as operators for composing coherent metamodels.

For example, the metamodel of AllFusion Gen can be modularized into several smaller metamodel fragments representing more specific domains, such as User Interface or Workflow. These metamodel fragments may overlap each other, resulting in interdependencies that shall be taken care of in a transparent way, as described in the *alignment* example in [11]. The metamodel fragments facilitate the integration of domain specific GUI and BPEL modeling tools, whose metamodels are directly mapped to metamodel fragments of the Gen tool. Thus 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.

2.2 ModelCVS Semantic Infrastructure

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

Tool Integration Knowledge Base. ModelCVS' semantic infrastructure makes use of ontologies for means of the integration of metamodels by relying on *modeling ontologies*, i.e., conceptualizations of modeling languages. We intend to build up a *tool integration knowledge base*, made up of ontologies capturing knowledge about (the concepts of) modeling languages of different domains, e.g., Workflow, and thus foster immediate reuse capabilities. Furthermore, the ontologies within the proposed tool integration knowledge base will be populated with *specific instance data*, stemming from reference examples of case studies. These reference examples contained in the knowledge base enable the semi-automatic integration of new tool metamodels that are as well populated with instance data from a suitable reference model. Thus, the process of specifying semantics for tool metamodels can be enhanced considerably.

Ontology-based Metamodel Integration. The knowledge captured in the tool integration knowledge base can be utilized in creating bridging specifications in a semi-automatic way by following a sequence of steps. For the sake of simplicity, in the following our running example focuses on the metamodels of BPEL and UML Activity Diagrams to be integrated, only. Details on Fig. 1, which generally depicts our setup used for ontology-based metamodel integration, will be given throughout the following subsections.

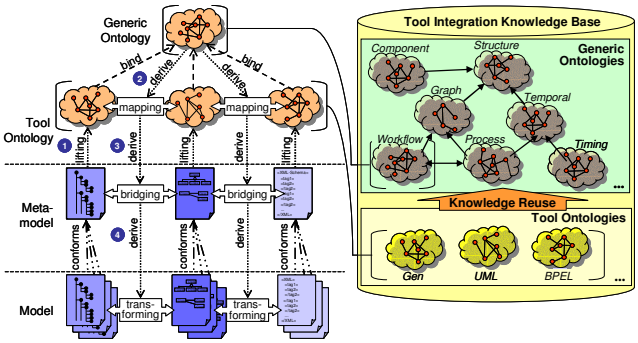

Figure. 1 Ontology-based Metamodel Integration

(1) Metamodel lifting. The creation of an ontology from some kind of metadata like an XML [4] or a DB schema [19] is generally referred to as *lifting*. Metamodel lifting in particular encompasses a mapping of elements in the metamodel to concepts in the ontology, thereby performing a step of abstraction and semantical enrichment such that the ontology explicitly expresses the semantics of the modeling concepts whose syntax is defined by the metamodel. Automatic as well as semi-automatic approaches to lifting have already been proposed in literature. Using ModelCVS' tool integration knowledge base, lifting will be guided by existing (generic) ontologies. During lifting, existing ontologies may be extended to capture the ideosyncracies of specific tools' languages, resulting in so-called *tool ontologies* that reuse semantics defined in generic ontologies. For instance, the BPEL and the UML-AD ontology reuse concepts from a generic 'Workflow' ontology, which in turn plays a role in integrating these. For a more elaborated description of ModelCVS' lifting functionalities we refer the reader to a technical report [11]. A generic solution for lifting arbitrary MOF models (tool metamodels) to tool ontologies can partly automate the lifting process. However, the entailment of specific semantics for newly lifted metamodels naturally requires user intervention.

(2) Ontology-level integration. The use of ontologies is based on the assumption that integration on the ontology layer is more easy to understand and can be automated to a greater extent. Lifting different metamodels' elements to concepts of some common ontology provides the first step of integration by establishing a common terminology. Thereby, it is necessary that the chosen generic ontology covers the domains of both tool ontologies appropriately. Furthermore, based on defined relations between concepts in the ontology, relations between the concepts of specific tools can be deduced, e.g., equivalence, subsumption, or substitutability. Continuing our example, we assume a generic *Workflow* ontology as the common upper ontology. As an example, we can imagine to map all of BPEL's control flow constructs onto the semantically appropriate classes in the *Workflow* ontology. Analogously we proceed with mapping the UML-AD metamodel onto the Workflow ontology. From the two mappings between tool and Workflow ontologies we employ structural reasoning to deduce relationships between ontology classes representing the control flow constructs of BPEL and ontology classes representing the UML-AD metamodel elements.

(3) Derivation of bridging. Once a mapping between tool ontologies exists, the next logical step is to derive bridging operators to express the desired integration behavior on the metamodel level. In a derived bridge between metamodels, depending on the integration pattern in use, semantic correspondence can be expressed by certain metamodel bridging operators accordingly. In case of a translation, a bridging operator might denote the creation of a new target model element for every encountered source model element, whereas in the modularization case, a bridging operator could denote that two model elements should be merged into one at check-out. Getting back to our example, the translation pattern will be the most appropriate, as both the Activity Diagram and the BPEL metamodels cover a largely similar domain. Hence, a relationship on the ontology level between 'equivalent' classes would be derived into a bridging operator relating the metamodel elements that initially got lifted to the respective ontology classes.

(4) Derivation of transformation. After bridging operators between metamodels are established, a code generation step results in QVT code representing the bridging on a lower, finer-grained level, which eventually leads to executable transformations. In the context of a *translation* from BPEL to UML at execution time, this basically results in code querying the source model and populating the target model appropriately.

3. RELATED WORK

Related work mainly encompasses work on tool integration, model transformation languages, and the integration of heterogeneous data sources in terms of models and ontologies.

Apart from categorizing tool integration approaches from a conceptual point of view [1][20], research efforts concerning the mechanical part of tool integration such as CDIF [5] and OMG's recent RFP OTIF[2] are of importance. Since past efforts like CDIF, for instance, were often grounded in large initiatives which have not been widely accepted, we focus on a layered architecture backed by model transformation techniques to most possibly avoid the pitfalls of strongly technology-dependent solutions that suffer from high maintenance overheads and most importantly, poor scalability. Hence, existing approaches in terms of model transformation languages play a key role in our proposed system. With ATL [8] *Bezivin et al.* have developed a hybrid (declarative/imperative) transformation language in response to QVT. Built upon EMF, ATL is especially applicable in context of Eclipse development, as is MTF[3] by IBM, which with a purely declarative transformation definition style might be harder to practically apply than ATL. Although *MDDi*[4] is still in its drafting phase, it provides interesting ideas for model integration in terms of a bus architecture. Although, QVT-like model transformation languages are a cornerstone also of our vision, existing proposals are too generic and lack appropriate abstraction mechanisms for different kinds of *model integration patterns*, which are highly needed in practice and well-known from other research areas such as *federated and multi database systems* [17] and *web service composition* [1]. However, there are only few related approaches providing abstraction mechanisms in terms of, e.g., high-level bridging operators. In the area of *model managemen*t, for instance, Rondo [13] provides high-level operations facilitating the integration of relational and XML schemata, whereas Clarke [2] and Straw [18] target UML models. Finally, considering ModelCVS' semantic integration capabilities, we can benefit from

[2] www.omg.org/docs/mic/04-08-01.pdf

[3] www.alphaworks.ibm.com/tech/mtf

[4] www.eclipse.org/mddi

a large body of literature which can provide useful input for our approach. For a comprehensive overview of this active research area compare, e.g., [9], [14] and [19].

4. CONCLUDING REMARKS

Currently, an early prototype of the proposed system with a basic amount of functionality exists, that already allows to carry out a comprehensive range of intended use cases, which will be validated in an industrial case study involving the Austrian Ministry of Defense and a partner of Computer Associates. The prototype is based on ATL for transforming models, AMW [3] for bridging metamodels and Jena (jena.sourceforge.net) for ontology management. Besides further developing the existing implementation, our focus lies on extending bridging languages and concepts for the implementation of ontology-based integration. We are aware that a successful realization of a system like ModelCVS as laid out in this paper faces a number of issues mainly concerning technological feasibility and practical applicability of the final result.

The use of an ontology layer introduces overhead, which is justified only if the overall integration process can be improved in terms of speed, quality, and reduced complexity. 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 AllFusion Gen. The ontology will express semantics of concepts and consequently integration mappings much more concisely, thus helping to keep mappings comprehensible and manageable. Another benefit is a comprehensive tool integration knowledge base containing readily reusable semantic definitions that may also be published on the Internet. 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.

Key challenges in achieving our goal will be to find generic ontologies and support for the lifting process. These ontologies have to be designed such that the concepts defined by the ontology are easy to grasp while covering the essential concepts of a given domain. A particular challenge will be the handling of concepts that are similar but not equivalent. Although such similarities inhibit full automation of the integration process, treating them at a conceptual level can improve the quality of integration by identifying similarities and defining possible ways of integration. This, however, requires precise modeling ontologies, which again has to be considered in the overhead trade-off. The lifting process should be supported by means of heuristic mapping strategies based on finding structural and linguistic similarities. The utilization of lexical reference systems (e.g., [6]) allows to identify and relate names in question as being synonyms, homonyms, antonyms and the like. Furthermore, the results of heuristic mapping techniques can be greatly enhanced when incorporating instance data during matching [7], which could be accomplished by populating tool ontologies with data of a common reference example. Although both of the above mapping methods can alleviate the burden when creating a mapping from metamodel elements to ontology concepts, a user is still needed to check the appropriateness of a proposed mapping and to eventually give it a finishing touch.

5. REFERENCES

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

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

[3] M. Didonet Del Fabro, J. Bézivin, F. Jouault, E. Breton, G. Gueltas: AMW: a generic model weaver. In: Proc. of the 1ères Journées sur l'Ingénierie Dirigée par les Modèles, 2005.

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

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

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

[7] 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.

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

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

[10] G. Kappel, E. Kapsammer, W. Retschitzegger: Integrating XML and Relational Database Systems, in WWW Journal, Kluwer Academic Publishers, June 2003.

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

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

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

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

[15] T. Reiter, E. Kapsammer, W. Retschitzegger, W. Schwinger: Model Integration Through Mega Operations, Proc. of the Int. Workshop on Model-driven Web Engineering (MDWE), Sydney, 2005

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

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

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

[19] R. Volz, D. Oberle, S. Staab, R. Studer: OntoLIFT, IST Project 2001-33052 WonderWeb, Deliverable 11, 2003.

[20] A.I. Wasserman: Tool integration in software engineering environments, Proc. of the Int. Workshop on Software engineering environments, Springer, New York, USA, 1989.