Thirty Minutes Overview

Table of contents

1. ModelCVS at a glance

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.

ModelCVS at a glance

2. Research Goals

ModelCVS at a glance

2.1. 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.

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. An initial set of integration patterns is proposed, 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 prerequisite for scalable bridging), and versioning (i.e., semantic-based migration of different versions of tool metamodels).

2.2. 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 above.

2.3. 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 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. Layered approach to Tool Integration

To address the challenges 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. 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. In the following subsection we address 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 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.

3.1. 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. 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 (UMLAD). 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 twoway 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. 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.

3.2. 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 ontologybased metamodel integration, will be given throughout the following subsections.

Ontology 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, finergrained 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.

4. ModelCVS Architecture

As can be seen in Figure 5, the proposed architecture for ModelCVS is organized into three major components. First, a Technological Framework provides the actual tool integration services and comprises among others, a repository supporting semantic versioning and transparent model transformation. It is supported by Tool Adapters, i.e., external components that mediate between proprietary tool interfaces and ModelCVS. Second, the Metamodel Bridging Toolkit provides support for defining bridges as to realize integration patterns, manually or automatically. Third, the Ontology Toolkit supports ontology-based metamodel integration in terms of lifting, mapping, and editing capabilities. In the following we will elaborate ModelCVS' components in more detail and lay out some of the design decisions taken during the realization of our prototype, whose functionality is detailed in a simple example.

Architecture

4.1. Technological Framework

The Technological Framework performs the actual tool integration, based on the configurations defined using the Metamodel Bridging Toolkit and the Ontology Toolkit. Its main component is the Repository which provides persistent storage and versioning of complex artefacts. The Repository is divided into two parts. First, the Model and Metamodel Base is dedicated to artefacts of the model and metamodel level, comprising, e.g., models, metamodels, and bridging definitions. Second, the Tool Integration Knowledge Base contains the ontology level artefacts such as tool and generic ontologies, as well as associated mappings and liftings. Concerning the repository for the model and metamodel base, our prototype relies on the Eclipse Modeling Framework (www.eclipse.org./emf) and Subversion (subversion.tigris.org) for persistence and versioning capabilities, along an ontology repository for hosting the tool integration knowledge base. The ontology repository is the only component among those depicted in Figure 5 for which no prototypical solution exists, as for the moment an evaluation of viable solutions is still ongoing we simply store ontologies in the filesystem. The Model Transformer plugs into the repository and provides model transformation capabilities as required for the various tasks defined by the integration patterns. The prototype currently realizes model transformations with ATL [14] as a QVT

Engine [23]. The metamodel bridges that are specified in a high-level language (cf. Section 2.1) using the Metamodel Bridging Toolkit have to be translated into that transformation language. A QVT Generator, prototypically realized through a template based approach, will perform this compilation task. The Model Merger also plugs into the Repository to provide merge conflict detection for models, based on the versioning capabilities provided by the repository back-end. Tool adaptors are a practical necessity, since it cannot be assumed that all tools to be integrated in a tool chain support the data format of ModelCVS. XMI is a natural candidate for exchanging models, as it is based on MOF, and supported by many tools, particularly UML tools.

4.2. Metamodel Bridging Toolkit

This component provides all functionalities dealing with the handling of metamodels and especially the creation of metamodel bridges according to the various integration patterns. A Bridging Editor for the bridging language can for instance be implemented by reusing a generic mapping tool like the Atlas Model Weaver [4] that can be customized to accommodate the specific concepts of the bridging language, as was done in our prototype implementation. The Bridging Generator makes use of any mappings created at the ontology level to automatically derive bridges between metamodels. These automatically generated bridges usually have to be reviewed and refined by the user, using the Metamodel Bridging Editor.

4.3. Ontology Toolkit

Finally, the Ontology Toolkit provides the means for metamodel lifting as well as mapping and editing of ontologies. Its key component is the Metamodel Lifting Jack, which provides means for the creation of an ontology from a metamodel through lifting. Our prototypical lifting solution is able to map EMF's Ecore metamodels onto OWL ontologies, enabling the further process of semantic enrichment. To actually manipulate and make use of the resulting ontologies further, tools like Protégé (protege.stanford.edu), the JENA API (jena.sourceforge.net) as well as several specialized inference engines like F-OWL (fowl.sourceforge.net) can be used, contributing to the Ontology Mapper and the Ontology Editor. Our prototype is currently built on IBM's Integrated Ontology Development Toolkit (IODT) .