Enabling API-based Tool Integration through Aspect-Orientation*

Thomas Reiter[1], Werner Retschitzegger[1], Andrea Schauerhuber[2],
Wieland Schwinger[1], Elisabeth Kapsammer[1]

[1] Johannes Kepler University Linz
{tr|wr|ek|ws}@ifs.uni-linz.ac.at
[2] Vienna University of Technology
schauerhuber@wit.tuwien.ac.at

The Goal: Realising model-based tool integration. 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, and thus, the potential of model-driven software development cannot be fully utilized. To tackle this problem, we are currently realizing a system called *ModelCVS* [4] which is based on semantic technologies and a set of model integration patterns for metamodel integration.

Problem: Often no tool metamodels available. Ideally, model transformations are specified between the metamodels representing a tool's modeling language and are executed with a dedicated QVT model transformation language (e.g., ATL [3]), which, for instance, transforms a model of a UML Activity Diagram into a model of a BPEL process specification. In many cases, however, tools do not offer an explicit metamodel ready for model transformation in the sense of QVT. For practical applicability, it is of crucial importance that a tool integration system also offers the ability of integrating tools through a programming interface (API). An ad hoc approach would be to implement an Observer which listens to events issued from the source tool's API and delegates the semantically equivalent method calls towards the target tool's API. Thereby, the events will be propagated in a way as that modifications in the source tool are reflected in the target tool appropriately.

Although this approach does not require an explicit metamodel representation, manually incorporating the notification code into the tool's source code is problematic. On one hand, this can prove to be cumbersome or even impossible, and on the other hand, with respect to separation of concerns, this glue code should not be mixed with the tool's native code.

Solution: Aspect-orientation to the rescue. Aspect-orientation can provide a clean solution in terms of encapsulating the notification mechanism within an aspect, thus avoiding the need to interfere with the tool's source code. The aspect code as well as the observer code that actually performs the transformation can be derived from a set of semantic correspondences established in an *API weaver*, which in turn could be built on a tool like AMW [1]. In our case, weaving would be an activity of establishing semantic correspondences between API method calls, for instance. The left-hand side of Figure 1 shows how these correspondences can be used to weave two tool APIs.

Problem: 'Mysterious' APIs. Even though tools may provide a public API, especially proprietary or open source tools often lack documentation and may not be well understood, which makes an immediate weaving merely impossible. As an example, instead of providing a set of distinct factory methods for object creation (e.g., *createClass()*, *createAttribute()*, …), generic-sounding and parameterized methods (e.g., *create(String[] params)*) may be used for object creation. Furthermore, the problem of not knowing which sequence of method calls constitutes a valid model manipulation arises.

Solution: Event Recording. Hence, a systematic method, which allows to reveal the inner works of a tool's API is proposed. The right-hand side of Figure 1 depicts a user-guided process which *records* (2) API events (e.g., method calls) occurring after certain tool interactions (1), like a GUI command or a model import. After a certain interaction is completed, a user's *commit* (3) assembles the recorded events as a valid *transaction* constituting a consistent model manipulation. Instead of the previously mentioned weaving of APIs, the elicited transactions (e.g. sequences of method calls) can now be subject to a weaving step.

* This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-810806, and the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002

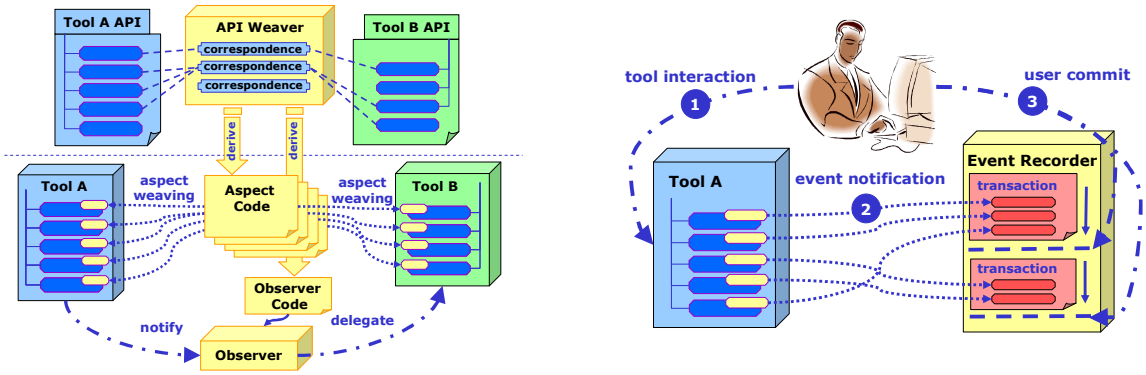

Fig. 1: API Weaving (left) and Event Recording (right)

Problem: Need for complex transformation logic. Direct API or transaction weaving works well if tools can be rather straight-forwardly mapped onto each other. The need for more complex transformation rules, however, would call for a more sophisticated API weaver and complicate the observer code generation, which would result in a badly-scaling solution that goes far beyond an event delegation mechanism.

Solution: QVT Engine. To tackle this problem, a tool metamodel (e.g., built with EMF) can be re-engineered from a tool API or from elicited transactions. Instead of directly delegating events, the observer now instantiates and manipulates a model, which represents the current state of the tool and which conforms to the re-engineered metamodel. Transformation specifications (e.g., expressed in ATL) can be defined between the metamodels to allow for complex model transformation logic. As shown in Figure 2, when a transformation is executed, the created or updated target model then notifies an observer which delegates events towards the tool, which changes its internal state accordingly.

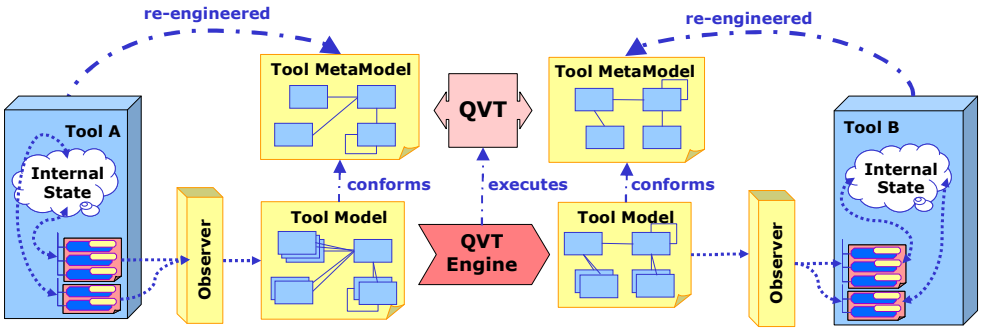

Fig. 3: Complex API Integration realized with a Model Transformation Engine.

Related Work. The proposed approach is related to [2], which focuses on domain composition through aspect orientation. However, we have a narrower focus on tool integration and aim at providing pragmatic solutions for problems encountered in various tool integration scenarios that lack an explicit metamodel representation. Furthermore, we aim at eliciting transactions from unknown APIs, deriving aspect code from integration specifications, and in case of the latter scenario rely on a dedicated model transformation language to integrate tools via models representing a tool's internal state.

Conclusion. The advantage of the proposed approach is that tool's without an explicitly defined metamodel can be systematically integrated via their APIs by relying on the aspect-orientation paradigm. Although we put our focus on model-based tool integration, the approach could be extended towards other integration tasks involving not just tools, but frameworks, libraries, and the like.

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

[2] Estublier J., Vega G. D. I. Anca: Composing Domain-Specific Languages for Wide-scope Software Engineering Applications. MoDELS 2005, October, Montego Bay, Jamaica, (2005)

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

[4] Kappel G., Kapsammer E., Kargl H., Kramler G., Reiter T., Retschitzegger W., Schwinger W., Wimmer M: On Models and Ontologies - A Layered Approach for Model-based Tool Integration. Modellierung 2006, Innsbruck, March, (2006)