Harmless Metamodel Extensions with Triple Graph Grammars

Jendrik Johannes
Technische Universität Dresden
Software Technology Group
01062 Dresden, Germany
jendrik.johannes@tu-dresden.de

Tobias Haupt
Technische Universität Dresden
Software Technology Group
01062 Dresden, Germany
s0500251@inf.tu-dresden.de

ABSTRACT

Designing Domain Specific Languages through metamodeling is an emerging engineering technique in software development. Language development, however, is also always tool development—which can be costly when languages are developed from scratch. Costs can be saved by developing new languages as extensions of existing ones—effectively by extending metamodels. Here, existing tools, developed for the existing language, can be reused to a certain degree. We argue that a certain group of metamodel extension is *harmless*: keeping existing tools functional, while integrating new tools to handle additional functionalities. To realize a platform for this, two things are required: 1) a transformation engine for the metamodel extension and 2) a synchronization mechanism for tool integration. In this paper we show how Triple Graph Grammars can be used to define both and introduce an interpreter for these grammars that works in an environment based on the Eclipse Modeling Framework.

1. INTRODUCTION

In contrast to general-purpose modeling language development, Domain Specific Language (DSL) development has to be much more cost-efficient. High development costs are not acceptable since each DSL is designed for a specific problem domain with a limited set of applications possibilities. A main issue for high development costs is tool building, which is crucial to render newly designed languages usable. Costs can be reduced by reusing existing tools where feasible instead of developing new ones from scratch. One way to achieve this is to reuse metamodels as well—extending existing ones rather than designing new ones.

Metamodel extension seems to be a promising direction for simplifying DSL design and implementation. Especially, if such extensions can be captured in a generic formalism for extending arbitrary metamodels in a similar fashion. Then tools based on the original metamodels and tools that know about the specific extension can be used in combination, without needs to develop new tooling. We have identified one such extension formalism—the Reuseware formalism [4][1]—which we believe of strong importance: adding reuse abstractions to a language. That is, adding notions of components, modules, aspects, or similar to leverage the reuse of artifacts written in that language.

[1]The formalism was defined for context-free grammars here. It was ported to metamodels, but publication is still pending.

The Reuseware extension of a metamodel is *harmless*: 1) all original language syntax and semantics are preserved and 2) the added features are syntactically closely related to original ones. For a harmless extension, reuse of the original modeling tools to design models is applicable: 1) the original language is supported anyway and 2) the new constructs can be mimicked by existing ones using naming conventions or an escape mechanism, like annotations or comments, if available.

Nevertheless, it is desired to translate such models into models conforming to the extended metamodel such that the extension specific (in this case the Reuseware specific) elements can be clearly identified. This allows further processing of the models (in this case by composition editors and engines). This dualism of tools working on two different representatives of the same model requires model synchronization.

In this paper we propose to utilize Triple Graph Grammars (TGG) [7, 5] for both, the metamodel extension and the synchronization of models. To realize this, we require a TGG environment which integrates into our tool chain. We are aiming for solutions based on the Eclipse Modeling Framework (EMF) [1]. Since there is no TGG engine available for EMF that supports model synchronization, we developed, driven by our use case, the TGG interpreter *Tornado*.

The remainder of this paper is structured as follows. In Section 2 the metamodel extension and model synchronization scenario within the Reuseware composition context is introduced. Following this, requirements for a TGG engine and their realizations in the Tornado engine are described in Section 3. Some related work is discussed in Section 4. Section 5 concludes and points at future directions of work.

2. METAMODEL EXTENSION AND MODEL SYNCHRONIZATION SCENARIO

We argue that through harmless metamodel extension a language can be enriched with new powerful features while preserving compatibility with original tooling. Such a harmless extension is the Reuseware formalism [4]. In this section we describe how Triple Graph Grammars are used 1) to perform the Reuseware extension on arbitrary metamodels and 2) to preserve tool compatibility.

2.1 Metamodel Extension

A metamodel extension can be expressed through a model transformation, which in turn can be formulated as a TGG.

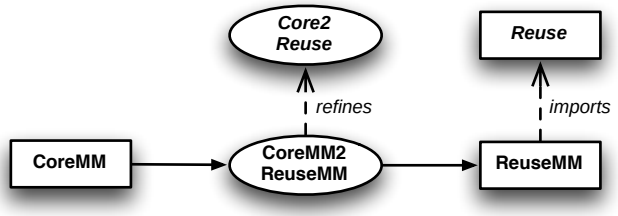

Figure 1: Reuseware metamodel extension formalism

Usinig this notion, the basic idea of the Reuseware metamodel extension is illustrated in Figure 1. A metamodel `CoreMM` of the original language (the *core metamodel*) is extended by the transformation `CoreMM2ReuseMM` to an extended metamodel `ReuseMM` (the *reuse metamodel*). The transformation is a specialization of a generic transformation (`Core2Reuse`) and the extended metamodel imports a metamodel defining the basic Reuseware concepts (`Reuse`).

For illustrative purposes, we will use a simplified version of the Reuseware metamodel extension formalism in this paper. The main idea is, to introduce metaclasses for variation point definition[2] for a selected set of existing metaclasses. Variation points can be seen as placeholders for concrete instances of metaclasses. They can be used to define template-like incomplete models (so-called *model fragment*s) that can be reused to compose complete models. A variation point is typed with the metaclass it stands for. That is, it can only be replaced by an instance of that metaclass during composition. This is reflected in the formalism, which can be formulated in abstract TGG rules. One rule is shown in Figure 2, where a metaclass (`Reusable`) is introduced as superclass of the original metaclass and the corresponding variation point metaclass. The latter additionally inherits features from the abstract concept of variation point. The other rules that complete the formalism are not shown.

As a concrete example, consider an UML-like metamodel defining the metaclasses `Package` and `Class` into which we would like to introduce variation points for classes. We specialize the TGG rule from Figure 2 to match the metaclass with the name `Class` by adding the constraint `name == "Class"` to the `coreEClass`. Application of the grammar, interpreted as a left-to-right transformation, gives us the extended metamodel, where `ClassVariationPoint`s are available as alternatives for `Class`es. Note that the specialization of the rule can be automated by a simple wizard tool, where the developer only states for which metaclasses variation points shall be introduced. This effectively hides the (sometimes complex) TGG rules from the developer.

2.2 Model Synchronization

The second, and more challenging, application of TGGs is the synchronization between *models* (instances of the core metamodel) and *model fragments* (instances of the reuse

[2]The full Reuseware formalism distinguishes different kinds of variation points and allows structuring and grouping of them as demonstrated in [3].

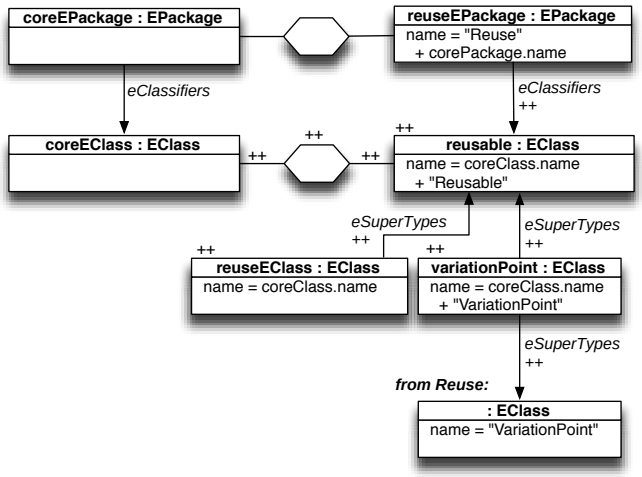

Figure 2: Operational TGG rule to introduce a variation point metaclass (in EMF-based metamodels)

metamodel). The synchronization is required because models are used to impersonate model fragments which allows the reuse of existing editors. The real model fragment, used in Reuseware specific tools, then describes conceptual the same as the impersonator and both have to be synchronized. While we focus on editor integration in this section, it is also beneficial to enable synchronization between fragments and composed models. That is a powerful and desired feature in a Reuseware environment, since it allows to modify fragments directly in a composition result. It can be achieved by realizing the composition algorithm in terms of TGGs as well.

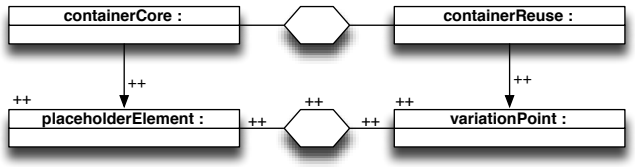

Figure 3: Generic synchronization rule for variation points

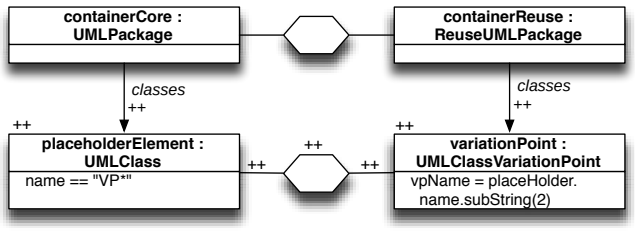

Figure 4: Specialized synchronization rule for UML class variation points

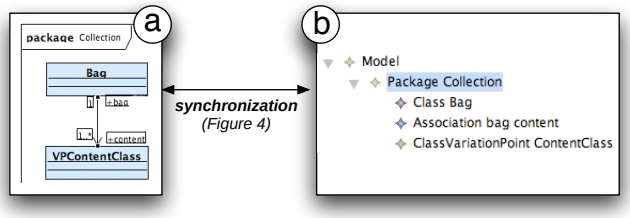

Figure 5: Synchronization of a UML model and a ReuseUML model fragment

As mentioned, we reuse existing editors and define model fragments by means of the core language to save the effort of developing new editors for every reuse metamodel. Variation points, not available as concepts in those editors, are expressed through name conventions, annotations, comments or similar means. TGG rules can then be applied to translate such marked fragments into real model fragments (i.e., instances of a reuse metamodel). An abstract TGG rule for this is illustrated in Figure 3.

The abstract rule can be specialized to be used in the UML scenario (cf. Figure 4). The rule defines that a naming convention `VP*` can be used to express variation points through classes. We can then use any EMF-based UML editor to model a UML model fragment (cf. Figure 5a). Application of the rule would translate it to a real model fragment (shown in an abstract tree notation in Figure 5b). Furthermore, if changes are made later on in the editor, the TGG engine will reflect those changes on the real fragment.

3. TORNADO TGG ENGINE

From the scenario presented in the last section, the following requirements for a TGG engine are derived:

- *Requirement:* EMF as a model repository and Ecore as a metamodeling language should be supported.
  Justification: EMF is an accepted tool platform for modeling and most (open-source) modeling editors are based on it, and are thus targets for tool integration
- *Requirement:* Triple Graph Grammar rules should be interpretable for incremental model synchronization.
  Justification: Different physical model elements can exist for the same conceptual element (e.g., model fragments and composed models) and have to be synced.
- *Requirement:* A rule abstraction mechanism (like rule inheritance) has to be provided to efficiently define generic rules and specialize them.
  Justification: The rules used in the metamodel extension and the model synchronization always have some common part independent of the concrete metamodel to extend. It is convenient to define this part once and specialize it for concrete metamodels.

The following sections describe how these requirements are fullfilled by the Tornado TGG engine and discuss problems that are still topics of research.

3.1 Processing of EMF–based Models

The kernel of the Tornado engine is a pattern matching and rule application algorithm which is capable of adding and synchronize model elements by updating or reseting. It utilizes the reflection facility of EMF for both: the analysis and the updating of the model graph. In EMF, metaclasses have the type `EClass`. Attributes and references are stored in `EStructuralFeature`s possessed by `EClass`es. The identification of model element types is done by matching the `name` feature of their `EClass`. The matching of references and attributes is a name-matching over the names of the `EStructuralFeature`s of the corresponding `EClass`. Through this metamodel independent implementation, models written in arbitrary EMF-based DSLs can be addressed on the left and right side of TGG rules.

3.2 Incremental Model Synchronization

To support incremental model synchronization on changes in the involved model graphs, the engine uses correspondence models that are persistent over all successive synchronisation tasks. These are also EMF-based models which consist mainly of `CorrespondenceNode`s and `CorrespondenceLink`s. The EMF model import feature—the possibility to reference elements in other models—is utilized to reference elements of the left- and right-hand-side models. EMF ensures that these links are kept when models are persisted.

Through the correspondence model, inconsistencies are recognized. If elements are deleted in one of the models this can be easily observed because a reference to the imported models breaks. Changes of elements (e.g., changes of attribute values) are in general difficult to track. Here another advantage of the EMF-based system can be utilized. If we activate the engine at runtime we can observe the model elements and react on changes immediately. This can be easily implemented since EMF comes with an observer mechanism that is inherited by all EMF-based applications.

In synchronization, the major problem is to derive the required and optimal transformation steps towards a consistent graph. As an example consider the change of attributes of an element. This can put constrains on the element that prevents the rule, used to create it, from matching any longer. The problem can be resolved by resetting the formally applied rule. Previously created elements have to be deleted and, consequently, all depending rule applications have to be reseted as well. An algorithm addressing this with certain optimizations is described in [5]. However, the reseting of rule applications is a severe action. Often, it leads to a deletion of elements that later have to be re-created by re-applications of rules leading to possible loss of information. We currently work on an improvement of the algorithm that tries to match other rules first on elements before deleting them irrevocably.

3.3 Rule Definition with Rule Inheritance

To realize the required abstraction mechanism for TGG rules, a rule inheritance mechanism has been realized in Tornado's TGG rule definition language. The mechanism supports multiple inheritance (as known from object-oriented languages) on the base of nodes and edges. That is, features of nodes can be refined and additional edges can be connected to nodes. Through multiple inheritance, nodes can

be merged if their feature and connected edges conform. The mechanism allows for abstract (i.e., incomplete) rules which can not be applied without being refined. An example are nodes without type declaration as used in Figure 3. The other features of the rule definition language are designed based on the general TGG formalism.

We provide an editor for rule definition which has been designed and generated with the Graphical Modeling Framework[3]. In the Reuseware metamodel extension case it is primarily applied to define the abstract rules. Figure 6 shows the rule from Figure 3 defined for the Tornado engine utilizing the editor. The concrete specializations can be semi-automatically derived by DSL-developers using provided wizards.

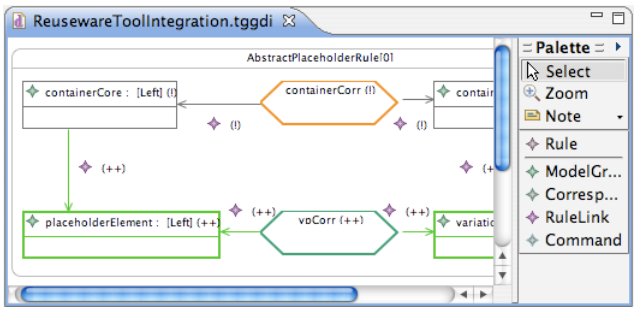

Figure 6: TGG rule editor

4. RELATED WORK

Fujaba provides the most advanced TGG engine at the moment [8]. Unfortunately, integrating it into an EMF environment is not trivial. A close integration, for instance, with the EMF observer mechanism to track changes is not realistic at the moment.

Another EMF-based interpreter is currently developed at the University of Paderborn [5, 6]. Unfortunately, up to now it only supports one-directonal model transformations.

The Atlas Model Weaver [2] can handle EMF-based models and realizes a similar idea as TGGs. Instead of synchronization models, so-called *weaving models* express relations between elements, which do not necessarily express equality. Thus, there is no interpreter for the synchronization semantics we require. However, the underlying Atlas Transformation Language[4] provides a rule inheritance mechanism with similarities to our TGG rule inheritance.

Other EMF-based tools for model management tasks, which were considered as alternatives to TGGs, are developed in the GMT project[5] (Epsilon, VIATRA2, oAW, and others). While these are powerful tools, they all process models in a batch-like fashion, have little support for model update, and no build-in synchronization mechanism. Thus, they could have been used as a base for the implementation of the synchronization mechanism. Because of the desired close EMF integration we decided on a Java implementation.

[3]http://www.eclipse.org/gmf
[4]http://www.eclipse.org/m2m/atl
[5]http://www.eclipse.org/gmt

5. CONCLUSION AND FUTURE WORK

The result of our work is twofold: First, we showed how Triple Graph Grammars can be applied in a scenario of harmless metamodel extension. Second, we developed a TGG interpreter that meets our specific needs.

Our feeling is that the Reuseware formalism is only one prototype of what we called *harmless metamodel extensions*. It is an interesting direction to identify other useful extensions like this. Possibly, the generic Reuseware TGG rules can be further abstracted to a set of TGG rules reusable in any harmless metamodel extension. This, paired with tool support for semi-automatic TGG rule specification, could lead to powerful, yet easy to use, DSL development tools.

The development of the TGG engine driven by the concrete use case has revealed interesting challenges. At many points trade-off decisions had to be made concerning expressiveness versus usability. We are optimistic that ongoing development will result in an interpreter that is utilizable in the EMF world by TGG experts for several tasks. However, it seems that the direct usage of TGG rules for model synchronization in software modeling is a too complex and error-prone task for many developers. Systems, like the presented one, where TGG experts define a set of abstract rules which are then refined by developers in a semi-autmatic way offer an interesting combination of the powerful TGG formalism with easy usability.

6. ACKNOWLEDGMENTS

This research has been co-funded by the European Commission within the 6^{th} Framework Programme project Modelplex contract number 034081 (cf. `www.modelplex.org`).

7. REFERENCES

[1] F. Budinsky, E. Merks, and D. Steinberg. *Eclipse Modeling Framework 2.0*. Addison Wesley, Jan. 2007.

[2] M. Didonet Del Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas. AMW: A Generic Model Weaver. In *Proc. of the 1^{res} Journées sur l'Ingénierie Dirigée par les Modèles, France, Paris*, 2005.

[3] F. Heidenreich, J. Johannes, and S. Zschaler. Aspect-Orientation for Your Language of Choice. In *Proc. of the 11^{th} Int'l Workshop on Aspect-Oriented Modeling (to appear), Nashville, TN*, September 2007.

[4] J. Henriksson, J. Johannes, S. Zschaler, and U. Aßmann. Reuseware – Adding Modularity to Your Language of Choice. *Proc. of TOOLS EUROPE 2007: Special Issue of the Journal of Object Technology (to appear)*, 2007.

[5] E. Kindler and R. Wagner. Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios. Technical Report tr-ri-07-284, University of Paderborn, June 2007.

[6] C. Lohmann, J. Greenyer, J. Jiang, and T. Systä. Applying Triple Graph Grammars For Pattern-Based Workflow Model Transformations. *Proc. of TOOLS EUROPE 2007: Special Issue of the Journal of Object Technology (to appear)*, 2007.

[7] A. Schürr. Specification of graph translators with triple graph grammars. In *Proc. of the 20^{th} International Workshop on Graph-Theoretic Concepts in Computer Science, Herrsching, Germany*, volume 903 LNCS. Springer, Berlin, 1994.

[8] R. Wagner. Developing Model Transformations with Fujaba. In H. Giese and B. Westfechtel, editors, *Proc. of the 4^{th} International Fujaba Days 2006, Bayreuth, Germany*, volume tr-ri-06-275, pages 79–82. University of Paderborn, September 2006.