Fujaba’s Future in the MDA Jungle

Fully Integrating Fujaba and the Eclipse Modeling Framework?

Basil Becker, Holger Giese, Stephan Hildebrandt, Andreas Seibel
Hasso Plattner Institute for Software System Engineering
Prof.-Dr.-Helmertstr. 2-3
14482 Potsdam, Germany


ABSTRACT
Fujaba has a long tradition as a CASE tool and provides outstanding features with its proprietary extensions in form of Story Diagrams and TGGs. However, in its current form Fujaba is not ready to support the development of tools or applications for MDA. In this paper, we report about our latest work on Fujaba and Eclipse and outline the used solutions developed and employed so far. In addition, we outline our vision of the next open Fujaba version, which more smoothly integrates with standard MDA environments and thus could help that the Fujaba concepts and our research results can be employed in standard development projects in the MDA world.

1. INTRODUCTION
Fujaba has a long tradition as a CASE tool and provides outstanding features with its proprietary extension in form of code generation for UML Class Diagrams [5], Story Diagrams [2] and TGGs [4]. However as outlined in this paper, Fujaba is currently not ready to support the development of MDA tools or applications. To employ Fujaba and its concepts, we have to restrict our projects to UML models encoded with the proprietary UML dialect supported by Fujaba. In practical applications, also other MDA techniques are required, which have to be used in combination. In fact, an open version of Fujaba is required, which can be employed in a well established technological platform.

To be compatible with such a platform, Fujaba must be able to support not only its proprietary UML models as basis for its concepts but also standard meta models and the standard APIs possibly generated by other non Fujaba code generators. The Fujaba offspring MOFLON [1] with its support for MOF 2.0 thus seems to be a good solution. However, neither MOF 1.x nor MOF 2.0 has found much acceptance in practice due to its complexity. Instead, the Eclipse Modeling Framework (EMF)\(^1\) has become more or less the quasi-standard for meta modeling, which provides a partial implementation of the OMGs Essential Meta Object Facility (EMOF) in form of the Ecore meta model. In addition to basic support for modeling meta models and code generation from the Ecore meta models, EMF also provides model serialization/de-serialization and validation facilities. Furthermore, a large variety of other tools for the development of tools for MDA and DSLs based on EMF exist, e.g., openArchitectureWare\(^2\), ATL\(^3\) and GMF\(^4\). Also a number of modeling tools already employ EMF such as the UML2 Tools\(^5\) for UML 2.0 modeling and TOPCASED\(^6\) for SysML modeling.

In this paper, we report about our latest work on Fujaba and Eclipse and outline the used solutions developed and employed so far. In addition, we outline our vision of the next open Fujaba version, which more smoothly integrates with standard MDA environments and thus could help that the Fujaba concepts and our research results can be employed in standard development projects in the MDA world.

The paper is structured as follows: we first present our current achievements and requirements for the meta model based development of tools in Section 2 looking into meta modeling (Section 2.1), Story Diagrams (Section 2.2), Triple Graph Grammars (Section 2.3) and consistency rules (Section 2.4). Then, we discuss our requirements for UML 2.0 based development of applications in Section 3. Finally, we discuss our findings in Section 4 looking into open issues and our envisioned solution before our final conclusions.

2. META MODEL BASED TOOL DEVELOPMENT

2.1 Meta Modeling
Fujaba uses a proprietary meta model to represent meta models. This makes interoperability with modeling tools, based on other meta model, difficult. To create transformation rules for our model transformation system (described in Section 2.3), we first have to import the existing EMF meta models of the models, we want to transform, into Fujaba. Unfortunately, there was no possibility to import EMF based meta models into Fujaba, so far. Manually remodeling meta models is usually not an option due to the complexity of these meta models (e.g., UML 2.0). Therefore, we developed a simple import plugin for Fujaba, which uses EMF to de-serialize Ecore files and create the EMF model tree in-memory. This tree is then traversed in order to create a new Class Diagram, classes, attributes, methods, generalizations and associations into Fujaba.

\(^1\)http://www.eclipse.org/modeling/emf/
\(^2\)http://www.openarchitectureware.org/
\(^3\)http://www.eclipse.org/m2m/atl/
\(^4\)http://www.eclipse.org/modeling/gmf/
\(^5\)http://www.eclipse.org/modeling/emf/
\(^6\)http://www.topcased.org
Problems arise if the Ecore model is split among several files. An example is the SysML meta model used in TOPCASED\textsuperscript{7}. This meta model references the UML 2.0 meta model from the UML 2.0 plugin, which in turn references the Ecore meta model. Within Eclipse, platform URIs are often used to reference elements in models provided by other plugins. In Fujaba, these URIs cannot be resolved, of course. Therefore, the meta models have to be extracted from the providing plugins, put into the same directory, and all references in the files have to be made relative using a text editor.

Then, all models can be imported into Eclipse in the order in which they depend on each other. In the SysML case, first the Ecore meta model must be imported because it is independent from the other models. Then, the UML 2.0 meta model and finally the SysML meta model can be imported. The reason is, that elements referenced from another file are not converted to Fujaba by the importer. The Fujaba objects must already exist. After the import, the meta models are available for use in Fujaba.

### 2.2 Story Diagrams

Story Diagrams allow modeling behavior, using graph rewriting rules. They are similar to Activity Diagrams extended by special Story Activities that contain graph rewriting rules. In Fujaba, Story Diagrams specify the behavior of a method related to a class. A code generator can generate Java code from these Story Diagrams that implements the modeled behavior.

Nevertheless, a problem occurs with meta models imported from EMF. In Fujaba, compositions are always bidirectional. This is not required in EMF. EMF provides the method `eContainer()` to access the container of a contained element. Importing a unidirectional EMF composition into Fujaba, automatically creates an explicit inverse association from the contained element to the container. Of course, this association does not exist in the EMF model or the code generated by EMF. This poses a problem for Fujaba's code generation from Story Diagrams because the code generator might use these `non-existent` associations in the generated code. This makes the generated code incorrect and it must be corrected manually.

### 2.3 Triple Graph Grammars

In the context of Model-Driven Development (MDD), various different models describe a system under construction. These models can be used to describe different aspects, subsystems or overlapping parts of the system at different levels of abstraction. Thus, models are not completely independent but rather relationships between them exist, e.g., models are derived from other models by means of model transformations or they are kept in sync by means of model synchronization. Model transformation and synchronization require a transformation system to transform different model types and propagate changes between models. A model transformation system must be seamlessly capable of being integrated into an existing tool chain.

The Triple Graph Grammar (TGG) based model transformation system introduced in \cite{3,4} has recently been migrated to Eclipse. Thus, TGG based model transformations can be easily used within Eclipse. This integration also allows exploiting EMFs change notification mechanism for efficient synchronization of models. However, the transformation rules, that perform the actual transformation, are not created within Eclipse but within Fujaba by means of Story Diagrams.

#### 2.3.1 Model Transformation on EMF Models using Triple Graph Grammars

The model transformation system consists of a series of Eclipse plugins. Core of the transformation system is the transformation engine. It loads the source model of the transformation and the required set of transformation rules to create a target model. The engine is independent from the model types. It just executes the transformation rules, which perform the actual model transformation. The transformation rules are therefore specific for two types of models, the source and the target model. Transformation rule sets are deployed as separate Eclipse plugins.

The TGG-based model transformation algorithm supports bidirectional model transformation and synchronization. The transformation can be executed on two conceptual levels: file-based transformations and in-memory transformations. The file-based transformation loads the source model from disk and saves the transformed target model to disk, as well.

The in-memory model transformation (shown in Figure 1) directly manipulates the models in the modeling tools' memory. For this purpose, a special tool adapter is required that allows accessing the models. If the modeling tool is another EMF-based Eclipse plugin, such an adapter is usually quite simple. In that case, the adapter just has to open the modeling tool, get the resource object that contains the model, and return it to the transformation engine. The transformation engine can then operate directly on the model. If the modeling tool is an external program, the adapter is much more complex and must translate modifications from the EMF modeling space to the external modeling tool, e.g., via COM or .NET whenever the external tool provides an appropriate interface. To execute a model transformation, the user has to provide a source and target model and a set of transformation rules.

#### 2.3.2 Creating Model Transformation Rules with Fujaba

Figure 2 shows the main steps, which are required to create transformation rules within Fujaba. Before the actual transformation rules can be defined, the meta models of the source and target models must be available in Fujaba. In case of Ecore meta models, these must be either remodeled or imported into Fujaba. Besides the source and target meta models, a third correspondence model is required for a model transformation with TGGs. This correspon-

---

\textsuperscript{7}We encountered this problem in an industrial development project where a model transformation for SysML models was developed.

---

Figure 1: Overview of an in-memory transformation
import EMF Meta Models
Create TGG Rules
Derive Story Diagrams
Create Transformation Java Code
Deploy Transformation Rule Plugin
Use Transformation Rules
Adapt Java Code Manually
Create Configuration Files
Put Java Code into Eclipse Plugin
Figure 2: Necessary steps to create executable transformation rules from TGG rules

2.4 Consistency Rules

We are currently working on another MDA application, which is a tool for model-based development of UML based applications deployment where we employ the notion of component development and component configuration & deployment. The developed software components provide software development aspects as well as deployment aspects like well defined variation points for configuration during deployment.

Fujaba does not provide the necessary modeling artifacts for this problem domain. In contrast, Eclipse does provide with EMF an appropriate environment for defining meta models, based on Ecore. Thus, we employ Eclipse and EMF as modeling environment. However, Fujaba is still a major part of the tool because we define consistency rules between domain specific models by means of Story Diagrams. Figure 3 shows the tool, its artifacts, the role of Fujaba and the relationships between these artifacts. Consistency rules have to be specified on the meta models and are subsequently applied on the domain specific models, which are conform to their meta-models (instantiations). If relationships between domain specific models exist, which is quite common in our case, consistency rules are applied in order to check whether all required domain specific models are consistently modeled.

In order to realize such a tool, we employ the previously explained EMF importer and the EMF code generation capability of Fujaba’s code generation. Thus, we import the EMF meta models into Fujaba, specify the consistency rules based on these meta models and subsequently generate EMF compatible code for the consistency rule application within Eclipse.

2.5 Open Issues

The major issue is the different meta models of Fujaba and EMF. This requires a conversion of meta models of one type to another. However, this conversion does not work seamlessly because of the major conceptual differences. One mentioned problem is compositions, which are always bidirectional in Fujaba. This makes the generation of correct Java code in Fujaba difficult.

Furthermore, the described workflow for creating a set of transformation rules is far too complex for an ordinary user of the transformation system. The process involves two different platforms, Eclipse and Fujaba, and requires a lot of manual steps. This workflow must be limited to only one platform and be automated as much as possible.
3. UML 2.0 BASED APPLICATION DEVELOPMENT

Fujaba does not allow UML 2.0 based application development, yet. It is missing the capability of modeling component based systems, a feature, which is nowadays of major importance. Of course, it is possible to add components and all other missing modeling aspects to Fujaba's meta model, but this is related to huge implementation efforts. Obviously, this reduces or even circumvents Fujaba's usage in industry cooperation's, as industry mostly uses and arroges UML 2.0 conform models, respectively.

Story Diagrams can only be used with Fujaba models and within these models only for objects. But using them for other modeling artifacts providing a similar instantiation concept, i.e. components, would be a great benefit. However, it is impossible to use the modeling capabilities of different MDA / CASE tools and then use Fujaba to extend these capability by means of Story Diagrams. Following, if a developer wants to use Story Diagrams, she/he is currently locked in Fujaba.

4. DISCUSSION

The sketched workflow of Section 2.3.1 is not quite optimal. Two different platforms and meta meta models, Fujaba and EMF, have to be used. Fujaba does only support Class Diagrams of UML 2.0, which are even not completely compatible to UML 2.0. The import of EMF models into Fujaba and the code generation of Fujaba do not work perfectly, yet. Necessary configuration files cannot be created automatically at all and several manual steps are required. Furthermore, TGGs and Story Diagrams are not available as EMF models, which would allow model transformations on these models using the explained transformation system. The main cause for these issues is the gap between the meta models that must be bridged somehow.

To overcome these problems, the tool chain should be limited to a single platform. Because EMF is already widely in use, it is the canonical choice. However, this requires the migration or redevelopment of editors for TGGs and Story Diagrams as well as a code generation from Story Diagrams. Although Fujaba4Eclipse already tried to integrate Fujaba within Eclipse, the migration did only change the user interface but the underlying technology is still incompatible to EMF.

Therefore, reusing established technologies of the Eclipse world must leverage the integration of Fujaba and Eclipse. Reusing EMFs widely used meta meta model would make Fujaba compatible to other EMF based tools and would also reduce maintenance effort. Such tools could also be reused and do not need to be redeveloped for Fujaba anymore.

This also opens another possibility for the generation of Story Diagrams from TGG rules as explained in Section 2.3.2. Up to now, the generation of Story Diagrams from TGG rules is hard coded. This complicates the subsequent development of the transformation algorithm because a large part of the algorithm is contained in the operational transformation rules. If the transformation algorithm is modified, the generation algorithm for the Story Diagrams has to be changed, as well. If TGG and Story Diagrams were also EMF models, the transformation algorithm itself could be used to create the Story Diagrams. Only a set of transformation rules has to be provided that transform TGGs into Story Diagrams. To modify the transformation algorithm, only this set of transformation rules has to be adapted.

To achieve the described goal, we are currently working on the development of a Story Diagram editor based on GMF. Because Story Diagrams are quite similar to UML Activity Diagrams, we simply extend the UML 2.0 meta model provided by the UML 2.0 plugin by reusing intersecting parts. Subsequently, a code generator for Story Diagrams must be developed because a Story Diagram's semantic is expressed in terms of the resulting code. EMFs code generation mechanism can be reused to generate the main structure of the code but a tailored code generation is required to generate code from the behavior modeled by means of Story Diagrams.

A drawback of the sketched solution is that story-driven modeling is then limited to Ecore models. Although, Story Diagrams and their associated Class Diagrams are based on UML 2.0, these Class Diagrams have to be converted to Ecore models in order to use EMFs code generation. This is a restriction because Ecore provides only a subset of the constructs supported by MOF or Fujaba. A code generator that can generate Java code directly from the UML 2.0 model without a prior conversion to Ecore can solve this problem.

5. CONCLUSION

In this paper, we have presented two major issues that appeared in our recent work. First, the future of meta-modeling seems to be hold by EMF and second, Fujaba’s Story Diagram and TGG features are important for engineering quality software. We have experienced and explained that Fujaba provides only poor compatibility to EMF and therefore is going to loose track of further developments in MDD. To overcome this, we propose to open Fujaba towards EMF and benefit from EMFs success on the one hand and enabling Fujaba’s key technologies on the other hand, such as Story Diagrams and TGGs for EMF based development.

6. REFERENCES


