EMF Code Generation with Fujaba

Leif Geiger
Universität Kassel
Wilhelmshöher Allee 73
34121 Kassel
leif.geiger@uni-kassel.de

Thomas Buchmann
Universität Bayreuth
Universitätsstr. 30
95447 Bayreuth
thomas.buchmann@uni-bayreuth.de

Alexander Dotor
Universität Bayreuth
Universitätsstr. 30
95447 Bayreuth
alexander.dotor@uni-bayreuth.de

ABSTRACT

Fujaba is a powerful tool for model driven development. But when it comes down to the development of graphical user interfaces, developers are still in need of massive manual coding. On the other side, GMF provides a way of generating graphical user interfaces, but it is tightly coupled to an underlying EMF model. In this paper we show a way how to extend Fujaba's code generation to preserve most of Fujaba's modeling and code generation benefits and map the Fujaba model onto a EMF specification.

1. INTRODUCTION

In its current state Fujaba generates standard Java code which can be easily integrated into most Java applications. The eclipse project offers a platform for editors and similar tools. With GEF and GMF [6] eclipse offers frameworks which simplify the construction of visual editors. To use these frameworks the underlying meta model has to be compatible to the EMF code style. It would be nice to be able to combine these two powerful tools: Fujaba and eclipse / GMF. Then the meta model and the transformations made on it can be done with Fujaba and the construction of the visual editor, menus, toolbars etc. can be built using eclipse / GMF. This work now introduces an EMF code generator for Fujaba.

2. COMPARING THE FUJABA META MODEL, ECORE AND EMOF

To be able to generate EMF compatible code from models specified in Fujaba it is necessary to define a mapping from the concepts used in Fujaba onto those used in EMF. In other words it is necessary to map the Fujaba metamodel onto Ecore. By comparing both meta models it is possible to identify which features of Fujaba can be mapped onto Ecore at all and which features will be lost during the generation process.

We compare the Fujaba metamodel as it is implemented in Fujaba 5 with the Ecore model as it is defined in EMF 2.3.1 [4]. As the EMF introductions says "There are small, mostly naming differences between Ecore and EMOF" [5], we compare both meta models with EMOF 2.0, too [11]. Table 1 shows the result of the comparison. The first column names the compared feature while the following columns contain either *yes* or *no* depending on whether the feature is available in the model or not accordingly.

Feature	Fujaba 5	Ecore 2.3.1	EMOF 2.0
Classes			
abstract	yes	yes	yes
interface	yes	yes	no
reference	yes	yes	yes
Attributes			
read-only	no	yes	yes
final	yes	no	no
static	yes	no	no
transient	yes	yes	no
default value	yes	yes	yes
Methods			
public	yes	yes	yes
protected	yes	no	no
private	yes	no	no
static	yes	no	no
exceptions	yes	yes	yes
Associations			
1:1	yes	yes	yes
1:n	yes	yes	yes
n:n	yes	yes	yes
unidirectional	yes	yes	yes
ordered	yes	yes	yes
int. qualified	yes	yes	no
ext. qualified	yes	no	no
Behavior			
Behav. model	yes	no	no

Table 1: Comparison of Fujaba metamodel and Ecore

Fujaba provides the most features followed by Ecore and EMOF at last which means that Fujaba is far more expressive than Ecore or EMOF. Table 1 highlights in bold letters the features of the Fujaba meta model that are lost when it is mapped onto Ecore. The following constraints must be made to be compatible to Ecore:

1. no concept of *static*, i.e. static final constants.

2. public methods only.
3. no externally qualified associations.
4. no behavioral modeling.

While the first three constrains are acceptable the 4th one is not as it renders one of the key features of Fujaba useless. So the code generation for EMF must preserve the behavioral model by generating the java code for it separately.

The comparison to EMOF shows that its features are a subset of Ecore. This means a mapping from Fujaba to EMOF must fulfill all constraints for Ecore and three more: no interfaces, no transient attributes and no qualified associations at all (highlighted in italics in table 1). Additionally we want to point out that EMOF is a true subset of Ecore and the statement "There are small, mostly naming differences between Ecore and EMOF" [5] does not hold anymore.

3. EMF CODE GENERATION

For generating EMF compatible java code from Fujaba models there exist two different approaches:

1. Generating EMF compatible java code directly from Fujaba models
2. Generating a Ecore model file and use the EMF code generator to generate the java code

The first approach has the benefit that the Fujaba code generation does not depend on other code generators to generate executable code and that the code executing the graph transformations can be easily integrated into the generated classes. As drawback the code generation has to be changed every time the EMF code generation changes. That was the reason (and because it meant much less work) why we decided to implement the second approach. That keeps the maintainance effort low even if the EMF specification changes but on the other hand we need to inject the code for the graph transformations into the code generated by the EMF code generator. Fortunately, the EMF code generator offers a way that makes this very easy.

Figure 1 depicts the various artifacts used during the code generation process as well as their dependencies. The code generation for EMF can be separated in two distinct parts concerning the structural (color on diagram: yellow) and the behavioral part (color in diagram: orange) of the Fujaba model. The first part maps the structural elements of the model (i.e. classes and associations) onto equivalent elements of the Ecore model. The second part generates immediately java code for the behavioral part of the model (i.e. story diagrams). To generate a fully executable java model the EMF code generation must be used to generate java code for the structural part of the model. The EMF code generation is able to merge the already generated code for behavioral part with the newly generated structural code.

We used Fujaba's template based code generation CodeGen2 [8] to generate the needed artifacts. CodeGen2 supports so called `CodeStyles`. That means that the developer can mark

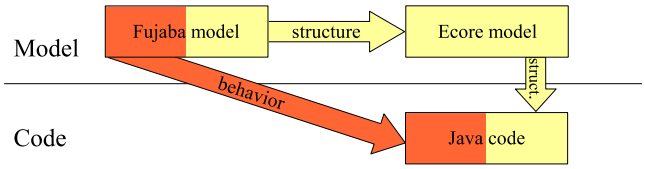

Figure 1: Different levels of code generation

model elements with `CodeStyle` tags. It is now possible to use a different set of templates for e.g. a EMF `CodeStyle` tag. We wrote such templates that are able to generate an XMI file representing the Ecore model from the model elements marked as EMF. Additionally we changed the java code generation so that it generates EMF implementation classes containing only the code for the graph transformation rules for such model element. So, the generated java code only contains the methods and still lacks the implementation of attributes and associations. This is added by the EMF code generator once it is invoked on the generated XMI file.

The generation of the XMI file is straight forward: Fujaba classes are mapped to EMF classes, methods to EMF operations etc. The generation of the java files requires some more work since the EMF mechanism for creation and deletion of objects and links have to be used.

Object creation in EMF can not be done using the `new` operator, one has to use factories. There is one factory responsible for each package which is able to create objects for all classes of this package. This factory can be found using EMF naming conventions (package name followed by `Factory`. Such object creation is done by the following (simplified) template snippet:

```
#set( $factory = "$utility.upFirstChar
      ($package.Name)Factory" )
$name = "${factory}.eINSTANCE.create${type}()";
```

Object removal is still done calling the `removeYou` method which removes all links of the object to make it collectable for the garbage collection. All other object operations like bound checks or assignments can be done as in the java code generation.

For link queries and manipulations one has to consider that some access methods for EMF associations differ from those normally generated by Fujaba. For to-1 associations nothing changes. So the old templates can be used. But for to-many associations instead of a `iteratorOf<role name>` method which returns a `Iterator`, EMF has a `get<role name>` method which returns the set. All queries and mutations have to be executed on this set. So creating a link results in an `add` operation, deleting in a `remove` operation etc. This can be easily implemented with the templates. Since all to-many associations in EMF are ordered what means they are implemented by lists, one may always specify an integer as range when querying a link. But if that index is not valid an `IndexOutOfBoundsException` is thrown which the Fujaba generated code can not handle. So we added an index check before every check or search operation with a

specified range. Such an index check is generated by the template shown below:

```
fujabaIndex = $range;
JavaSDM.ensure ((fujabaIndex >= 0) &&
  (fujabaIndex < ${name}.get$roleName ().size()));
```

Another operation that might be thrown when manipulating EMF object structures is the `ConcurrentModificationException`. This exception is thrown when a set is changed while iterating through it. This might happen in Fujaba in an forEach activity when a link which is used for searching is then deleted or a new one of the same type is added. The standard generated code uses set classes which do not throw this exception. For EMF code this is not possible. This is why we decided to use a pre-select semantics (cf. [15, 14]) for forEach searches here. We copy the set and iterate through the copy. Changes are performed on the original set. So changes will not affect which objects are visited during the search. This is a slight difference to the semantics normally used in Fujaba but we consider it as a minor drawback.

The last missing element of story diagrams are path expressions. Such expressions are generated as Strings into the code and interpreted at runtime. So we do not have to change the code generation here but adapt the path interpreter. There is an implementation of the path interpreter which uses the so called feature abstraction developed at the university of Kassel for link searches. The feature abstraction defines an abstract way to access classes, their methods and their fields. There exists implementations for the java reflection layer, the java debug interface and JMI. We wrote an implementation for EMF. This implementation can be used by the path interpreter and so path expressions do also work for EMF.

EMF offers methods to serialize object structures to XMI. Anyhow using CoObRA [12, 13], the framework for object persistency in Fujaba, has several benefits. In addition to persistency for object structures CoObRA offers generic support for undo-redo operations and multiuser environments including merging of object structures. Fortunately, CoObRA also uses the feature abstraction to access the model. So the implemented EMF feature abstraction can be used here, too. To be able to have full CoObRA support for the generated EMF models, it was neccessary to implement an adapter from EMF property changes to JavaBeans property changes which are used in CoObRA.

4. EXAMPLE

In this section we present a small example to show how the EMF code generation works. The example is derived from the project management domain and is a simple model derived from dynamic task nets [9] and is a simplification of our dynamic task net editor [2]. The model is depicted as UML class diagram in figure 2 and it consists of three classes:

1. `DynamicTaskNet`, which instances represent dynamic task nets

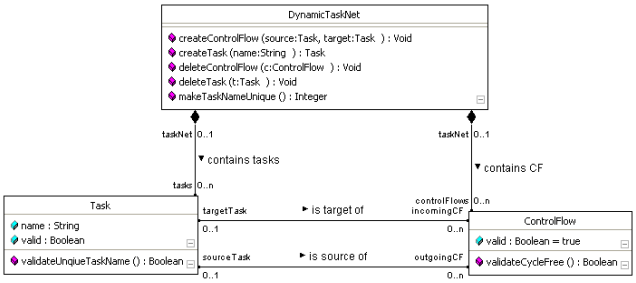

Figure 2: Class diagram of the example model

2. `Task`, which instances represents specific tasks within the net

3. `ControlFlow`, which represents specific relations between tasks, e.g. sooner-later-dependencies

The code generation maps the structural part of the model onto the Ecore metamodel by generating an ecore file. Figure 3 shows a section of the ecore file for our example model: the description of class `Task`. An *eClassifier* with name "Task" is defined and within it several *eStructuralFeatures* (attributes) and *eOperations* (methods). Note the two different types of *eStructuralFeatures*: "EAttribute" and "EReference". The first is used for primitive data types, the second for associations including cardinalities and containment relationship.

The behavioral part is mapped directly onto executable java code. For each class in Fujaba with class name *name* a corresponding class *name*Impl is generated which contains all code generated from story diagrams. In case of the sample class `Task` the generated *TaskImpl* contains code for two methods: *validateUniqueTaskName()* and *removeYou()*. When the EMF code generation is applied onto the ecore file it merges the existing Impl-files with the generated ones. *TaskImpl* contains then additional 17 methods for accessing and changing its attributes and links.

5. CONCLUSION

We have shown that the Ecore model is a subset of the Fujaba language and thus it is possible to generate EMF compatible code from Fujaba models. We have developed such a code generator as a plugin for Fujaba. That code generation not only translates the static part of the model to Ecore but is also able to generate code for story diagrams. Thus the developer can now specify behavior for EMF models with Fujaba. Here the full expressiveness of story diagrams can be used. We have evaluated our tool in a first example case study at the University of Bayreuth and a student project at the University of Kassel. In those projects we were able to specify complex models including behavior with Fujaba and generate EMF code from those models. We used the GMF framework [6] to build a graphical user interface on top of this code. Using these techniques the students in Kassel built a petri net editor and the team in Bayreuth a dynamic task net editor [2]. Note, that thanks to the model based specifications in Fujaba and in GMF, those two projects could be realized with very few lines of hand written code.

```
<eClassifiers xsi:type="ecore:EClass" name="Task" abstract="false" interface="false"
    eSuperTypes="">
   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
       eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" transient="false">
   </eStructuralFeatures>
   <eStructuralFeatures xsi:type="ecore:EAttribute" name="valid"
       eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean" transient="false">
   </eStructuralFeatures>
   <eOperations name="validateUnqiueTaskName"
       eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean" eExceptions="">
   </eOperations>

   <eStructuralFeatures xsi:type="ecore:EReference" name="taskNet" ordered="false" lowerBound="0"
       upperBound="1"eType="#//DynamicTaskNet" transient="false" containment="false"
       eOpposite="#//DynamicTaskNet/tasks">
   </eStructuralFeatures>
   <eStructuralFeatures xsi:type="ecore:EReference" name="outgoingCF" ordered="false" lowerBound="0"
       upperBound="-1"eType="#//ControlFlow" transient="false" containment="false"
       eOpposite="#//ControlFlow/sourceTask">
   </eStructuralFeatures>
   <eStructuralFeatures xsi:type="ecore:EReference" name="incomingCF" ordered="false" lowerBound="0"
       upperBound="-1" eType="#//ControlFlow" transient="false" containment="false"
       eOpposite="#//ControlFlow/targetTask">
   </eStructuralFeatures>
   <eOperations name="removeYou"/>
</eClassifiers>
```

Figure 3: Ecore representation of the class *Task*

6. RELATED WORK

The *TIGER* project [7] is based on AGG, an approach to graph transformations based on category theory. It is dedicated to the automatic generation of GEF editors using graph transformations, in contrast to our work, which uses GMF for editor generation. Therefore we may benefit from further developments for GMF, i.e. design tools for graphical elements and sophisticated wizards for the generation process. Furthermore our editors are extensible like any other GMF-editor. The Tiger EMF Transformation Project [1] adds support for graph transformations on EMF models to TIGER. Here actions on the EMF model can be modeled using graph transformations. But one actions always is one graph transformation rule. More elaborate features as control flow are not yet supported.

The DiaMeta tool [10] is an freehand editor generator based on graph grammars. The grammars are specified against MOF 2.0 meta models specified in MOFLON [3]. Since MOFLON is the MOF 2.0 plugin for the Fujaba Tool Suite, the normal Fujaba graph transformations can also be specified for MOFLON meta models. But the code generated by MOFLON is not compatible to EMF, so the generated applications do not easily integrate into eclipse and can not benefit from all the tools and libraries already available for EMF / eclipse.

7. FUTURE WORK

Following issues will be addressed in the future to improve the Fujaba code generation for EMF: First the mapping of classes with *reference* stereotype onto ecore by using EMFs *EDataType*. Second we are looking for a way to map constants onto ecore. And third we still have to implement support for the new qualified associations of EMF 3.3.

8. REFERENCES

[1] E. Biermann, K. Ehrig, C. Köhler, G. Kuhns, G. Taentzer, and E. Weiss. Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In *MoDELS'06*, 2006.

[2] T. Buchmann, A. Dotor, and B. Westfechtel. Model driven development of graphical tools: Fujaba meets gmf. In *Proceedings of the 2nd International Conference on Software and Data Technologies (ICSOFT 2007)*, pages 425–430. INSTICC, jul 2007.

[3] T. R. A. S. C. Amelunxen, A. Königs. MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In *in: A. Rensink, J. Warmer (eds.), Model Driven Architecture - Foundations and Applications: Second European Conference, Heidelberg: Springer Verlag, 2006; Lecture Notes in Computer Science (LNCS), Vol. 4066, Springer Verlag, 361–375*, 2006.

[4] Eclipse Foundation. *The Eclipse Modeling Framework (EMF) Overview*, 2005. `http://dev.eclipse.org/-viewcvs/indextools.cgi/org.eclipse.emf/doc/-org.eclipse.emf.doc/references/overview/-EMF.html` – last visited: 03/08/2007.

[5] Eclipse Foundation. *The Eclipse Modeling Framework (EMF) Overview*, 2005. `http://dev.eclipse.org/-viewcvs/indextools.cgi/org.eclipse.emf/doc/-org.eclipse.emf.doc/references/overview/-EMF.html` – last visited: 03/08/2007.

[6] Eclipse Foundation. *Graphical Modeling Framework*, 2007. `http://www.eclipse.org/gmf/` – last visited: 30/08/2007.

[7] K. Ehrig, C. Ermel, S. Hänsgen, and G. Taentzer. Generation of visual editors as eclipse plug-ins. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 134–143, New York, NY, USA, 2005. ACM Press.

[8] L. Geiger, C. Schneider, and C. Record. Template- and modelbased code generation for MDA-tools. In *3rd International Fujaba Days*, Paderborn, Germany, 2005.

[9] P. Heimann, C.-A. Krapp, and B. Westfechtel. Graph-based software process management. *International Journal of Software Engineering and Knowledge Management*, 7(4):431–455, 1997.

[10] M. Minas. Generating meta-model-based freehand editors. In *Electronic Communications of the EASST, Proc. of 3rd International Workshop on Graph Based Tools (GraBaTs'06), Natal (Brazil), September 21-22, 2006, Satellite event of the 3rd International Conference on Graph Transformation*, 2006.

[11] Object Management Group. *Meta Object Facility (MOF) Core Specification 2.0*, 2006. `http://www.omg.org/`.

[12] C. Schneider. CASE Tool Unterstützung für die Delta-basierte Replikation und Versionierung komplexer Objektstrukturen (Diploma Thesis, german), 2003.

[13] C. Schneider, A. Zündorf, and J. Niere. CoObRA - a small step for development tools to collaborative environments. In *Workshop on Directions in Software Engineering Environments; 26th international conference on software engineering*. ICSE 2004, Scotland, 2004.

[14] M. Tichy, M. Meyer, and H. Giese. On Semantic Issues in Story Diagrams. In *Fujaba Days 2006*, 2006.

[15] A. Zündorf. Rigorous Object Oriented Software Development, 2001.