Integrating Fujaba and the Eclipse Modeling Framework

Jendrik Johannes
TU Dresden
Fakultät Informatik
Institut für Software- und Multimediatechnologie
D-01062 Dresden
jendrik.johannes @mailbox.tu-dresden.de

Ilie Savga
TU Dresden
Fakultät Informatik
Institut für Software- und Multimediatechnologie
D-01062 Dresden
ilie.savga @tu-dresden.de

Tobias Haupt
TU Dresden
Fakultät Informatik
Institut für Software- und Multimediatechnologie
D-01062 Dresden
s0500251 @inf.tu-dresden.de

ABSTRACT

With Fujaba4Eclipse the Fujaba Tool Suite [18] is intergrated into the Eclipse Platform [5, 13]. Through this integration, Fujaba benefits from the stable and well-documented Eclipse infrastructure and from re-use of some Eclipse tools, such as the Graphical Editing Framework (GEF) [16]. Still, there are other powerful Eclipse technologies, which could be enabled in Fujaba. We show, how this can be done by the Eclipse Modeling Framework (EMF) [11, 3]. More exactly, we demonstrate, how the EMF's metamodel Ecore is used to define EMF-complient Fujaba metamodels. The benefit is twofold. First, Fujaba technologies (e.g., pattern recognition) can be applied to EMF models directly in Eclipse. Second, existing EMF-based tools and applications become applicable to Fujaba models.

1. INTRODUCTION

By integrating Eclipse and Fujaba, both platforms can profit from each others technologies. Fujaba4Eclipse is a first step of that integration. It relies on the Eclipse plugin mechanism and on the Eclipse Graphical Editing Framework (GEF). Still, it uses its own representation of modeling concepts, which limits the interoperability with other Eclipse based modeling tools.

To alleviate the problem of interoperability, we propose to improve modeling (and metamodeling) in Fujaba towards standards such as the OMG metamodeling hierarchy (Figure 1). This hierarchy is a foundation for most CASE environments and enables compatibility and exchangeability of models. Such environments can be developed, for instance, using the Eclipse Modeling Framework (EMF), which complies to OMG standards.

"EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model."[11] Models defined in EMF have a common metamodel called *Ecore*, which is essentially a subset of OMG's MOF standard [8]. This, in turn, implies a use of Ecore as a metametamodel (i.e. a model to define metamodels)[1]. In fact, Ecore is defined in terms of itself, too.

The integration with Ecore opens Fujaba tooling to a broader community by allowing distinct Fujaba features, like the *pattern recognition* mechanism or support for *story driven modeling*, to be used with models developed in other Ecore aware applications. On the other hand, Fujaba models can be passed to other applications for further use. Such applications are, for example, EclipseUML [10], Rational Rose [7] or E-Comogen [6], the latter being a software composition tool developed at our institute that employs Ecore models to describe languages.

As a consequence of the integration, Fujaba's modeling can further profit from other standards implemented by the EMF community. Those include, among others, a common UML2 metamodel implementation [15], a common *Ontology Definition Metamodel* implementation [14] and OCL integration [17] (both of the latter are part of the *EMF technologies* subproject [12] which contains more such standard implementations).

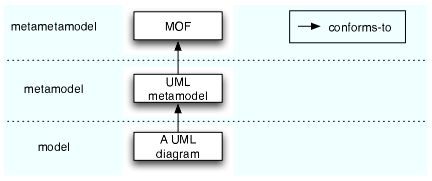


Figure 1: The OMG's metamodeling hierarchy.

To gain compliance to the metamodeling hierarchy, we propose to use Ecore in two ways. First, we require Fujaba metamodels to be defined in Ecore (using Ecore as metametamodel). There is no explicit Fujaba metametamodel at the moment, all implemented metamodels use Fujaba's *metamodel framework*. We show that there does exist an (implicit) metametamodel and how it corresponds to Ecore. Second, we argue that there are correspondences between Ecore as a metamodel and Fujaba metamodels (such as the Fujaba's UML metamodel). Once these correspondences are defined, models (and metamodels) can be exchanged between the two platforms.

[1]One could argue, that, when used to define other models, Ecore serves as a (meta) language, rather than a metamodel. For the sake of simplicity, in this paper we intentionally avoid discussions of relations between modeling languages and the metamodel hierarchy. See [4] for a deep discussion on that matter.

The rest of the paper is organized as follows. Section 2 gives an overview of Ecore. Sections 3 and 4 present our two main ideas of how to approach the integration of Eclipse modeling and Fujaba. After discussing the evaluation of our proposal in Section 5, we shortly present related work in Section 6 and conclude in Section 7.

2. ECORE: OVERVIEW

Ecore, based on the object-oriented paradigm, is used to define models (then it is a metamodel, comparable to the UML metamodel) or metamodels (then it is a metametamodel, comparable to MOF). Many elements found in Ecore are equivalent to elements found in UML class diagrams. Thus, one way to specify Ecore models is by means of UML class diagrams. As mentioned, Ecore is expressed in terms of itself. Figure 2 shows how Ecore's basic subset, called kernel, is defined in UML and figure 3(a) shows a simple Ecore model, also in UML.

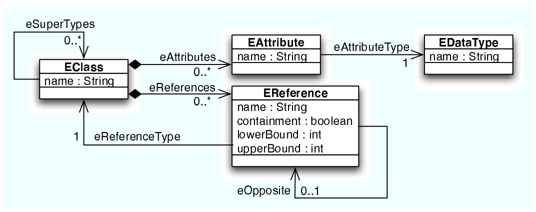
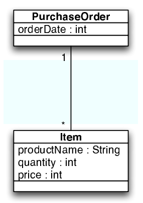


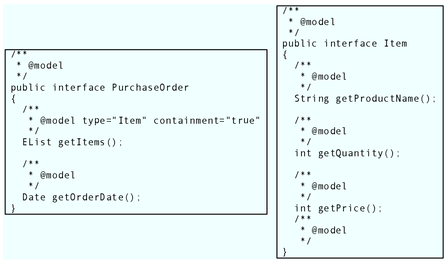
Figure 2: The Ecore kernel [3].

Alternatively to using UML class diagrams, one can define Ecore models by Java interfaces and classes (Figure 3(b)). Such interfaces contain Javadoc annotations to identify model elements and include further constrains which can not be expressed directly in Java. A model expressed in annotated Java can be translated to an equivalent UML representation and back, as both are serializable to XMI (XML Metadata Interchange) [9].

From a model (regardless of how it is expressed), annotated Java code is generated by the code generation module. For the annotated interfaces, an implementation of the interfaces and factory classes are generated automatically. It is still possible to create classes manually or modify generated ones. Generated and manual written code is distinguished by special annotations to avoid overwriting of manual classes in case of a repeated code generation.



(a) UML



```
/**
 * @model
 */
public interface PurchaseOrder
{
  /**
   * @model type="Item" containment="true"
   */
  EList getItems();

  /**
   * @model
   */
  Date getOrderDate();
}
```

```
/**
 * @model
 */
public interface Item
{
  /**
   * @model
   */
  String getProductName();

  /**
   * @model
   */
  int getQuantity();

  /**
   * @model
   */
  int getPrice();
  /**
   * @model
   */
}
```

(b) Java

Figure 3: An Ecore model expressed in UML and Java.

3. ECORE AS METAMETAMODEL: DEFINING FUJABA METAMODELS

In this section, we discuss the use of Ecore as an explicit metametamodel in Fujaba (i.e., how to describe any Fujaba metamodel in terms of Ecore concepts). By this, we obtain an uniform metamodel representation inside of Fujaba as well as between Fujaba and other Ecore aware tools. We consider two scenarios:

- Creation of new Fujaba metamodels. New metamodels for Fujaba can be easily defined by using EMF model editors and their implementation can be generated usign EMF's code generation.
- Re-using existing Fujaba metamodels. To avoid the re-definition of metamodels, we want to extrapolate Ecore models out of available Fujaba metamodel implementations. By carefully annotating the implementation of a Fujaba metamodel (e.g., Fujaba's metamodel for UML class diagrams) with Ecore terms, we extend it to represent a valid Ecore metamodel (Figure 4). We are aware of a possible complexity of this task. Moreover, we expect to discover conceptual flaws in existing metamodels, which may needed to be corrected.

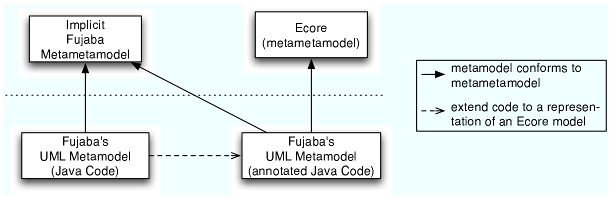

Implicit Fujaba Metametamodel
Ecore (metametamodel)
metamodel conforms to metametamodel
extend code to a representation of an Ecore model
Fujaba's UML Metamodel (Java Code)
Fujaba's UML Metamodel (annotated Java Code)

Figure 4: Fujaba's UML metamodel code extended to conform to Ecore as metametamodel.

4. ECORE AS METAMODEL: RELATING TO FUJABA METAMODELS

Once Ecore is introduced as a metametamodel (Section 3), we can approach an even tighter integration of Ecore into Fujaba. In this section, we go one step down in the metamodeling hierarchy and discuss how Ecore as a metamodel relates to Fujaba metamodels. We also discuss how this relationship can be realized in the Fujaba implementation to achieve our goal of integration.

In Fujaba, interfaces to express concepts of object-oriented languages exist (e.g., *FClass* or *FMethod*). We will call the set of these interfaces the *abstract metamodel*, since it is implemented in different Fujaba metamodels, like Fujaba's UML metamodel (e.g., *UMLClass* implements *FClass*). Ecore models the same concepts of object-oriented languages (e.g., *EClass* and *EOperation*). Table 1 enumerates similar elements found in Fujaba's abstract and UML metamodel and in Ecore.

These similarities suggest to use Ecore as a metamodel in Fujaba as well. This has the advantage that Fujaba can be used to define Ecore models and that it can treat Ecore models defined elsewhere.

Ecore	*Abstract Metamodel*	*Fujaba UML*
EClass	FClass	UMLClass
EAttribute	FAttr	UMLAttr
EReference	FAssoc	UMLAssoc
EOperation	FMethod	UMLMethod
EClassifier	FType	UMLType
EDataType	FBaseType	UMLBaseType
EString	FBaseType.STRING	UMLBaseType.STR
EInteger	FBaseType.INTEGER	UMLBaseType.INT
EBoolean	FBaseType.BOOLEAN	UMLBaseType.BOOL

Table 1: Relations between elements of Ecore and Fujaba metamodels.

To achieve this, a Fujaba metamodel should refer to Ecore comparable to how it refers to the abstract metamodel. However, the difference is that Ecore should be referred by delegation, while the abstract metamodel is referred by inheritance.[2] In the case of UML class diagrams, for instance, every element in the model references a corresponding Ecore element, which content accessable from Fujaba. The interface of the Fujaba UML metamodel stays unchanged. A Fujaba UML model contains Fujaba specific information and functionality not contained in the delegated Ecore model — it *refines* the Ecore model. Such Fujaba specific features are, for example, navigation and access capabilities required by different parts of Fujaba (e.g., graph transformation based on Story Driven Modeling). The Ecore model, on the other hand, can be shared with other applications.

In fact, one can also replace the inheritance relationship between a Fujaba metamodel and the abstract metamodel by a delegation relationship (Figure 5). From that viewpoint, the abstract metamodel becomes redundant, because it represents similar concepts as Ecore and does not add any model information. We can remove the abstract metamodel and let the UML metamodel refine Ecore directly (Figure 6).

The downside is, that all Fujaba code and plugins based on the abstract metamodel, if any, will have to be rewritten to use Ecore interfaces instead. For the beginning one could also keep the Fujaba interfaces as deprecated and base new plugins and metamodels on Ecore with the option of a full integration in the future.

Metamodels not based on the abstract metamodel, like the metamodel for UML state diagrams, can also refine Ecore, although they do not have directly related concepts. A state can, for example, be modeled by a class using the state design pattern. In this case, a model element of type *State* should have a reference to an element of the type *EClass* that represents the corresponding state. Identical approaches could be developed for a story diagram metamodel as needed for Story Driven Modeling.

[2]We chose to use delegation for a cleaner separation of Ecore and Fujaba modeling concepts. In addition, an Ecore model is represented by concrete classes, while the Fujaba's abstract metamodel consists of interfaces only.

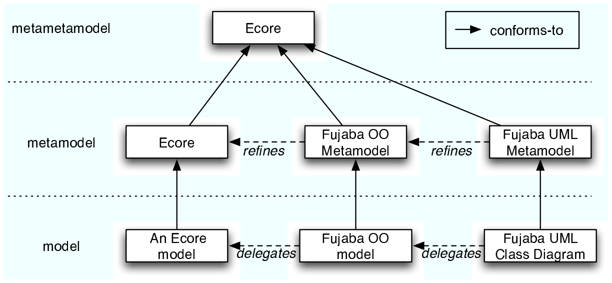


Figure 5: Ecore as a metamodel refined by Fujaba's abstract metamodel.

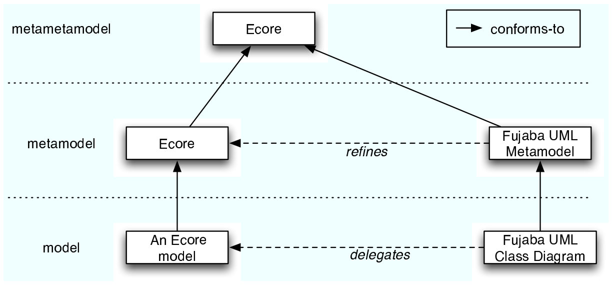


Figure 6: Ecore as a metamodel directly refined by Fujaba's UML metamodel.

5. EVALUATION

While prototypical testing our ideas, we have encountered certain difficulties concerning structural and semantic incompatibility of EMF generated code and Fujaba's core implementation. A big problem poses the fact that the classes implementing metamodels in Fujaba also include functional code concerning, for example, the visual representation of diagrams. To maintain this functionality, manual alterations to generated EMF metamodel code are required.

We also encountered difficulties with the fact that Fujaba misses a clean separation between an interface and an implementation layer, while this is provided in EMF generated code. That makes it hard to identify metamodels and the metametamodel in Fujaba code. As a consequence, annotating Fujaba's metamodel implementations (as proposed in Section 3) is difficult.

Another problem concerns the pattern recognition. The structure of the EMF generated metamodel code does not comply with the structure expected by the engine. Especially, the class naming scheme of EMF's code-generation is by default different from Fujaba's. One needs to overcome these and other minor issues before a full integration of Fujaba and EMF can be realized.

6. RELATED WORK

Amelunxen [1] reports about a tool for editing MOF diagrams, implemented as a Fujaba plug-in. The tool allows to specify MOF metamodels and generate Java metamodels, which are complient to the SUN's Java Metadata Interface (JMI) standard. Despite its relation to MOF, this work does not concern metamodeling in Fujaba's core and does not aim at modeling interoperability.

Amelunxen et al. [2] discuss, how Fujaba can be adapted to serve as a core of a metamodeling framework. Fujaba is regarded as a good foundation for this purpose, due to its modeling features (diagrams and OCL constraints) and functionality, like code generation and a powerful graph rewriting system. Eclipse is proposed as an integration framework by a means of its plugin mechanism and a number of other Eclipse tools like its user interface and editor frameworks. However, they do not consider to integrate EMF to support standardized modeling.

7. CONCLUSIONS

Our vision is that a tight integration of Fujaba with the Eclipse Modeling Framework will bring profits for both communities. To achieve a necessary degree of integration, the metamodeling approaches presented here are applicable. Converging modeling in Fujaba towards EMF and Ecore also means improving the modeling towards standards, since EMF and related Eclipse projects employ OMG standards like MOF and XMI. That increases the quality of Fujaba and the ability to combine it with other tools using the common standards. Furthermore, the Model-Driven Engineering principles in Fujaba can be improved by support of such standards as MOF, XMI, OCL, CWM and SPEM.

8. REFERENCES

[1] C. Amelunxen. A MOF 2.0 Editor as Plugin for FUJABA. In H. Giese, A. Schürr, and A. Zündorf, editors, *Proc. 2nd International Fujaba Days*, volume tr-ri-04-253, pages 43–48. Universität Paderborn, 2004.

[2] C. Amelunxen, A. Königs, T. Rötschke, and A. Schürr. Adapting FUJABA for Building a Meta Modelling Framework. In H. Giese and A. Zündorf, editors, *Proc. 1st International Fujaba Days*, volume tr-ri-04-247, pages 29–34, 10 2003.

[3] F. Budinsky, S. A. Brodsky, and E. Merks. *Eclipse Modeling Framework*. Pearson Education, 2003.

[4] J.-M. Favre. Foundations of meta-pyramids: Languages vs. metamodels – episode ii: Story of thotus the baboon. In J. Bezivin and R. Heckel, editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005.

[5] T. E. Foundation. Eclipse platform technical overview, April 2006.

[6] T. D. S. E. Group. E-CoMogen webpage. http://web.inf.tu-dresden.de/~jh30/work/rewerse/comogen, July 2006.

[7] IBM. Rational Rose homepage. http://www-306.ibm.com/software/awdtools/developer/rose/index.html, July 2006.

[8] Object Management Group. MOF homepage. http://www.omg.org/, July 2006.

[9] Object Management Group. XML metadata interchange (XMI) specification. http://www.omg.org/technology/documents/formal/xmi.htm, July 2006.

[10] Omondo — the live UML company. EclipseUML homepage. http://www.omondo.de/, July 2006.

[11] The Eclipse Foundation. Eclipse modeling framework — EMF. http://www.eclipse.org/emf/, July 2006.

[12] The Eclipse Foundation. Eclipse modeling framework technologies — EMFT. http://www.eclipse.org/emft/, July 2006.

[13] The Eclipse Foundation. Eclipse project. http://www.eclipse.org/, July 2006.

[14] The Eclipse Foundation. EMF based ontology definition metamodel implementation. http://www.eclipse.org/emft/projects/eodm/, July 2006.

[15] The Eclipse Foundation. EMF based UML2 metamodel implementation. http://www.eclipse.org/uml2/, July 2006.

[16] The Eclipse Foundation. Graphical editing framework — GEF. http://www.eclipse.org/gef/, July 2006.

[17] The Eclipse Foundation. OCL integration for EMF. http://www.eclipse.org/emft/projects/ocl/, July 2006.

[18] University of Paderborn — Software Engineering Group. Fujaba tool suite webpage. http://www.fujaba.de/, July 2006.