STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

INTEGRATING ALF EDITOR WITH ECLIPSE UML EDITORS

CODRUŢ-LUCIAN LAZĂR

Abstract. In this paper we present an approach for integrating graphical and textual editors for UML models in Eclipse workbench. In our concrete example, we are integrating an Eclipse Xtext based textual editor for OMG Alf language, with the Eclipse UML2 tree editor for UML models. This integration enables the user to view and edit the operation behavior of UML classes from a big UML model using a textual editor with OMG Alf language.

The integration will also benefit the ComDeValCo framework, where we need to model the functionality of the components created with a graphical editor.

1. Introduction

The Executable Foundational UML (fUML [4]) is a computationally complete and compact subset of UML [1], designed to simplify the creation of executable UML models. The semantics of UML operations can be specified as programs written in fUML.

Currently, there is a standardized concrete textual syntax for a fUML based action language. The concrete syntax is standardized by Object Management Group (OMG) and it is called Action Language for Foundational UML (Alf) [3]. A few other action languages exist, some of them being proprietary. None of these action languages are based strictly on fUML, so our research will focus on using the Alf language.

We also introduced an action language based on fUML [7], with a concrete syntax similar to the concrete syntax of the structured programming languages (e.g. Java, C++). As Alf does not restrict the way the concrete textual constructs are mapped to the fUML structures of elements, we will use our

Received by the editors: March 31, 2011.

2010 *Mathematics Subject Classification.* Software, Programming languages.

1998 *CR Categories and Descriptors.* D.2.2 [**Software**]: SOFTWARE ENGINEERING: Design Tools and Techniques – *Computer-aided software engineering (CASE).*

Key words and phrases. Eclipse, EMF, Xtext, fUML, UML, Alf, Action Language.

previous experience in order to define these mappings and implement a textual editor in Eclipse.

In order to be able to use an Alf textual editor in a real environment, it is needed to integrate the textual editor for Alf language with existing UML graphical editors. We have chosen Eclipse as the modeling environment in which to try to develop our modeling tools. The UML graphical editors are implemented using Eclipse tools: tree editors, or diagram editors based on Eclipse Graphical Modeling Framework (GMF) tool. For the Alf language, we can use the Eclipse Xtext utility [5], on top of which to implement our editor. Xtext offers a simple way to define the textual grammar and generates a textual editor that can serve for us as a starting point.

The main problem is that these Eclipse editors work with resources that cannot be nested, and we need to be able to view and edit only parts of the bigger UML model with our Alf editor.

This paper is a technical paper that will describe how we achieved the integration of the graphical UML editor (tree based, in this example) and the textual Alf editor.

The remainder of the paper is organized as follows: section 2 presents the general context and section 3 presents the research problem. Then, section 4 describes the integration approach we propose. Section 5 presents the existing work related to ours and section 6 gives the conclusions of this paper.

2. Background

Creating executable UML models is difficult, because the UML primitives intended for execution (from the UML Action Semantics package) are too low level, making the process of creating reasonable sized executable UML models close to impossible. In order to speed up the process of writing the behavior, different approaches have been used. One approach was to replace the behavior with opaque expressions expressed in the destination programming language (Java, C++) and add them directly in the UML model. Another approach was to fill in the behavior after code generation, in the special marked places. These types of models will not react well to changes in the target programming languages or frameworks. Also, simulating the execution of these types of models and testing them is not easy.

The fUML standard provides a simplified subset of UML Action Semantics package (abstract syntax) for creating executable UML models. It also simplifies the context to which the actions need to apply. For instance, the structure of the model will consist of packages, classes, properties, operations and associations, while the interfaces and association classes are not included.

Creating fUML models is still a hard thing to do, so it is required to use a concrete textual syntax for this.

A concrete textual syntax is in process of being standardized by OMG - Alf language. With this easy-to-use concrete syntax, the creation of executable UML models will become an easy task. A textual syntax enforces a certain way of constructing models, which means that a lot of elements that need to be created explicitly in the graphical UML Activity Diagram can be implicitly derived from the syntax and created automatically. The control flow between the statements is implicit in structured programming languages. Accessing the parameters and variables is done with a simple name reference in text, while the model will contain all the object flow edges and fork nodes needed to represent them. And the examples can continue.

What is required at this point is to create editors based on this Alf language and to integrate them with the existing UML graphical editors and other tools that process UML models.

The action language is also useful for our framework: ComDeValCo (Framework for Software Component Definition, Validation, and Composition) [10, 9, 6]. We want to use it to model the functionality of the components, to simulate the execution of the models and to test the models [8]. The main advantage is to be able to test our models without having to generate code.

3. Research problem

The research problem is to find a way to integrate the textual editor for the Alf language with other graphical editors for UML models in the Eclipse environment.

We have chosen to integrate our textual editor based on Alf language with the tree based editor for UML from Eclipse. This tree based editor will play the role of the graphical editor. We are attempting to make the integration process in such a way that it does not affect the tree based editor, so that we will be able to integrate our textual editor with any other graphical editor for UML models from Eclipse (UML diagram editors based on Eclipse GMF tool, for instance).

4. Integrating Eclipse editors

Similar to any other editor from the Eclipse workbench, the UML tree editor from Eclipse loads the model inside a Resource. Our Xtext based textual editor also loads the model inside a Resource. Each editor will load and save the contents inside the Resource it is working with (by delegating the load and save actions to the Resource).

The general approach is that the contents (model elements) of each Resource will be saved by that Resource. Contents cannot be shared between Resources. A model element can have only one model element owner or no owner (if it is the root model element). All model elements from a contents tree are contained by one and the same Resource.

The Resources cannot be nested, so we cannot have an editor that works on a Resource that represents a partial model from a different Resource. For instance, suppose the UML model is loaded by the UML graphical editor inside an UML Resource. We want to open a textual editor (which needs a Resource to work with) that will edit only a part of the bigger UML model, more specifically, an Activity.

The Xtext editor's parser will re-parse, discard and refresh the whole (or part) of its model at a frequent rate. It is not feasible to have the textual editor work on the whole UML model and simply not show the parts on which we are not interested (packages, classes, ...), inferring this information behind the scene.

Alf standard recommends to save the textual representation of the operation's behavior (which is an Activity) inside a stereotyped Comment attached to the Activity inside the UML model. So, we need to make the Xtext editor load and save the textual representation from that Comment. And the Xtext parser will create the whole UML model elements (actions, nodes, edges, ...) that represent the actual Activity functionality represented by the textual representation from the Comment.

At the same time, the whole UML Activity that is generated by the textual editor based on the Alf code will need to be placed at runtime inside the big UML model, so that it can be validated. The Activity associated with a Classifier's Operation (as its behavior) needs to reside in the same Classifier. The problem that arises is that the Activity will be created by the Xtext editor inside its own Resource and it will be contained by this Resource, which means that the Activity cannot be contained at the same time by the Classifier from the big UML model, which is contained by the UML graphical editor's Resource.

We need to make use of a certain feature from EMF which allows us to use containment reference proxies for certain elements, which means that we need to intervene in our Xtext editor and make it work with proxies for the actual elements that it generates from the textual elements and for the elements that we will place in the big UML model, under the Classifier. We will have one real Activity element and one proxy Activity element. And we need to figure out when to create these elements, where to place each one of them, what to do when the Xtext editor is required to save the Resource (or when the

UML graphical editor wants to save its resource), how often should the Xtext re-generated model elements be replaced in the big UML model.

This means that the Xtext editor's Resource will have a transient in-memory Model element as its root element, and this element will nest the actual elements that are contained by the big UML model, in the other Resource. Logically, the model elements generated inside the Activity are contained by the big UML graphical editor (which is required in order to properly validate the UML model), and the model elements are to be saved inside the Xtext Resource.

We need to use temporary containment proxies in the big UML model, under the Classifier, and keep the actual elements in the Xtext Resource. This will happen only at runtime, while the editing is in process. When a save action occurs, the Xtext Resource will save the textual representation in the stereotyped Comment and will replace the Activity from the Classifier from a proxy Activity to the actual Activity, thus performing the actual change in the big UML model. At this point, the UML model will contain the new elements generated with the textual editor, which may be explored with the UML editors or used with other tools (code generators, for instance).

5. Related Work

Being able to integrate different editors is a natural requirement for every development environment. However, what we present in this paper is strictly dependent on the Eclipse IDE.

There have been Eclipse projects where the textual and graphical editors have been mixed. TEF [2] provides a textual editor integrated as a popup inside the graphical editor, using custom concrete and abstract syntaxes.

The Xtext framework also has two integration examples. In one example the graphical and textual editors both work on the whole model, which is not an acceptable approach, as the UML models may become large models and the Xtext parser is refreshing the whole model quite frequently. And in the other example the textual editor is integrated as a popup inside the graphical editor. The popup is still in an experimental phase and it doesn't work and cannot be used as a fully fledged textual editor.

6. Conclusions and Further Work

Creating reasonable sized executable UML models is hard, because the UML primitives from the UML Action Semantics package are too low level. The fUML standard provides a simplified subset of UML Action Semantics package and it also simplifies the context to which the actions need to be

applied. However, an easy-to-use concrete textual syntax is still needed in order to speed up the process of creating executable models.

In this article, we have presented a way to integrate a textual action language editor with the existing UML graphical editors in the Eclipse IDE. The action language for which we want to create a textual editor is Alf and it is standardized by OMG. Alf is based on fUML and it follows the structured programming principles, with a concrete textual syntax similar to the most popular object-oriented programming languages (e.g. Java, C++).

In the future, we plan to create a fully functional editor based on Alf action language and also to integrate it into our ComDeValCo framework. We already have a prototype textual editor that uses fUML for the abstract syntax and is implemented as an Eclipse plug-in using Xtext.

References

1. *Uml superstructure specification*, 2007.
2. *Textual editing framework (tef)*, 2008.
3. *Concrete syntax for uml action language (action language for foundational uml - alf)*, 2010.
4. *Semantics of a foundational subset for executable uml models (fuml)*, 2011.
5. Eclipse Foundation, *Xtext - language development framework*, 2011.
6. C.-L. Lazăr and I. Lazăr, *On Simplifying the Construction of Executable UML Structured Activities*, Studia Universitatis Babeş-Bolyai, Informatica **LIII** (2008), no. 2, 147–160.
7. C.-L. Lazăr, I. Lazăr, B. Pârv, S. Motogna, and I.-G. Czibula, *Using a fUML Action Language to construct UML models*, 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (2009), 93–101.
8. ———, *Tool Support for fUML Models*, International Journal of Computers, Communications & Control (2010), no. 2, 775–782.
9. I. Lazăr, B. Pârv, S. Motogna, I.-G. Czibula, and C.-L. Lazăr, *An Agile MDA Approach for Executable UML Structured Activities*, Studia Universitatis Babeş-Bolyai, Informatica **LII** (2007), no. 2, 101–114.
10. B. Pârv, S. Motogna, I. Lazăr, I.-G. Czibula, and C.-L. Lazăr, *ComDeValCo - a Framework for Software Component Definition, Validation, and Composition*, Studia Universitatis Babeş-Bolyai, Informatica **LII** (2007), no. 2, 59–68.

Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: clazar@cs.ubbcluj.ro