UML Toolchain

Using Fujaba and UML Lab in a toolchain

Andreas Koch, Albert Zündorf
Kassel University, Software Engineering,
Department of Computer Science and Electrical Engineering,
Wilhelmshöher Allee 73,
34121 Kassel, Germany
[andreas.koch | zuendorf]@cs.uni-kassel.de
http://www.se.eecs.uni-kassel.de/

ABSTRACT

Every CASE-Tool has its strengths and weaknesses. Of course, Fujaba is not apart from this rule. Thus, why not combine the strengths of Fujaba with another application in a toolchain.

This paper introduces a toolchain covering Fujaba and UML Lab. Traditionally a toolchain is based on the use of an im-/export functionality to transfer data between the different tools by persisting this data with a common file format. As the requirements of the introduced toolchain cannot be fulfilled by this mechanism, a synchronization of models based on the Fujaba respectively UML Lab metamodel is implemented. One requirement is to ensure that changing anything in the model of one tool has an immediate impact on the related model in the other tool. Therefor the model synchronization handles every change separately by analyzing change event objects resulting from each model modification and ensures a (preferably) immediate handling of changes.

1. INTRODUCTION

Each software application is developed in terms of a specific purpose. In the context of this purpose the application can (or at least it should) provide any necessary functionality. However, the requirements of an user often exceed these functionalities or the user prefers a similar approach offered by another application. Either way, one application alone cannot satisfy the requirements of this user. Therefore the creation of a toolchain containing all necessary applications to complete the desired task is recommendable.

The concept of a toolchain can be interpreted differently. It can describe an unidirectional order of tools (see figure 1(a)) as well as a bidirectionally traversable chain (see figure 1(b)) or any other set of tools. Depending on the structure creating a new toolchain is affected by different challenges, but a main task always refers to the data exchange between the included tools. This problem can be separated into two connected sub tasks. The first task addresses the general way how data, created in one tool, can be transferred into the next tool (in the chain) to proceed working. The second task deals with handling subsequent changes in an inbound document. This means to define how, if at all, the modification of data in one tool can be applied to existing data in another tool.

A traditional approach to provide the data exchange is the use of an im-/export functionality: at first, all necessary data is persisted by exporting it with tool A. Afterwards tool B imports this data and the user can continue to work. This approach lacks, amongst other things, in the necessity of a (manual) intervention each time a data exchange is performed. Additionally, with a common and standardized data format, there can be difficulties persisting any tool specific data (e.g. positioning of GUI elements) or preserving this data after a future export by another tool. This lost data is usually not relevant in an unidirectional toolchain, but impedes the usage of a bidirectional traversable toolchain. There, every step back would result in the, at worst manual, restoration of all lost data.

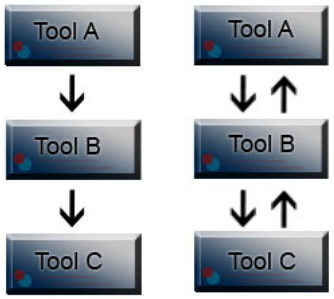

Figure 1: Structure of different toolchains

The introduced *UML Toolchain* with the CASE-Tools Fujaba, to be more precise Fujaba4eclipse, and UML Lab is targeted on a tight integration of their functionalities and bidirectional traversable. Therefore it is necessary that changes, for example on a model in Fujaba, are immediate applied to the equivalent model in UML Lab. Using an im-/export functionality does not fulfill these requirements in several points. Besides the manual interaction to im-/export any data, this approach does not rely on altering existing data, but storing and restoring (means deleting and recreating) of all available data. To address these problems an automatical synchronization of equivalent parts of the metamodels in both tools has been implemented. Additionally, this synchronization depends on the usage of event objects triggered by each modification of a model. This means: every change is handled separately.

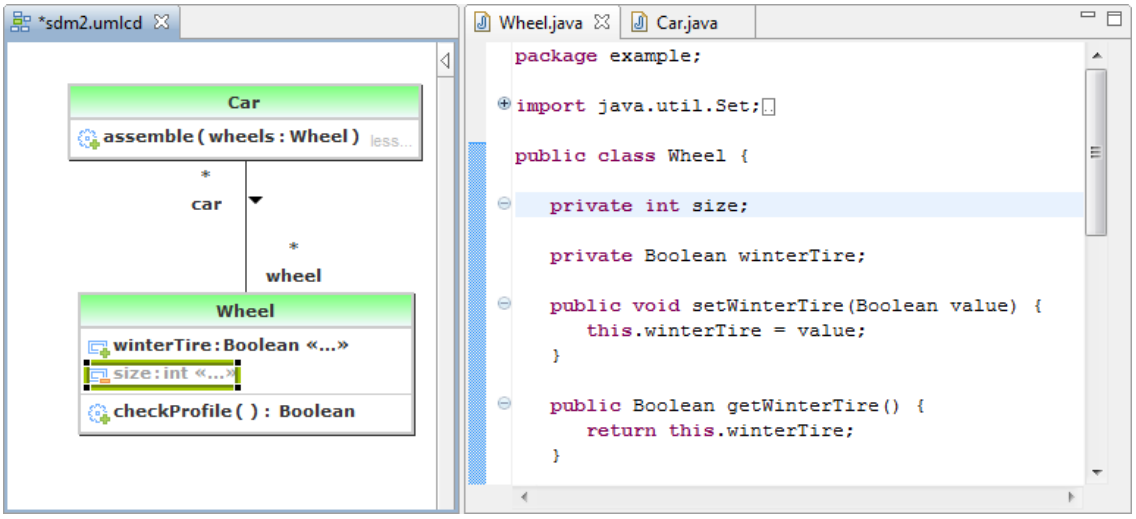

Figure 2: screenshot of UML Lab showing source code and the resulting class diagram

2. UML LAB

UML Lab is a modeling IDE developed by the Yatta Solutions GmbH[7]. As it is based on the eclipse platform [2], the implementation of the UML specification 2.x[6] for the eclipse platform is used as metamodel. The influence of Fujaba during the development of UML Lab is reflected in the effort to combine code generation and reverse engineering. Based on the concept "From UML to Java and back again" a language independent synchronization of source code and model was developed and integrated into UML Lab. This Round-Trip-Engineering^{NG}[1] uses textual templates for generation as well as reverse engineering to enable the parallel work in source code and diagram as shown in figure 2.

3. DEFINING THE TOOLCHAIN

To define the core requirements of the introduced toolchain two different groups of users have to be analyzed.

The first group consists of users, that are familiar with Fujaba. They usually work with UML Lab only, if it provides a noticeable advantage. The Round-Trip-Engineering^{NG}, accessible through the toolchain, enables the synchronization of generated source code with the Fujaba class diagram and therefore falls into this category. Additionally, as UML Lab uses the implementation of a current version of the UML metamodel, the toolchain provides access to new features of the UML specification without the necessity to change the Fujaba metamodel.

The second group of users are familiar with UML Lab. These consider using Fujaba only where UML Lab does not provide the appropriate tools. Especially the concept of *Story Driven Modeling*(SDM)[3] [8] in Fujaba has to be mentioned in this context. This includes the creation/editing of story diagrams, which can (now) be based on Fujaba and UML Lab models. An additional use case is to integrate source code for these modeled story diagrams into UML Labs generated source code. An explicit request apart from this is, that legacy projects from Fujaba shall be usable (and editable) in UML Lab.

To satisfy both groups needs, the effort to use the other tool has to be kept as low as possible. In addition, as one wants to use his tool-of-choice, a class diagram has to be editable with both tools.

4. IMPLEMENTING THE TOOLCHAIN

To discuss the details of the implementation, an example usage of the toolchain is given:

1. Reverse Engineering of existing source code (UML Lab)
2. Extend one class with an additional method (Fujaba or UML Lab)
3. Create a story diagram for this method (Fujaba)
4. Generate code (UML Lab (with Fujaba))

After one or more existing classes are reverse engineered with UML Lab (step 1), the resulting class diagram can be edited with both tools - depending on one's preferences. In step 3 switching to Fujaba is necessary to create the story diagram for a formerly created method. The last step, the code generation, needs more explanation. As both tools provide an own code generation, each tool can solely generate executable code. However, UML Lab does not generate code for story diagrams and the code generation of Fujaba is not automatically linked to the previously reverse engineered classes. Accordingly, the combined generation with both tools is recommended: first of all the source code for story diagrams is generated with Fujaba; afterwards it is passed to UML Lab and integrated into its generated source code. Therefore switching back to UML Lab is required again.

As steps like these are executed regularly and in an undefined order, one has to switch between Fujaba and UML Lab frequently. Two requirements result from this conclusion. First of all, as much as possible should be done without any explicit user interaction. This includes the automatical synchronization of model changes or the combined code generation. With the automatic model synchronization one has not to take care about the currently active tool, because both models (Fujaba and UML Lab) are always consistent with each other. Secondly, features of the other tool have to

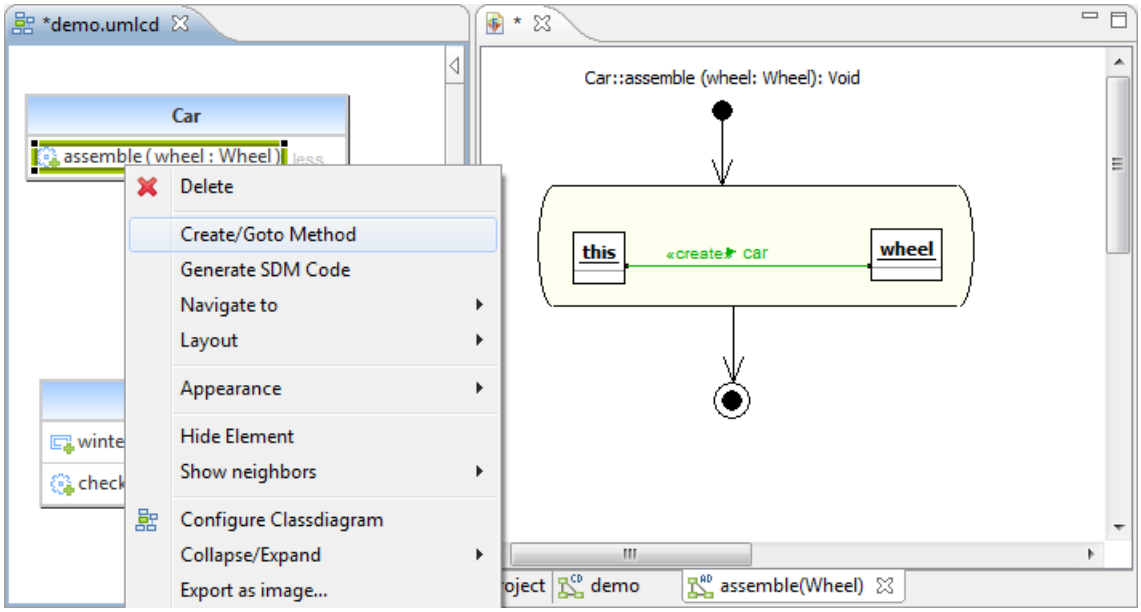

Figure 3: Usage of the story diagram editor in the UML Lab class diagram editor

be accessible directly. A simple example: to create and edit story diagrams while using the class diagram editor of UML Lab, its context menu is extended with a new menu entry to open the story diagram editor and switch to it automatically (figure 3).

All those features are provided externally and added by using plugin mechanisms; on the one hand to avoid dependencies between the tools, on the other hand because they shall not (Fujaba) or cannot (UML Lab) be modified. For this reasons an adapter is used to connect Fujaba and UML Lab. This adapter takes care of the model synchronization and the extension of the tools; for example the additional entries in context menus.

5. MODEL SYNCHRONIZATION

The model synchronization is not discussed in detail in this paper, but we will give a brief overview of its general idea and structure. For the implementation details see [5].

To perform a valid synchronization of two models their metamodels need to be examined and similar parts identified. By analyzing the metamodels of Fujaba and UML Lab one common part concerning class diagrams can be found. Accordingly the synchronization is restricted to this common elements. These elements are afterwards used to find suitable mappings; for example *FMethod* (Fujaba metamodel) can be mapped to *Operation* (UML 2.x metamodel). Based on this mappings different handlers are implemented, each responsible for one field of one metamodel element pair; for example one handler synchronizes the return type of *FMethod* with *Operation* and vice versa.

Imagine the following scenario: there are two already synchronized models in Fujaba and UML Lab that contain a method (*assemble()*) associated with a story diagram. This method is extended by a new parameter (*wheel:Wheel*) using the UML Lab class diagram editor. After switching back to its story diagram, this new parameter should be immediately added and usable. The result is shown in figure 3 with the class diagram editor on the left and the story diagram editor on the right side.

As both metamodels provide an implementation of the observer/listener pattern, these are used to synchronize every change separately; for example events are fired after the creation of the new parameter, the change of its name or adding it to a method. These events are analyzed and delegated to the responsible handlers. The only information that cannot be extracted from these events is the target object the change should be applied on. For this purpose the adapter manages all known object mappings and makes them available to the handlers. These mappings are generated every time a new object is created as well as after loading and scanning of associated models. Finally the handler combines the informations from the event with the mapped object to synchronize the change.

Avoiding inconsistent models was a main challenge during the creation of the synchronization mechanism; especially asymmetric structures were problematic. For example *Property* needs to be mapped to *FAttr* or *FRole* depending on attribute values of a *Property* object. Therefore every time a *Property* object is changed the mapped target object has to be evaluated correctly. As a result some events have to be collected and not synchronized until a consistent result can be ensured.

6. SUMMARY AND FUTURE WORK

UML Toolchain can be used to work in parallel with Fujaba and UML Lab. This means one can create/edit class and story diagrams, generate code or reverse engineer existing code without a manual switch between the tools. Therefore every model change is automatically synchronized i.e. immediately applied to the related model in the other tool.

For the future work two topics can be examined separately. The first deals with the ideas to expand the toolchain; not by integration another tool, but by extending the synchronized part of the metamodels. This includes for ex-

ample the synchronization of story diagram when they are integrated into UML Lab. Additionally, the usability can be improved in some points for example to administrate the synchronized models. This includes a temporary disconnect of models or a general overview to show all synchronized (loaded or not loaded) models.

The second issue is aimed onto the model synchronization. As the example uses a programmatically implementation, the framework provides interfaces to easily attach other mechanisms. For example the integration of the triple interactive graph grammar (TiG) interpreter[4] is planned in the near future.

7. REFERENCES

[1] M. Bork, L. Geiger, C. Schneider, and A. Zündorf. Towards roundtrip engineering - a template-based reverse engineering approach. In I. Schieferdecker and A. Hartman, editors, *ECMDA-FA*, volume 5095 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008. http://dblp.uni-trier.de/db/conf/ecmdafa/ecmdafa2008.htmlBorkGSZ08.

[2] Eclipse platform. http://www.eclipse.org/eclipse/.

[3] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story diagrams: A new graph rewrite language based on the unified modeling language. In *Proc. of the* 6^{th} *International Workshop on Theory and Application of Graph Transformation*. Paderborn, Germany, 1998.

[4] B. Grusie. Ein objektorientierter, interaktiver Triple Graph Grammatik Interpreter. Master's thesis, University of Kassel, Kassel, Germany, 2010.

[5] A. Koch. Echtzeit Synchronisierung von UML-Modellen unterschiedlicher technischer Basis am Beispiel von UML Lab und Fujaba. Master's thesis, University of Kassel, Kassel, Germany, 2010.

[6] Implementation of the UML 2.x metamodel for the Eclipse platform. http://wiki.eclipse.org/MDT-UML2.

[7] Yatta Solutions GmbH. http://www.uml-lab.com/en/uml-lab/.

[8] A. Zündorf. Rigorous object oriented software development. Habilitation Thesis, University of Paderborn, 2001.