Visualizing differences of UML diagrams with Fujaba

[Position Paper]

Jörg Niere
Software Engineering Group
Hölderlinstr. 3
D-57068 Siegen, Germany
joerg.niere@uni-siegen.de

1. MOTIVATION

Software Configuration Management (SCM) is indispensable when developing large software systems. There exists well established tool support such as the popular open source tool CVS or several commercial systems, e.g. Microsoft Visual SourceSafe. Common to those tools is that the management is based on text files and therefore mostly used in the late phases of the software development process, i.e., the implementation phase.

In early phases of a software development process such as the requirements or design phase, engineers use CASE tools and more precise UML modeling tools. Today, CASE tools usually provide a set of different diagrams and use an abstract syntax graph representation, namely a meta-model. Persistently storing an actual syntax graph representing of a set of diagrams in text files is a possible solution, but managing those persistent text files with a textual SCM tool is not feasible [7, 5]. Suppose the situation, that you load a set of diagrams and save the diagrams without changes. If the sequence of diagram items has been changed, e.g. depending on the internally used container classes, the SCM tool will say that the file as changed although the diagrams have not.

CASE tools need their own SCM tool working on the internal abstract syntax graph representation [6] in order to prevent problems such as sequence changes that have no semantical meaning for the diagrams. For example the Rose tool [4] comes along with an SCM tool, which supports calculating differences between two versions of a set of diagrams. The Rose Model Integrator uses the internal unique identifiers to compare diagram items in different versions and represents the differences in a kind of internal abstract syntax tree view. The engineer has to know for example that the class *MethodDecl* is the internal representation of a *method* in a class diagram, or class *RelEnd* represents roles of associations. Especially when there are a large number of differences, the actual context of a certain difference is not always clear to the engineer.

The Fujaba CASE tool also supports versioning of diagrams [8]. Fujaba's approach is to log all edit operations; so-called changed-based versioning [5]. Calculating differences between two versions means to replay the edit operations starting from the common original diagram and using unique identifiers to compare diagram items in the different versions. Fujaba displays difference information about internal syntax graph classes similar to the Rose Model Integrator.

As the previous examples show, the inner tool SCM support is already understood for CASE tools, although the representation of the difference information is inadequate. This position paper sketches our approach of visualizing differences of UML diagrams within the diagrams themselves. We call those diagrams difference diagrams. Although we use Fujaba as tool to display difference diagrams, our approach is independent from a certain internal representation of diagrams, because the calculation algorithm takes two XMI documents as input and does not depend on unique identifiers. Those XMI files may be produced any CASE tool.

2. DIFFERENCE VISUALIZATION

Our approach takes two XMI documents as input for the difference calculation algorithm and the algorithm produces a unified XMI document including difference information. To represent the difference information within the unified XMI document we use the XML extension mechanism. The advantage is that the unified document can also be understood by other CASE tools having a standard XMI import facility where our specific extension will have no effect. Currently the difference calculation algorithm handles class diagrams only and the visualization is under construction.

Figure 1 shows two different class diagrams. Each diagram represents a subset of the internal structure of HTML documents containing advanced features such as forms, lists and combo boxes. Both class diagrams have been produced independently by different development teams during a practical software engineering course at the university. The class diagram on the left hand side contains hard restricted relations between the different elements such as forms only have one combo box element and one list. The class diagram on the right hand side uses the composite design pattern [2] where an actual HTML document can contain a number of forms that can contain a number of lists and combo boxes. In addition, a combo box is subtype of a list, because combo boxes are also lists but showing only the selected item and not the whole list. A design such as on the right hand side in Figure 1 can be found in many graphical user interface libraries, e.g. the Java Swing library. In the following we call the class diagram on the left hand side in Figure 1 sim-

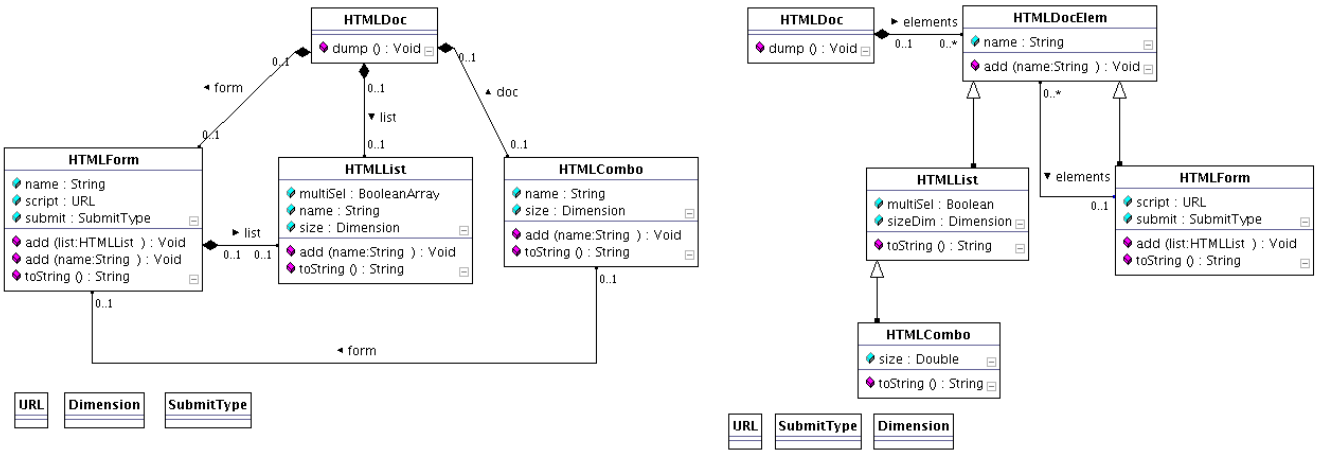

Figure 1: Two class diagram variants of an internal HTML document representation.

ply the left class diagram and the other one the right class diagram.

After three weeks design phase, the different development teams must present and defend their design solutions and all teams have to agree on a common design as basis for further work. In general the common design includes parts from more than one original diagram and the question of how such an operation is supported arises. One solution is to create a unified class diagram containing all classes and relationships in such a way that all elements of both diagrams are contained in the unified diagram. The result is comparable with the union operation presented by Selonen [9]. Consequently you get a unified diagram where the original two diagrams lie next to each other, similar to Figure 1. The similarities in both diagrams such as combo boxes, lists, forms or documents have been completely ignored.

Our difference calculation algorithm tries to identify similar diagram elements automatically, which results in a difference diagram containing all elements of both original diagrams but also glued elements identified as similar. The resulting difference class diagram is shown in Figure 2. In addition, we use Fujaba's internal class diagram layout algorithm. On the one hand layout is an important information, especially when handling large diagrams. On the other hand our approach takes independently developed diagrams with different layout aspects as starting point and thus we currently use standard layout algorithms available in the actual visualization tool and skip the layout information of the diagrams, completely.

Our approach of calculating and visualizing differences bases on the definition of diagram elements. For example, we decided that diagram elements of class diagrams are classes, attribute definitions, method signatures, associations, inheritance relations, notes, etc. Diagram elements consist of attributes and can contain other diagram elements and can also be connected to other diagram elements. Simplified we get something like an internal meta-model for which we can define similarity rules. The difference calculation algorithm applies these rules to the input files after parsing them.

Based on the internal meta-model, we distinguish between three kinds of differences. Most simple are *move differences*, because they indicate a renumbering of elements. For example, a rearrangement of attributes or methods of a class. We visualize move differences by adding a small number button (n-button) to the diagram element. Hence the current implementation automatically sorts attributes and methods by name and all other sequences are ignored, there exists no move difference in Figure 2.

Structural differences indicate that a certain diagram element belongs only to one of the original diagrams, i.e., there exists no similar element in the other diagram. In general, elements contained only in the left class diagram are displayed in red color and elements contained in the right class diagram are displayed in green color. Referring to the screen-shot in Figure 2, the red color becomes light gray and the green color is shown as dark gray.[1] For example, class HTMLDocElem belongs only to the right class diagram and therefore the whole class, i.e., its name and border as well as its attributes and methods, get green color. All other classes have been identified as similar and therefore we visualize them in black color, e.g. HTMLDoc, HTMLForm, HTMLList and HTMLCombo.

The difference calculation algorithm has identified no similar relationships between the classes, i.e., inheritance relationships and associations. Therefore the two elements associations and the two inheritance relations starting from class HTMLDocElem to HTMLForm as well as HTMLList and the inheritance relation between HTMLCombo and HTMLList are shown in green (dark gray) color. All others are shown in red (light gray) color.

Structural differences can also occur within classes as shown in Figure 2. Attributes and methods, which have not been identified as similar are shown in red color if they belong to the left class diagram and shown in green color if they belong to the right class diagram. For example the name attribute

[1] We are working on using dotted line styles and fonts instead of using colors for better readability of gray-scaled screenshots.

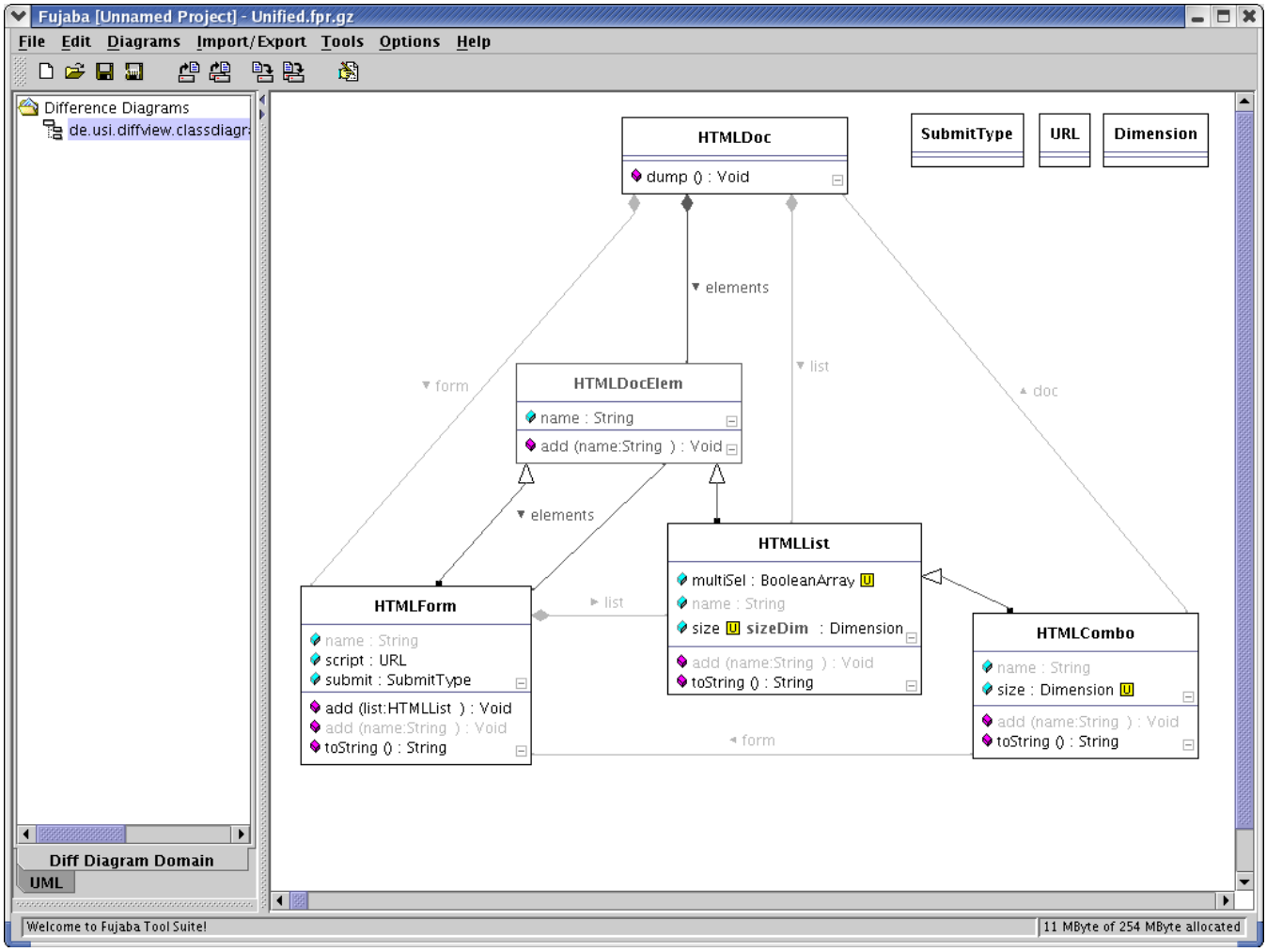

Figure 2: Difference class diagram.

and the method add(name:String) of class HTMLForm belong only to the left class diagram, whereas the attributes script and submit belong to both class diagrams and are shown in black color. A situation where a green or red class contains attributes and/or methods of a different color than the class itself can not occur, because a movement of attributes and methods from one class to another not similar class will not be identified by the algorithm. Such a movement is only detectable with a three-way-difference approach and not by a two-way-difference approach, see [6].

The third kind of differences are so-called *update differences*. Update differences are differences within one diagram element, i.e., attribute value changes of a diagram element. For example, the difference calculation algorithm has identified that two attributes of a class are similar, but the names are not identical. In addition, not identical visibilities, types and initial values of attributes result in update differences as long as the difference calculation algorithm identifies two attributes as similar. Otherwise it is a structural difference.

We visualize update differences by showing a small yellow button (u-button), cf. Figure 2. The u-button has three states. One state in which the value from the left class diagram is shown in red color, one state in which the value from the right class diagram is shown in green color and one state in which both values are shown side-by-side in red and green color, respectively. For example in Figure 2 the type BooleanArray of attribute multiSel in class HTMLList is the value in the left class diagram. In case of the attribute with type Dimension both name values are shown, i.e. size and sizeDim.[2]

3. TECHNICAL REALIZATION

The difference calculation algorithm and the visualization come along as Fujaba plug-ins. The difference calculation algorithm plug-in is coupled in such a way, that the algorithm is started by a menu entry in the tools menu. After selecting two XMI documents to be compared, the algorithm produces an output XMI document including the difference information. The last part of this process is to notify the visualization if required and pass the produced XMI document as parameter. This design allows us to ship also an

[2]The font looks a little bit bold.

independent tool, which can be used outside of Fujaba, e.g. as extension tool to commercial UML tools.

The second plug-in is the visualization plug-in. As we have started with the project we wanted to use the meta-model enhancement facilities [1] to reuse the already existing diagram meta-models for class diagrams, statecharts, activity diagrams, etc. Unfortunately, Fujaba's graphical user interface library, i.e. the FSA library, can not deal with different visualizations in different contexts. The idea was to display classes in a conventional class diagram context as they are and in a difference view context as classes with difference information such as red and green color. Enhancing the FSA library would have been resulting in a complete redesign of the library and an adaptation of all existing diagrams and plug-ins using the library. Therefore we decided to cut-out the meta-model for each diagram and copy it to our plug-in. In addition, we had to relax some restrictions concerning association types and cardinalities, e.g. difference class diagrams may contain classes with the same name and the same package, which was not allowed in the original meta-model.

Today, we support difference class diagrams and therefore the visualization plug-in consists of copies of nearly all meta-model elements representing class diagrams such as DiffUMLClass, DiffUMLAttr, DiffUMLAssoc, etc. We also developed a generic solution to add unparsing facilities for the difference information, which enables us to easily modify existing meta-models to display difference information.

In order to test our calculation algorithm, we developed a plug-in, that exports the pure model information of Fujaba class diagrams as XMI document. In general, we export all visible class diagram elements and leave out all other elements, e.g. get- and set-methods. The exported model is consistent, which means that we also export all necessary data types as well as all stereotypes. The XMI document format is compatible with Poseidon's XMI format [3]. Therefore our difference calculation algorithm is able to work with XMI documents produced either by Fujaba as well as by Poseidon.

4. FUTURE WORK

The short-term target is to complete the visualization plug-in and to enhance the plug-in with additional kinds of diagrams. We plan a sequence such as statecharts, activity diagrams and collaboration diagrams. An integration of additional diagram kinds will be long-term targets. Equipped with our generic difference unparsing facilities such an integration should be easy. In particular, we will test our developed facilities by integrating statecharts, first. In addition, we hope to benefit from the UML 2.0 MOF based meta-model and XMI export, which will produce a normalized XMI document and not specific documents from Rose, Fujaba or Poseidon.

Our long-term target is to develop an interactive merging process. Hence our difference calculation algorithm uses heuristics, which have been optimized for a certain set of diagrams but will also produce bad results on other ones. The idea is to integrate the developer in the process and benefit from his/her changes. Therefore we also support the developer with specific editing operations either for setting similarities as well as for merging elements. A first step is the n-button and u-button, which allow the developer to select one of the two available alternatives from the different diagrams. The result of the process will be a consistent merged class diagram.

5. ACKNOWLEDGMENTS

Special thanks to Jürgen Wehren, who is developing the difference calculation algorithm and to Stephan Lück, who is integrating the difference view plug-in into the Fujaba Tool Suite.

6. REFERENCES

[1] S. Burmester, H. Giese, J. Niere, M. Tichy, J. Wadsack, R. Wagner, L. Wendehals, and A. Zündorf. Tool integration at the meta-model level within the fujaba tool suite. In *Proc. of the Workshop on Tool-Integration in System Development (TIS), Helsinki, Finland, (ESEC / FSE 2003 Workshop 3)*, pages 51–56, September 2003.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[3] Gentleware. *Poseidon for UML. Online at http://www.gentleware.com (last visited June 2004).*

[4] IBM. *Rose, the Rational Rose case tool. Online at http://www.rational.com (last visited June 2004).*

[5] T. Mens. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, 28(5):449–462, 2002.

[6] D. Ohst. *Versionierungskonzepte mit Unterstützung für Differenz- und Mischwerkzeuge*. PhD thesis, University of Siegen, Siegen, Germany, 2004. in german (to appear).

[7] D. Ohst, M. Welle, and U. Kelter. Difference tools for analysis and design documents. In *Proc. of the IEEE International Conference on Software Maintenance 2003 (ICSM2003), Amsterdam, The Netherlands*, pages 13–22. IEEE Computer Society Press, 2003.

[8] C. Schneider, A. Zündorf, and J. Niere. Coobra - a small step for development tools to collaborative environments. In *Proc. of the Workshop on Directions in Software Engineering Environments (WoDiSEE), Edinburgh, Scotland, UK*, May 2004.

[9] P. Selonen. Set operations for unified modeling language. In *Proceedings of the Eight Symposium on Programming Languages and Tools, SPLST'2003, Kuopio, Finland, June*, pages 70–81. Kuopio, Finland: University of Kuopio, 2003.