Constructing Difference Tools for Models Using the SiDiff Framework

Research Demonstration

Maik Schmidt
University of Siegen
Software Engineering Group
Hölderlinstr. 3, 57068 Siegen, Germany
mschmidt@informatik.uni-siegen.de

Tilman Gloetzner
ETAS GmbH
Global Operations
Borsigstraße 14,70469 Stuttgart, Germany
tilman.gloetzner@etas.de

ABSTRACT

Model-driven development requires a full set of development tools. While technologies for constructing graphical editors, compilers etc. are readily available, there is a lack of approaches for constructing version management tools which compare models and show their difference. The general problem is aggravated by the fact that such tools must consider the semantics of each particular model (or diagram) type, i.e. a whole family of tools needs to be constructed. This research demonstration shows how such families of difference tools can be constructed using the SiDiff framework.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software configuration management*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Algorithms, Design

1. INTRODUCTION

Model-based software development is becoming daily practice in important application areas, such as classical information systems, embedded software in automotive systems, or aerospace systems. Different modellings languages and tools are used in these application areas, e.g. UML-based tools, Matlab/Simulink, or ASCET [1]. Further domain-specific languages are expected in the future. With model-based software development, models are changed frequently and are developed in teams. This leads to multiple revisions and variants of models. Hence tools providing version management services, such as comparing and merging of documents, are needed. Normally, several different model (or diagram) types are used in model-based software development. In principle, each model type requires its own specific toolset. Thus a whole family of tools needs to be constructed. This research demonstration shows an approach for constructing families of difference tools using the SiDiff framework. The construction of a difference tool involves two major tasks: (1) Implementing an algorithm which compares two diagrams and which computes a difference and (2) implementing a user interface which displays the difference. Suprisingly, the external design of the user interface and its internal architecture have a significant impact on what is expected as the conceptual result of the difference computation. Section 2 will illustrate this by showing three different user interface designs and discuss the required depiction of the differences. Section 3 points out how the SiDiff framework can be adapted to fullfil all sorts of requirements.

ICSE'08, May 10–18, 2008, Leipzig, Germany.
ACM 978-1-60558-079-1/08/05.

2. USER INTERFACE DESIGNS

2.1 Using a Unified Diagram

The SiDiff framework [2] has been used to construct a difference tool embedded in the Fujaba environment. The basic idea is to "unify" both diagrams. The specific parts of the diagrams are coloured red and green, the common parts are shown in grey and are shown only once. Parts of the diagrams can only be common in this user interface design if they are identical. The layout of the graph can be chosen similar to one of the compared diagrams or be computed from scratch; in any case, layout changes cannot be displayed with this user interface design and must be considered irrelevant in the computation of the difference.

2.2 Synchronized Parallel Display

Another user interface design is the integrated parallel display with synchronized scrolling in both models. The ASCET-Diff environment [1] uses this design. This design is also very common in difference tools for texts. Corresponding model elements can be (slightly) different here, different layouts in both models can be displayed, but may be difficult to detect by a user.

2.3 Clickable List of Local Changes

The SiDiff framework has also been used to construct a difference tool embedded in the Matlab/Simulink environment [4]. This user interface design uses three separate windows. One is a list of "local changes", so to say small differences which constitute the complete difference. Each local change is described by a short text. Clicking on the entry opens two Matlab windows that show both diagrams (or only the relevant parts of them). The modified parts involved in this local change are highlighted. While the previous user interface designs show the complete difference, this approach can show only one local change at a time. Common parts of both diagrams, i.e. parts which are not covered in any

local change, can be different. The layout of both compared diagrams is fully preserved. Therefore changes of the layout, such as moving a diagram element over a significant distance, can be considered as relevant should this be desired.

3. ADAPTABILITY

3.1 Configuration Data

The SiDiff-Project follows the approach to build one kernel which is adaptable to a large range of model types and to all relevant user interface designs. Difference computation in SiDiff has the following main features:

SiDiff compares models based on the similarity of model elements (rather than on persistent identifiers as in other approaches, see [3, 5] for a more detailled discussion). A configuration for the SiDiff kernel basically specifies for each type of diagram element how to compute the similarity. Typically, the specification consists of a set of similarity-relevant properties such as local or remote attributes or structural properties, an algorithm that is to be used for comparing the selected properties, and a weight for each property. Currently, SiDiff offers around 20 algorithms for computing the similarity of properties.

The configuration depends on the diagram type, on how diagrams are typically changed, and even on the taste of the users, i.e. whether a user considers certain modifications as important.

SiDiff represents a difference basically as a table of correspondences. A **correspondence** consists of a pair of references to an element in the first and second diagram which are determined as "matches", i.e. these elements are similar engough to be considered as "common" in both diagrams. Often, they are required to be strictly equal. Using the table of correspondences and the sets of elements in the first and second diagram, one can easily derive the "inserted" model elements. Corresponding elements with changed attributes or references are considered as changed. The configuration for a diagram type specifies for each diagram element type a least similarity of corresponding elements, i.e. pairs of elements whose similarity is below a threshold cannot form a correspondence. The configuration further specifies a set of element types for which correspondences are to be output in the table of correspondences. This set depends on the user interface design.

Further configuration items specify the heuristics to form correspondences and other optimization strategies. The effort to write a new configuration is comparable to writing a metamodel of the diagramm type. In fact, large parts of the configuration data are almost equivalent to a metamodel of the diagram type. Essential differences between a metamodel and the configuration data are the weights, and all aspects that control details of the comparison algorithm to be applied.

3.2 Interfaces of the SiDiff Kernel

While most adaption issues are handled by configuration data, some remaining issues - notably the technical integration into existing environments - are better resolved by appropriate interfacing and a choice of "glue code". The architecture of the SiDiff kernel is shown in [2]. This architecture has proven to be easily adaptable, i.e. without much glue code, to the architecture of editors and environments.

The models to be compared can be accessed in two ways: (a) They may be read from XML files. A run-time representation is built from scratch here. (b) One uses an existing run-time representation. The representation might stem from an editor which runs in parallel on the same virtual machine. Such a representation can be accessed via an adapter. EMF- and MOFLON/JMI- based representations are currently supported by the SiDiff core. At first sight, the second approach appears to be more elegant. However, experience shows that the internal representation of diagrams in editors is often not usable as a basis to compute differences as they may contain a large number of auxiliary objects. These objects may be useful for an editor, but they are semantically irrelevant. This applies in particular for runtime representations that are automatically generated using MDA frameworks. The auxiliary objects can drastically increase the volume and complexity of the configuration data. The first approach allows one to weed out unwanted data very easily. XSLT transformations for instance result in tree representations of the models that are very easy to handle.

The basic result of the correspondence computation is the table of correspondences. It must be post-processed in most cases. For example, if the user interface design is based on a unified diagram, this diagram needs to be constructed first. Other user interface designs need additional attributes of each correspondence, e.g. the degree of similarity of the corresponding elements.

4. SUMMARY

Constructing a comprehensive version management tool set for supporting model-based development is expensive. The tools needs dedicated difference tools which consider the characteristics of the model types. Additionally, the user interface designs depends on the context. The SiDiff systems shows that an essential component of difference tools, the computation of the difference, can be implemented efficiently using a standard framework. Thus, the effort to create a appropriate tool family shifts from the computation of the differences to a suitable presentation of the comparison result. The SiDiff kernel has also been used as a basis of a tool for analyzing version histories of models [6].

5. REFERENCES

[1] ETAS: ASCET Products `http://www.etas.com/en/products/ascet_software_products.php`, 2007

[2] The SiDiff Project `http://www.sidiff.org/`; 2007.

[3] Kelter, U.; Wehren, J.; Niere, J.: A Generic Difference Algorithm for UML Models; Proc. GI-Fachtagung Software Engineering 2005, Essen, LNI; 2005

[4] Stürmer, I.; Dörr, H.; et al.: Das MATE Projekt - Visuelle Spez. von MATLAB/Simulink/Stateflow Analysen und Transformationen; Dagstuhl Seminar Modellbasierte Entwicklung eingebetteter Systeme; 2006

[5] Treude, C.; Berlik, S.; Wenzel, S.; Kelter, U.: Difference Computation of Large Models; p.295-304 in: Joint ESEC/FSE Conference, Dubrovnik, Croatia; 2007

[6] Wenzel, S.; Hutter, H.; Kelter, U.: Tracing model elements; p.104-113 in *ICSM'07*, Paris, France; 2007