A. Weisbecker, M. Burmester & A. Schmidt (Hrsg.): Mensch und Computer 2015 Workshopband, Stuttgart: Oldenbourg Wissenschaftsverlag, 2015, S. 411-417.

Towards Collaboration on Accessible UML Models

Stephan Seifermann, Henning Groenda

Software Engineering, FZI Research Center for Information Technology, Karlsruhe

Abstract

The Unified Modeling Language (UML) is one of the most used software description languages. Its graphical syntax, however, impedes visually impaired people from taking part in discussions. Editors for textual and graphical UML syntaxes exist but lack advanced support for collaborative editing. In this paper, we describe our plans on implementing a collaborative editing environment consisting of an accessible textual UML editor with state-of-the-art user support and consistency preservation mechanisms for real-time collaboration. We present our ideas to discuss their completeness or identify additional accessibility features. An industrial software engineering team that includes visually impaired engineers will evaluate our results continuously.

1 Introduction

Models are well known in the software engineering domain. They are used to model the structure, behavior, or interaction in a software system. The most commonly used modeling language is the Unified Modeling Language (UML) (OMG 2011). Users of other domains have picked up the modeling approach as well. They often use the UML or languages based on the UML such as EAST-ADL (EAST-ADL Association 2013) or AUTOSAR (AUTOSAR Development Corp. 2013) in the automotive or SysML (OMG 2012) in the systems engineering domain. Commonly, users read and edit models using graphical representations.

Graphical representations provide a good overview of a system and improve comprehensibility. Unfortunately, these representations are not accessible for visually impaired people. Human assistants often bridge the gap and translate the model into a textual representation, which is accessible. This process takes time and impedes real-time collaboration scenarios. Additionally, it is error-prone on updates and personnel-intensive.

There are approaches on textual representations of UML with limited coverage of the implemented UML diagram types. They usually do not provide feature-rich, accessible

editors, which are necessary for effective interaction with the model. The transfer between graphical and textual representation is often limited to one direction ignoring the changes on the other side. This prohibits collaborative editing.

In this paper, we present our realization approach of an accessible UML editor that enables the collaboration on UML models in an accessible way. The accessibility will be achieved by a textual syntax, and versatile assistive reading and editing capabilities. Model synchronization mechanisms enable collaboration. The implementation will be based upon the commonly used Eclipse integrated development environment (IDE), which's core parts are accessible (Eclipse 2013). Our results will be published as open source on GitHub (Cooperate Project Team 2015). We present our approach in order to discuss the completeness and gather ideas for additional accessibility features.

The remainder of this paper is structured as follows: Section 2 presents our approach for accessible interaction via a feature-rich textual UML editor. In Section 3, we describe the collaborative editing support via model synchronization. Section 4 discusses related work. In Section 5, we conclude and outline future work.

2 Accessible Interaction with UML Models

Reading and editing requires interaction. UML defines an abstract syntax and a graphical concrete syntax. An abstract syntax describes what information a model contains and how it is stored. A concrete syntax describes how the information is presented to users. An example is provided in Figure 1. We address accessibility by providing a concrete syntax mapping from model elements to structured text. An assistive editor complements the syntax. The following shows why a textual representation is accessible and discusses the implementation of necessary editor features for accessible interaction.

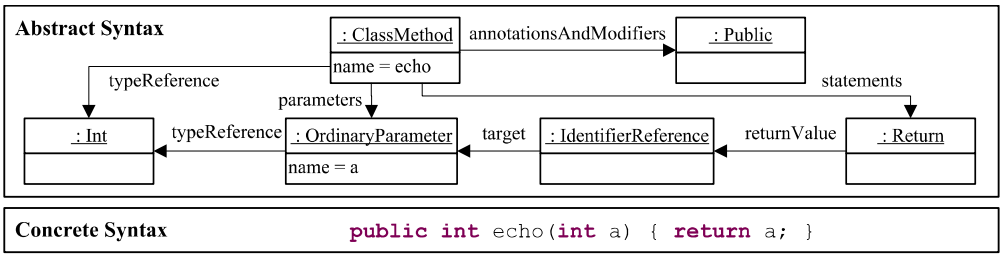

Figure 1 Representation of a Java method in concrete Java and abstract JaMoPP (Heidenreich, et al. 2009) syntax

Existing accessibility tools such as screen readers or braille displays can access textual syntax. We complement this by mapping domain-specific abbreviations, e.g. structured text, to special audio outputs for screen readers. Existing textual concrete syntaxes are limited in at least one of the categories: completeness, comprehensibility, or accessibility. For instance, miming graphical elements such as arrows with a minus and a greater symbol affects accessibility. The XML Metadata Interchange (XMI) serialization of UML models is another

inaccessible syntax: It is complete but it is not meant to be editable for humans because of the verbose XML syntax. An accessible textual concrete syntax must be concise, and allow omission of information. The latter is important to support common UML usage in discussions, in which information such as types of attributes is omitted or deferred. We plan to compare twelve existing syntaxes, determine best and worst practices, and create a new accessible syntax. Half of them are mentioned in related work.

An editor must support two use cases, which are not disjoint but have different focuses: modeling from scratch and modifying an – at least partially – existing model. When modeling from scratch, the user focuses rather on creating than on modifying a model. We identified three major assistive features that have a considerable impact on efficient accessible editing: code completion, accessible error reporting and refactoring support.

Code completion provides keywords and identifiers of elements that are syntactically allowed at the current cursor position. Thereby, users do not have to know and type the exact name of keywords or elements, which increases efficiency and leads to syntactically correct models. Human-readable references instead of technical identifiers such as randomly generated unique identifiers ease the understanding and memorization. Groupings of the code completion suggestions make suggestions more accessible in case of a large list because it is not necessary to read not-intended entries. The exact grouping depends on the UML diagram type and on the context.

Real-time error reporting allows fast fixing of issues and prohibits subsequent errors. The Eclipse IDE already provides an error view and mechanisms to show errors in text editors. We reuse this feature but improve the accessibility: If an error is detected, a non-intrusive audio notification will be issued in addition to the usual Eclipse mechanisms. The user can ignore this notification or can listen to the full error message via a shortcut. Shortcuts also simplify the navigation between error view and the corresponding model part. Additionally, the errors are separated into syntactic and semantical errors. Semantic errors are violated constraints. The separation is indicated in the error description and the audio feedback.

Refactorings allow the fast execution of common edit operations. The editor for the concrete textual syntax supports existing general refactorings such as renaming of identifiers as well as modeling-specific refactorings such as changes on a method's signature. Besides the provided refactorings, custom refactorings will be possible via extension mechanisms.

When modifying an existing model, the user spends more time on reading and understanding. Navigation and customization of the current view heavily support this. The editor will provide adjusted features of the Eclipse IDE to achieve this. For instance, links from model elements to their declaration allow fast navigation as known from programming language editors. We envision a modeling-specific search. Instead of type selection, the search executes predefined common queries such as finding all classes that implement a selected interface in a class diagram. These queries are accessible via context menus and a search dialogue. In addition to searches, commonly used jumping shortcuts such as jumping to a base class are available in context menus. Focused reading is also supported by reducing visible information using folding techniques. Shortcuts allow to fold and unfold information at the current position and indicate availability of these operations via audio feedback.

3 Collaborative Editing of UML Models

The UML is often used during discussions to communicate problems or develop ideas in a collaborative and iterative way. Graphical representations exclude visually impaired people. Using the concrete textual syntax proposed in the last section solves this problem only if a) every team member is familiar with it, and b) the changes are synchronized between all team members. In this section, we describe how our editor solves both of these problems.

We address a) by providing consistency preservation between concrete syntaxes. Each team member can choose the syntax freely. There are two fundamental approaches to keep concrete syntaxes consistent: synchronization, and a common model (see Figure 2).

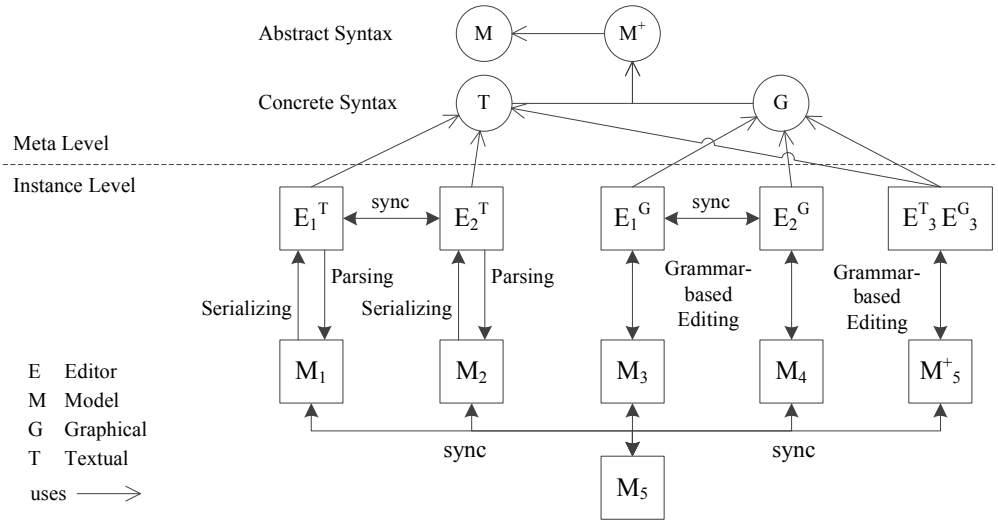

Figure 2 Methods for keeping concrete syntaxes consistent

On the meta level, the concrete syntaxes T and G are based on the abstract syntax M and M^+ respectively. M^+ extends M and allows storing layout information for model elements.

The common model approach shown on the right side circumvents inconsistencies by simply using the same model M^+_5 as data source for all concrete syntaxes. This model contains all information across concrete syntaxes and is directly edited via so-called projectional editors E^T_3 and E^G_3. The editor(s) can use the syntax definition G and T even at the same time. Projectional editors ensure consistency by grammar-based editing. They limit free text editing by providing templates and predefined edit operations in case of complex changes.

The synchronization approach shown in the rest of Figure 2 keeps separate models for every concrete syntax editor and synchronizes changes between them. Textual editors $E_1{}^T$ $E_2{}^T$ using a parsing and serializing approach can be used alongside with graphical editors $E_1{}^G$ $E_2{}^G$ using grammar-based editing. The synchronization requires a considerable effort for defining the mapping between M_1, M_2, M_3, M_4, M_5, and M^+_5. We favor this approach because of less

editing restrictions, and reusability of existing (graphical) editors. The synchronization will be based on a bijective mapping function for model and layout elements to prohibit information loss and ensure completeness. Layout information between different concrete syntaxes cannot be mapped as indicated by missing arrows between E_2^T and E_1^G in Figure 2.

Enabling concurrent real-time editing addresses b). All team members can use their own favorite (accessible) editors. We plan to build on the synchronization framework ModelBus (Fraunhofer FOKUS 2014), which already covers the synchronization of graphical layout information. We plan to use the Eclipse Shared Editing project (The Eclipse Foundation 2007) for textual layout synchronization. The synchronization will take place after every complete change automatically. A change is complete if the syntax is valid after editing. In case of concurrent editing conflicts on the same elements, a standard resolution strategy such as *last commit wins* will be applied. Additionally, we plan to integrate the modeling environment Enterprise Architect[1], which is often used in industry, via the same synchronization mechanism.

A major problem for visually impaired people is tracing synchronized changes from other team members. To make collaborative editing accessible, our editor provides a list of incoming changes that is updated automatically in real-time. A non-intrusive audio notification indicates a new entry. Shortcuts provide the navigation between the change list and affected model locations. The change list contains a comprehensive textual.

4 Related Work

There are two common approaches to make UML models accessible: Textual editors and haptic approaches. Textual editors pass the information to assistive techniques such as screen readers and braille displays. Haptic approaches use special devices to make graphical information touchable or combine navigation devices with audio output.

PlantUML (PlantUML 2015) aims for generating graphical diagrams from textual descriptions. It supports a wide range of diagrams and is already used to make the UML accessible by users of the BlindUML user group (BlindUML 2015). The examined Netbeans editor supports syntax highlighting but does not provide linking between elements, or text completion. Mixing model and layout information slows down comprehension.

Syntaxes that meme programming languages such as TextUML (Chaves 2015), Umple (Lethbridge 2014), or UMLGraph (Spinellis 2003) are developer-oriented. The editor support, however, is poor compared to common programming editors: Neither UMLGraph, nor yUML (Harris 2015) have dedicated editors. None of the remaining editors provides linking. Additionally, the Umple editor does not provide text completion.

[1] http://www.sparxsystems.eu/enterprisearchitect/newedition/

Earl Grey (Mazanec and Macek 2012) provides a syntax for class, sequence, and state diagrams with an editor that supports syntax highlighting, linking, code completion, and folding. Its syntax and editor are optimized for sighted people. The editor has no special features for accessibility such as audio output. It does not support synchronization and collaborative editing. This holds true for all other editors as well except for Umple.

TeDuB (Horstmann, et al. 2004) is a haptic approach and allows navigating through the hierarchical representation of UML class, use case, sequence, and state diagrams via joystick. Editing is not supported and the representation of the hierarchy cannot be adjusted. Metatla, et.al. (Metatla, et al. 2012) use a similar approach enabling collaborative editing. Unfortunately, the accessible hierarchical representation in the public available prototype could not be activated.

Tactile displays provide a haptic representation of graphical information. Loitsch and Weber (Loitsch and Weber 2012) stated that resolution and size constraints severely limit the displayable amount of information compared to graphics. Editing is not yet supported.

5 Conclusion

This paper showed our plans on implementing an accessible editing environment for UML models. Section 2 showed that a textual concrete syntax integrated in a state-of-the-art editor allows the accessible interaction with UML models. The editor's following features need adjustments for accessibility: Text completion, real-time error reporting, and modeling-specific refactorings will support the modeling from scratch. Linking, outline views, predefined common queries, and folding will support modifying existing models.

Section 3 sketched the intended support for collaborative editing of UML models in teams. We showed alternatives and opted for a solution based on a distributed editing environment consisting of multiple synchronized editor instances. The synchronization between textual and graphical editors allows free tool selection for each team member based on their specific needs. The presented accessibility features such as change histories, or audio notifications will specifically support visually impaired people.

Our next step is the detailed evaluation of the twelve existing textual syntaxes for UML with respect to accessibility impediments. In parallel, we will survey language workbenches that support implementing the features mentioned in the previous sections. The most promising workbench is used for the implementation of the sketched solution. In the midterm, the synchronization mechanisms between graphical and textual editors will be implemented.

Acknowledgements

This work is funded by the German Federal Ministry of Labour and Social Affairs under grant 01KM141108.

References

AUTOSAR Development Corp. *AUTOSAR - Automotive Open System Architecture v4.2.* 2013. http://www.autosar.org/specifications/release-42/ (accessed June 9, 2014).

BlindUML. *BlindUML User Group.* 2015. https://groups.yahoo.com/neo/groups/blinduml (accessed June 09, 2015).

Chaves, R. *TextUML Toolkit.* 2015. http://abstratt.github.io/textuml/readme.html (accessed June 09, 2015).

Cooperate Project Team. *Cooperate Project.* 2015. https://github.com/Cooperate-Project.

EAST-ADL Association. *EAST-ADL Domain Model Specification v2.1.12.* 2013. http://www.east-adl.info/Specification/V2.1.12/html (accessed June 9, 2015).

Eclipse. *Accessibility.* 2013. https://wiki.eclipse.org/Accessibility (accessed June 8, 2015).

Fraunhofer FOKUS. "ModelBus User Guide." 2014. https://www.modelbus.org.

Harris, T. *Create UML diagrams online in seconds, no special tools needed.* 2015. http://yuml.me (accessed June 09, 2015).

Heidenreich, Florian, Jendrik Johannes, Mirko Seifert, and Christian Wende. *JaMoPP: The Java Model Parser and Printer.* Technical Report, TU Dresden, 2009.

Horstmann, M., et al. "TeDUB: Automatic Interpretation and Presentation of Technical Diagrams." *CVHI'04.* 2004.

Lethbridge, T. C. „Teaching modeling using Umple: Principles for the development of an effective tool.“ *CSEET'14.* 2014. 23-28.

Loitsch, C., and G. Weber. "Viable Haptic UML for Blind People." *ICCHP'12.* Springer Berlin Heidelberg, 2012. 509-516.

Mazanec, M., and O. Macek. "On General-purpose Textual Modeling Languages." *DATESO'12.* CEUR-WS.org, 2012. 1-12.

Metatla, O., N. Bryan-Kinns, T: Stockman, and F. Martin. "Cross-modal Collaborative Interaction Between Visually-impaired and Sighted Users in the Workplace." *ICAD'12.* Georgia Institute of Technology, 2012. 164-171.

OMG. *Systems Modeling Language (SysML) v1.3.* 2012. http://www.omg.org/spec/SysML/1.3/.

—. *Unified Modeling Language (UML) v2.4.1.* 2011. http://www.omg.org/spec/UML/2.4.1/.

PlantUML. *PlantUML.* 2015. http://plantuml.sourceforge.net/ (accessed June 09, 2015).

Spinellis, D. "On the declarative specification of models." *Software, IEEE*, 2003: 96-95.

The Eclipse Foundation. *RT Shared Editing.* 2007. https://wiki.eclipse.org/RT_Shared_Editing (accessed June 10, 2015).