Scribbler: From Collaborative Sketching to Formal Domain Specific Models and Back Again

Martin Vogel, Tim Warnecke, Christian Bartelt

University of Clausthal
Department for Computer Science - Software Systems Engineering
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld - Germany
{m.vogel, tim.warnecke, christian.bartelt}@tu-clausthal.de

Abstract. Most of the time developers make extensive use of software tools in a software development process to support them in their day-to-day work. One of the first and most important phases of this process is the design phase, but within this phase intuitive and easy to use tools, which support the creative but also collaborative workflow (parallel/distributed), are missing. At the moment, developers use whiteboards to express their ideas in team meetings. Subsequently a coworker takes a picture of the sketches and remodels them with a modeling tool. That procedure is very inconvenient, error-prone and hindering in a creative modeling cycle. For overcoming this ineffective process this paper shows a new software tool using digital whiteboards to transform free hand sketches in formal models and back again during modeling in a distributed team. The transformation is completely independent from a pre-defined modeling language. The tool provides also a training mode to learn new graphical syntax elements and map these to formal metamodel entities.
Video: https://www.youtube.com/watch?v=0i3M9djPrRM
[Mirror: http://sse-world.de/index.php?cID=3611]

Keywords. Sketch Recognition, Model Based Software Development, Collaborative Software Engineering

1 Introduction and Motivation

Software development is a creative and distributed team process. The early and creative design phase is very important for the success of a software project. Normally software designers do not use modeling tools, like for example MagicDraw UML [1], in such an early phase because of their inconvenient handling. Modeling tools are made for precise model design, but not for creative sketching. Because of that software designers are using whiteboards in team meetings to visualize and communicate their ideas within the group [2]. Nevertheless after a meeting one of the designers has to transform the sketches in formal models using the previously rejected modeling tools. This transformation is error-prone because the designer tries e.g. to optimize the diagram or forgets some elements. This can lead to a situation where the new formal

diagram cannot be recognized anymore by the other team members because of it changed appearance. Another widely known problem in this domain describes that everybody has to be present to such a creative meeting. One possible solution is to use Screen Sharing Software to share some kind of drawing or modeling software and additionally utilize a telephone conference system. An issue with this attempt is that two or more software applications are used together and none of them is in particular conceptualized for software designers. In the last few years, several research initiatives were started with the topic, intuitive modeling respectively model sketching. In [3] a recognition mechanism for sketched model elements was presented which uses a similarity calculation between drawing traces based on the Levenshtein distance [4] and is also a basis for the tool implementation explained in this paper. Some innovative research results about sketch recognition in the area of requirements modeling were described in [5]. Some further recognition techniques based on vector comparison between sketches and GEF/GMF model elements were published in [6] and [7]. In comparison to these approaches the tool presented in this paper named Scribbler does not need a GEF/GMF specification of concrete language syntax and it also uses the sketched trace for recognition of model elements.

2 Scribbler: Architecture and Implementation

In sum Scribbler is an extensible drawing tool. It provides services, which allows developers create fast new pluggable components for their requirements.

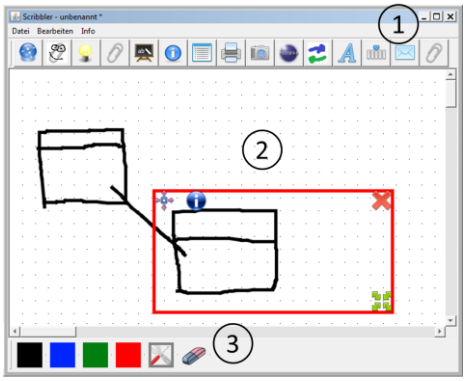

Fig. 1. Interface of Scribbler

A screenshot of the tool is shown in Fig. 1. At the top of it (1) all current loaded plugins are represented with an icon. In the center (2) is the canvas and last but not least the toolbar is located at the bottom (3), which consists of four colors, an edit button and a rubber.

2.1 Architecture and features of Scribbler

The most important object in Scribbler is the SketchObject[SO]. Each sketch has a unique [SO] that contains the x- and y-coordinates, the convex hull and the mouse

movements which were necessary to draw the sketch. For the convex hull the Quick-Hull algorithm is used [8]. Thus the convex hull is needed because each sketch can be modified afterwards. Additionally the mouse movements for each sketch are recorded. This is necessary to recognize handwriting and more complex objects. These three parts of information are atomic for each sketch in Scribbler. Scribbler's core provides an extensibility mechanism which enables developers to create their own plugins for a new domain like learning, recognizing and transforming sketches. The core fires events for the most basic operations, like draw, edit and save and put these into a queue. Each plugin has an own thread, which sequentially reads this queue and executes the stored actions asynchronously. Thus the plugins are completely decoupled from the core. Plugins can communicate with each other over previously registered interfaces. For collaboration Scribbler uses another concept as typical screen sharing tools. Only mouse movements/events and sketch coordinates instead of whole images are transferred. Thus, devices with a small bandwidth and different screen resolutions have no drawbacks. Another benefit is that the server and Scribbler save the history of each session. So the genesis of the sketch model remains for the following meetings.

3 Scribbler: Sketching Formal Domain Specific Models

The tool Scribbler presented in this paper tries to accomplish the previously described problems modeling in distributed teams, transforming of sketches in formal model elements and back and recording the development process of a sketched model in the early phases of a software development process.

3.1 Sketch Recognition of Domain Specific Models

Recognizing sketches is a difficult task especially if model elements with different graphical representations of arbitrary domain specific languages are used. Scribbler solves this problem with help of a modified algorithm used in [9]. Similar to this approach Scribbler uses a grid, encodes stroke properties into numbers and compares sketches with help of the Levenshtein distance. But additionally Scribbler uses two different sequences of numbers and scaling to probably improve the recognition rate. Fig. **2** shows how both sequences are constructed. First a drawn sketch is scaled up to a previously defined size and with the help of a grid the intersections between it and the sketch are calculated. Later on the intersections are represented by the number corresponding to the field in the grid where it was discovered. Looking at the circle example in Fig. **2** step 1 the algorithm begins at field 1 and finds two intersections with the grid and this leads to the sequence {*1 1*}. Continuing to field 2 a new intersection is found (previously found intersections are ignored) and the sequence is extended to {*1 1 2*} and so on until the last field which results in {*1 1 2 3 4 5 8 9 12 13 14 15*}.

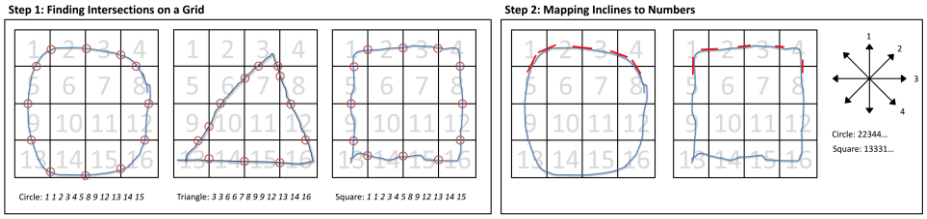

Fig. 2. Visualization of the Sketch Recognition Algorithm

As seen in step 1 a circle and a square could be the same sequence so another sequence is introduced in step 2 based on the inclines of the strokes at the intersections, e.g. is a stroke horizontal it is mapped to number *3*. Subsequently the Levenshtein algorithm compares in a first step the first sequence generated from the intersections with entries from a knowledgebase (described in the following paragraph). If two or more similar entries are found, like in the example, the second sequence is compared.

The previously described algorithm can't be used without a knowledgebase which contains a list of previously learned sketches. Hence Scribbler has an own dialog for training new sketches for different domains and saving them in a file with their corresponding sequences. Thereby, an extensible knowledgebase for each domain can be created. After learning a series of sketches the next step is to transform recognized sketches in formal model elements. In preparation for this the DSL-metamodel (based on Ecore) and its graphical representation (GMFGraph-file) are needed. Once the knowledgebase and both model files are loaded the mapping from sketches to Ecore elements have to be done. Afterwards the whole hand drawn diagram can be exported to a single GMF-file. Another functionality of Scribbler is that users at different locations are able to draw simultaneously on the same canvas. This is possible because of an integrated client-server-approach which allows it to start immediately a new server. The drawn sketches are transferred as soon as possible so every user is aware about the actions of the other participants. It should be noted that the knowledgebase is maintained by every single client, because only drawing actions are transferred.

4 Evaluation

During the term of the project three industry partners from different domains used Scribbler for their daily work in creative meetings with their customers and in architectural meetings for about four weeks. Every team had an experimental setup composed of a digital whiteboard and tablet pcs. Furthermore they got a catalogue of questions to answer to evaluate Scribbler. The three teams - altogether 14 participants - used the Scribbler only for collaborative work, especially the history viewer/server session to comprehend their decision from the last meeting. The teams assessed this functionality as valuable and it is very helpful for their daily work. Furthermore they used the learning environment to insert elements for their own domain languages. Scribbler was able to learn all of these elements and the recognition rate was very good. Concerning the usability and the training period every participant rated the Scribbler as good.

5 Conclusion and Acknowledgements

Ensuing from the requirements regarding an intuitive modeling infrastructure that does not hinder the creative engineering process, the sketching/modeling platform Scribbler was implemented. Scribbler allows a distributed, parallel (collaborative) sketching of engineering models on digital whiteboards, the transformation of those sketches in (semi-)formal domain specific models and back again, an easy and interactive learning of new domain specific syntax elements and a recording/playback of the modeling/sketching history. The realized use cases and the functionality are explained in this paper. For future work the recording of further context information during the sketching modeling process (e.g. voices of modelers within the history of model evolution etc.) is planned.

This research work was supported by "German Federal Ministry of Education and Research" (BMBF) within the Project "KoMo – From Sketch to Model: Cooperative Modeling with Domain Specific Languages" (2011-2013).

6 References

[1] MagicDraw, "NoMagic - MagicDraw," 2013. [Online]. Available: www.nomagic.com/products/magicdraw.html. [Accessed: 05-Jul-2013].

[2] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2007, pp. 557–566.

[3] U. B. Sangiorgi and S. D. J. Barbosa, "SKETCH: Modeling Using Freehand Drawing in Eclipse Graphical Editors," in *Proc. FlexiTools Workshop*, 2010.

[4] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," in *Soviet Physics Doklady*, 1966, vol. 10, pp. 707 – 710.

[5] D. Wüest, N. Seyff, and M. Glinz, "FlexiSketch: A Mobile Sketching Tool for Software Modeling," in *Mobile Computing, Applications, and Services*, vol. 110, D. Uhler, K. Mehta, and J. L. Wong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 225–244.

[6] A. Scharf and T. Amma, "Dynamic Injection of Sketching Features into GEF Based Diagram Editors," 2013, pp. 822–831.

[7] A. Scharf, "Scribble - A Framework for Integrating Intelligent Input Methods into Graphical Diagram Editors," in *Software Engineering 2013 Workshopband (inkl. Doktorandensymposium)*, 2013, pp. 591–596.

[8] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, 1996.

[9] A. Coyette, S. Schimke, J. Vanderdonckt, and C. Vielhauer, "Trainable sketch recognizer for graphical user interface design," in *Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction*, Berlin, Heidelberg, 2007, pp. 124–135.