Scribble - A Framework for Integrating Intelligent Input Methods into Graphical Diagram Editors

Andreas Scharf

andreas.scharf@cs.uni-kassel.de

Abstract: Creating modern software is a challenging but also a very creative task. Especially in early development phases like requirements engineering or architectural design software engineers use different mediums to manifest their thoughts and to discuss possible ambiguities. These mediums range from analog tools like pen & paper or whiteboards to digital ones like tablet pc's or smartboards. Whereas editing capabilities for analog mediums are restricted to add/remove operations, there already is great support in the digital world to later move, rotate or even share thoughts and diagrams with distributed teams. In addition the tool support for creating complex diagrams used to express software architecture and design along with sophisticated techniques like code generation is large. However, most of these tools restrict the user input to valid data, decreasing the software engineers flexibility which is why they often fall back to non formal tools. This doctoral thesis aims to combine the flexibility of informal sketching with the power of formal software engineering tools. As part of this thesis, a new generic framework will be created which dynamically augments new and already existing diagram editors with sketch-based input features.

1 Motivation

Building modern software is a complex but also a very creative task and a lot of frameworks for solving common problems in nearly every domain are available. On the one hand these frameworks simplify the development of big and more stable software but on the other hand one has to check thoroughly how to design and interconnect different software components. A good example are web frameworks: even though they support developers creating amazing desktop like applications, they also introduce great complexity like new communication concepts or connecting systems written in different programming languages. Therefore, writing the actual code is not the first task of software engineers in most cases. In fact, there are usually a couple of steps before the code writing phase like requirements engineering and architectural design. Especially during these early stages of development software engineers use different mediums to manifest their thoughts, discuss architecture and design with their team members or to clarify ambiguous interpretations of a conversation [MCK07]. Informal analog tools like pen & paper or whiteboards provide a great degree of freedom concerning allowed content but lack editing flexibility like copy/paste, scale and rotate or even share sketched content with distributed teams. With digital sketch-input enabled devices like tablet pc's or smartboards it is possible to overcome these problems [JINW04] and a remarkable amount of work has already been done

to further support this task [GSW01, MIEL99]. However, most approaches focus on creating and editing informal sketches while developers often use formal diagram editors for modifying diagram types like UML class diagrams or sequence diagrams. On the one hand these tools usually provide sophisticated editing-, validation- and code generation support but on the other hand they mostly stick to a strict syntax which limits the developers creativity by restricting editing operations to valid data. For that reason, developers fall back to more informal tools in many cases. Due to that, a lot of work has to been done twice since people first manifest their thoughts using pen & paper for instance and digitize the result afterwards to benefit from the mentioned tool support.

Therefore it is desirable to combine the strengths of software engineering tools with the flexibility of informal sketching. Several approaches addressing this problem have emerged, e.g. [QJJ03, PPY10] which provide support for recognizing hand drawn content. But most of the work either focuses on providing sketching capabilities for a single diagram tool or lack user control over several recognition aspects like used algorithms or how formalization happens. Marama [GHNN06] and SKETCH [SB10] both try to provide generic sketch support for graphical diagram editors in the Eclipse [1] environment. Marama is a diagram editor generation framework which supports developers in creating graphical editors. Sketch-input capabilities are restricted to editors generated by Marama. SKETCH on the other hand aims to add sketching features into new and already existing diagram editors based on the Eclipse Graphical Editing Framework (GEF). However, development of this framework is stuck and no information about extensibility and flexibility is given.

In the next section the resultant research questions are presented and the approach to answer these questions will be outlined.

2 Research questions and Approach

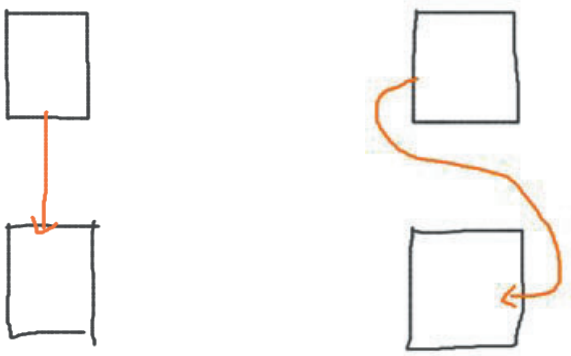
Although there already has been a remarkable amount of work in sketch-input related research areas, there are still a lot of open research questions in different categories. Some of them are of a very technical nature whereas others are more general. In the following the terms *sketch-based* and *scribble-based* are used synonymously and are related to hand drawn input whereas *Scribble* refers to the framework which should be created within the scope of this doctoral thesis.

- *Can complex graphical diagram editors be seamlessly augmented to support sophisticated sketch-based input? If yes, how and to what degree?*
- *How should a framework be designed to be useful for different graphical editor frameworks in different environments?* Are there any common components that can be used cross-platform? Example environments to evaluate include (but are not limited to) Eclipse and Visual Studio [2].

[1] `http://www.eclipse.org`
[2] `http://www.microsoft.com/visualstudio/eng`

- *What kind of extension points does the framework have to provide?* Which parts of the framework should be extensible, configurable or even exchangeable?
- *What are "intelligent" input methods?* How can different input capabilities of different devices like single-touch capable smartboards or multi-touch capable tablets be addressed? What about speech-input? Is it possible and reasonable to combine different kinds of input methods?
- Various questions concerning technical and usability aspects of the Scribble framework:
  - *Can incremental formalization be integrated?* At what exact points in time is recognition of sketched input performed?
  - *Can formalized and sketched input co-exist?* This is a technical question on the one hand but a usability question on the other hand. Do users want to be able to switch between these two forms of visualization?
  - *Is it possible to provide (auto-) correction mechanisms of unrecognized or wrong recognized elements?* It is possible that the meaning of a sketch is not clear at the point the user created it but elements created subsequently may give hints to what the initial sketch was meant to be.
  - *Is it possible to provide convenient text input mechanisms?* The core of this question is: How can text input be distinguished from all other sketched input? A related problem is the technical support for recognizing handwritten text. Even though there are plenty algorithms that claim to support text recognition, only a few perform well.
- *How can different types of edges be recognized?* Recognizing different types of nodes is well supported by a lot of algorithms even without a large amount of training data. However, recognizing edges is a different kind of recognition problem and cannot be solved with classical template matching based approaches. Figure 1 visualizes this problem.
- *How much training is needed to provide acceptable recognition results?* Who is training the algorithms? Can training data be shared?
- *Is scribble-based modeling accepted by modelers? What would be additional requirements on such a framework and its usability?*

To answer the research questions above, a requirement analysis has to be done first. This starts with asking software engineers about their opinion of usability aspects in the "end user" role on the one hand and what features they would like to have as integrators in the "developer" role on the other hand. These information would give some kind of feature list and also provide first hints of potential feature requirements of "client-frameworks" like Eclipse or Visual Studio. Also the extensible or exchangeable Scribble framework parts can be identified.



(a) Straight edge of type E between two nodes (b) Curved edge of type E between two nodes

Figure 1: The edge problem: two differently drawn edges of the same type E.

Analyzing different graphical editor creation frameworks along with diagram editors produced with these APIs would be the next step to get information about similarities and differences. All similarities are potential candidates for common components which could be used cross-platform. Furthermore, this analysis can be compared against the above mentioned requirement analysis to answer the question if it possible to support sophisticated sketch-based input within the evaluated client-frameworks and to what degree.

To support different input devices like single-touch tablet pc's or multi-touch capable smartboards along with a possible combination of speech-input, different use-cases covering common usage aspects have to be created. An example use-case would be "Move a node" which can be run on single- and multi-touch capable devices leading to different behavior of the Scribble framework. In this example, the user could use the common "two finger swipe" gesture to move a node on a multi-touch device. For single-touch devices some additional UI might be used to distinguish sketch-input from move commands.

Integrating sketch-, text- and speech-input recognition first involves some research of existing approaches and a study of their performance (qualitatively and quantitatively) and their eligibility to be integrated or extended. Also the research question how different types of edges could be recognized is part of this phase. Depending on the requirement analysis above, these parts are also subject to be extensible or exchangeable.

A very important question concerns the amount of needed training data and if this data can be shared. If it is reasonable to share training data amongst users, it would make sense to evaluate existing training data providers if any or create a new service for that purpose.

During the above mentioned research and different analysis, a prototype of the Scribble framework can be created to get a first picture to what extend all requirements can be covered. This prototype can be used to evaluate the different requirements of the end user and developer role. This evaluation also gives first results to the question if scribble-based modeling is accepted by modelers and might also reveal additional requirements.

3 Status Quo and Future Work

A first requirement analysis to create a feature list for both end user and developer role has already been done but this list is expected to change as mentioned in the last chapter. Related work like Marama and SKETCH have been evaluated and suitable algorithms for recognizing hand drawn content where identified.

The GEF framework in the Eclipse environment was the first candidate to analyze. A strategy to dynamically inject sketch-based input features into GEF along with a first prototype of the Scribble framework has been created. This work was submitted as a technical research paper [SAon] to the International Conference on Software Engineering 2013 and has been accepted for presentation in San Francisco from May 18th - 26th.

Evaluation of previous work in the domain of sharing training has revealed that it is reasonable to share such data in some kind of online database [FGP+09]. Although there is already some training data available, this data contains material only for primitive types like rectangles or ellipses in most cases. For that reason an "online training center" called WebScribble [Sei12] has been created as part of a bachelor thesis. WebScribble is a web application to create new diagram editor types along with supported node and edge types. For each node and edge type the user can add sample sketches and associate these sketches with the appropriate type. All results are persisted in a database whose content is planned to be accessible in form of a web service. This training data can then be imported into the Scribble framework and decreases the amount of required training material for other users.

The next steps include the evaluation of Visual Studio as another environment to create graphical diagram editors for. This will answer the question if common Scribble framework components are technically reasonable. A first look at available approaches for recognizing edges was made. This is a difficult problem since the same edge can look arbitrarily different and simple template matching algorithms cannot be used here. More work has to be investigated into this field and perhaps a new edge recognition algorithm has to be created. Also some currently available approaches for text recognition have been analyzed. The performance of most approaches is relatively worse and not usable in productive environments. Microsoft's text recognition capabilities on tablet pc's are excellent but only available in windows machines. However, the support of text-input methods is a crucial feature for some diagram types like UML class diagrams. Therefore more work has to be investigated here as well. A possible integration of speech-input is planned but no work has been done in this field yet.

The Scribble framework prototype is designed to work well with single-touch capable devices. If a device supports multi-touch, this feature should be utilized to increase user experience which also is future work.

At last, an evaluation of the final Scribble prototype has to be undertaken and suitable methods of measurement have to be found to answer the above mentioned research questions. It is planned to let students and professional developers test Scribble to see if the framework can be integrated easily into existing diagram editors and if requirements concerning usability are met.

References

[FGP+09] Martin Field, Sam Gordon, Eric Peterson, Raquel Robinson, Thomas Stahovich, and Christine Alvarado. The Effect of Task on Classification Accuracy: Using Gesture Recognition Techniques in Free-Sketch Recognition, 2009.

[GHNN06] J. Grundy, J. Hosking, Nianping Zhu, and Na Liu. Generating Domain-Specific Visual Language Editors from High-level Tool Specifications. In *21st IEEE/ACM International Conference on Automated Software Engineering, 2006. ASE '06.*, pages 25–36, 2006.

[GSW01] François Guimbretière, Maureen Stone, and Terry Winograd. Fluid interaction with high-resolution wall-size displays. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, pages 21–30. ACM, 2001.

[JINW04] Wendy Ju, Arna Ionescu, Lawrence Neeley, and Terry Winograd. Where the wild things work: capturing shared physical design workspaces. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, pages 533–541. ACM, 2004.

[MCK07] Gina Venolia Rob DeLine Mauro Cherubini and Andrew 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*, pages 557–566, 2007.

[MIEL99] Elizabeth D. Mynatt, Takeo Igarashi, W. Keith Edwards, and Anthony LaMarca. Flatland: new dimensions in office whiteboards. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 346–353. ACM, 1999.

[PPY10] Beryl Plimmer, Helen C. Purchase, and Hong Yul Yang. SketchNode: intelligent sketching support and formal diagramming. In *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*, OZCHI '10, pages 136–143. ACM, 2010.

[QJJ03] Qi Chen, John Grundy, and John Hosking. An E-whiteboard application to support early design-stage sketching of UML diagrams. In *In Proceedings of the 2003 IEEE Conference on Human-Centric Computing*, pages 219–226. IEEE CS Press, 2003.

[SAon] A. Scharf and T. Amma. Dynamic Injection of Sketching Features into GEF based Diagram Editors. Accepted at International Conference on Software Engineering 2013, submitted for publication.

[SB10] Ugo Braga Sangiorgi and Simone D.J Barbosa. SKETCH: Modeling Using Freehand Drawing in Eclipse Graphical Editors, 2010.

[Sei12] M. Seiler. WebScribble - Entwurf und Realisierung einer Webanwendung zur Erstellung von grafischen Editoren durch Freihandzeichnungen, 2012. thesis.