Flexible Sketch-Based Requirements Modeling

Dustin Wüest and Martin Glinz

Department of Informatics, University of Zurich, Switzerland
{wueest,glinz}@ifi.uzh.ch

Abstract. [Context and motivation] Requirements engineers and stakeholders like to create informal, sketchy models in order to communicate ideas and to make them persistent. They prefer pen and paper over current software modeling tools, because the former allow for any kind of sketches and do not break the creative flow. **[Question/problem]** To facilitate requirements management, engineers then need to manually transform the sketches into more formal models of requirements. This is a tedious, time-consuming task. Furthermore, there is a risk that the original intentions of the sketched models and informal annotations get lost in the transition. **[Principal ideas/results]** We present the idea for a seamless, tool-supported transition from informal, sketchy drafts to more formal models such as UML diagrams. Our approach uses an existing sketch recognizer together with a dynamic library of modeling symbols. This library can be augmented and modified by the user anytime during the sketching/modeling process. Thus, an engineer can start sketching without any restrictions, and can add both syntax and semantics later. Or the engineer can define a domain-specific modeling language with any degree of formality and adapt it on the fly. **[Contribution]** In this paper we describe how our approach combines the advantages of modeling with the freedom and ease of sketching in a way other modeling tools cannot provide.

Keywords: Requirements sketching, adaptable formalization, requirements modeling

1 Introduction

When modeling requirements during requirements elicitation, stakeholders and requirements engineers would benefit from being able to freely draw sketches first and convert these into models later. This is due to the fact that sketching fosters creativity [7, 3] and can also be applied by stakeholders who do not master a modeling language with a formal syntax. However, the power and ease of sketching comes at the expense of a *media break*, i.e., of later having to re-create the sketched models from scratch in a modeling tool in order to be able to manage requirements properly. This re-creation process is time consuming, error-prone, and can lead to a loss of information [5].

Sketch recognition tools have been created to relieve the task of converting sketches into models (e.g. [4, 10]). However, such tools rely on predefined notations, so that the user needs to know the underlying modeling language and is restricted to its vocabulary. Hence, in terms of expressivity and creativity, sketch recognition tools do not help. Moreover, they introduce the problem of sketch recognition errors.

In this paper, we give a preview of a new approach that (i) allows users to sketch any informal models, (ii) provides means for assigning syntax and semantics to sketched elements on the fly, and (iii) supports the transformation of sketches into classic semi-formal models (e.g., a class diagram or a statechart) by a semi-automated method, thus avoiding media breaks. The goal of our approach is to unite the flexibility of unconstrained sketching with the power of formal modeling.

The rest of the paper is structured as follows. In the next Section we give an overview of our planned research. Section 3 discusses related work. In Section 4 we conclude the paper.

2 Flexible Sketch-Based Requirements Modeling

2.1 Main Goal

The goal of the planned research is to provide a sketch-based modeling approach. We envision that our approach allows requirements engineers and stakeholders to sketch any informal models. Users should not be restricted in what they may draw, nor should they need to decide for a specific notation beforehand. Further, our approach should support the semi-automated transition of sketches into classic models. Our key research activities include: (i) identify the needs of requirements engineers with respect to sketch-based modeling, (ii) provide method and tool support, and (iii) evaluate the approach and demonstrate its practical benefits.

We plan to realize tool support by incorporating an existing sketch recognition framework that compares drawn shapes with the symbols included in a library. In contrast to other approaches, our symbol library does not hold a predefined modeling language, but is dynamic: users can augment and modify it at any time during the modeling process (see Fig. 1). Syntax and semantics can be added to the sketches on demand, and modeling may be performed on various levels of formality.

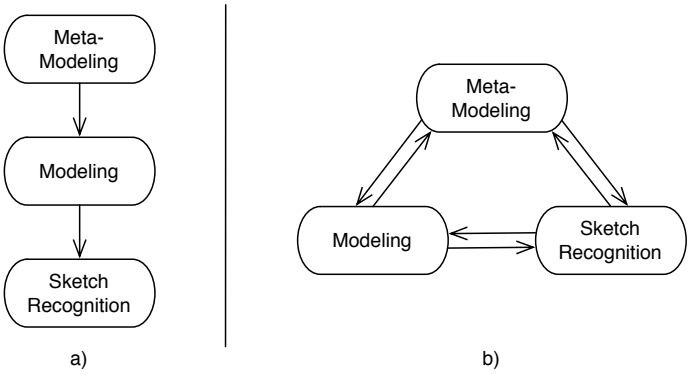

Fig. 1. (a) Existing sketch recognition approaches restrict users to do tasks in a given order. (b) Our envisaged approach allows to change flexibly between tasks.

2.2 Scenarios for Flexible Sketch-Based Requirements Modeling

We illustrate the usefulness of our approach with two typical elicitation scenarios. Both describe a meeting of a requirements engineer with stakeholders in an early phase of elicitation.

Scenario 1. A requirements engineer and two stakeholders stand in front of an electronic whiteboard. The interface of our envisaged tool is projected onto the whiteboard. The stakeholders start to sketch requirements. They do not use a specific notation since they are not familiar with modeling languages. Towards the end of the meeting, there are annotated rectangles, circles, and arrows on the board. In a discussion with the stakeholders, the requirements engineer clarifies the meanings of the symbols. After the meeting, the engineer selects one of the drawn symbols and assigns a type to it. The tool adds the symbol to the symbol library. Similar symbols are then identified automatically. Next, the engineer selects one of the arrows and defines it as a Connector that determines the order of two connected symbols. The engineer also defines the type of the connection as being a temporal relationship. The other arrows are recognized automatically. The engineer has now a minimalistic modeling language and a formalized version of the sketched model in that language. The engineer further defines new symbols that represent the drawn symbols in a formalized version of the sketch, and can now switch between the sketch version and the formal version.

Scenario 2. Some days before the meeting, the stakeholder sends a documentation of a domain-specific language (DSL) including two notations to the requirements engineer, who is also knowledgeable in metamodeling. The stakeholder intends to use the same language in the meeting. With the help of the documentation, the engineer creates the DSL description within our software tool. This work results in two symbol libraries, one for each notation. When the engineer and the stakeholder meet, most of the symbols drawn by the stakeholder are immediately recognized by the software tool. Some symbols are not identified because they are distorted. The engineer selects them and assigns the correct type manually. The stakeholder also introduces a new symbol not included in the DSL description. The engineer adds the symbol to the description by assigning a type to it. In total, the stakeholder draws two diagrams using the notations that are part of the DSL. Both diagrams include the same rectangular symbol, but it has a different meaning in each of the diagrams. To enable correct symbol recognition, the engineer tags the elements of the first diagram (e.g. by encircling them) and assigns the proper symbol library to the tagged elements. Then the engineer does the same for the second diagram.

2.3 The Elements of Our Approach

In the following paragraph we discuss the required features and components of a software tool that supports the activities described in the scenarios.

Modeling Interface. A tool following our approach has to support both informal and formal requirements modeling activities. On one hand, it must allow unconstrained sketching on an empty canvas. On the other hand, the tool must provide a way to edit (e.g. scale, move, copy) objects and to formalize the sketches. Both modes must be integrated in a unified, unobtrusive, simple interface.

Symbol Libraries and Multiple Contexts. A symbol library includes symbols that can be recognized when a user is drawing them. Symbol libraries must be modifiable at any time. Further, we envision separate libraries for different modeling languages, and therefore a tool must be able to manage a list of symbol libraries. People like to use and mix different visual conventions [5], thus the tool must support the use of different notations at the same time. When a user starts to draw symbols that are part of multiple symbol libraries (which can happen very quickly for overloaded, simple shapes such as rectangles or circles), it is impossible for the tool to detect the correct language automatically. Even if the user adheres to one particular language such as UML, some shapes have different meanings in different diagram types. Thus a user must have the option to tag parts of the sketch and define the context by choosing the correct library for the recognition from a list. Explicitly assigning contexts also helps when symbols are used inconsistently.

Sketch Recognition Framework. A recognition framework processes sketches on two levels. Low level recognition combines groups of individual pen strokes to distinct symbols. High level recognition compares drawn symbols with those in the symbol library. For our tool, the recognition framework must be able to handle a dynamic symbol library. Sketch recognition algorithms can be divided into two categories: while an *offline* algorithm starts the recognition process only after the user has finished sketching, an *online* algorithm starts processing right away when the user begins to draw, and takes temporal properties of drawn strokes into account. These temporal properties provide additional hints for the sketch recognition engine. As our approach is interactive, we will need an online algorithm. We plan to modify an existing online sketch recognition algorithm such that it fits our needs.

Modeling Language Definition. The tool must provide an interface that helps users in defining a modeling language. Users must be able to easily add new symbols to the symbol library. They also need convenient means for adding syntax and semantics to symbols. No scripting or programming should be required during the metamodeling task. Creating a lightweight, sketch-based interface that also allows to add all kinds of syntax and semantics is a challenging task. This is probably the most critical part of the tool. Therefore, our work will not only be based on research in the fields of requirements engineering (RE) and sketch recognition, but also relies on findings from work in the human-computer-interaction domain. The interactivity and the flexibility of a tool add a great deal to its usability. The graphical user interface for the language definition needs to be seamlessly integrated into the sketch environment.

2.4 Benefits and Limitations

Our tool will allow requirements engineers to formalize sketches into models without having to recreate them, and therefore eliminate media breaks. It gives them the flexibility to co-evolve diagrams and meanings of the drawn elements, and also allows them to define a DSL first, start sketch-based modeling using this DSL and then augment or modify the DSL in the modeling process.

The co-evolution usage style is more flexible than any existing tool-supported approach, but is probably limited to simple, lightweight modeling languages. Otherwise, this style requires too much metamodeling overhead and can only be used properly

by metamodeling experts. Working with a lightweight modeling language [6] matches our intention of supporting early requirements engineering. At this stage, when stakeholders and requirements engineers sketch their intentions and creative ideas, using a lightweight language is crucial, because adherence to a sophisticated modeling language would impede the creative flow [7, 3].

The DSL style has similarities to using a meta-edit tool (the modeling language definition has to be entered first), but gives more freedom to users as it allows adding or modifying symbol definitions during the modeling process. This style is particularly useful when metamodeling experts predefine a symbol library for a standard modeling language such as UML, which then can be extended by requirements engineers with domain-specific elements.

2.5 Current Research Status

A literature review has shown that while there is a lot of ongoing research in sketch recognition, sketch-based tools do not focus on the RE domain and also have deficiencies in terms of flexibility and ease of sketch-based modeling. From discussions with requirements engineering experts about this topic, we conclude that our approach will provide value to requirements engineers. We currently have completed a preliminary concept of our approach. Right now we are comparing different sketch recognition concepts and look at theories about graphical user interfaces. Future steps include the creation of a tool prototype, its evaluation with a case study, and its improvement.

3 Related Work

Related research in this area includes software tools that incorporate a natural drawing interface and a sketch recognition engine, e.g. SUMLOW [4], and Tahuti [10]. These tools use predefined libraries and thus only support certain languages.

Some meta-tools allow users to define their own languages, e.g. Marama [9] and MetaEdit+ [11]. The meta-tool then builds the actual modeling tool. If users want to change the language, they have to go back to the meta-tool, modify the definitions, and tell the meta-tool to rebuild the modeling tool.

Domain-independent sketch recognition toolkits like SketchREAD [1] and InkKit [12] can handle additional domain languages. These notations must either be scripted/ programmed or imported via a library plug-in. Thus, these tools are better suited for developers rather than requirements engineers.

Gross [8] and Avola *et al.* [2] present sketch recognition frameworks that work with dynamic libraries of user defined shapes. While Avola *et al.* focus on the technical details of sketch recognition, Gross additionally enables users to define some spatial constraints between shapes. Apart from this, none of the two approaches support user defined syntax and semantics. We will reuse a sketch recognition framework like the one presented in [2] for our work.

4 Conclusions

State-of-the-art sketch-based interfaces either lack formalization functionality or restrict users to use specific modeling languages. Although some of the discussed tools can be used for requirements engineering, they are not built for this purpose. We envisage a tool that is tailored to the needs of requirements engineers and allows unconstrained sketching with a subsequent, semi-automated formalization of the sketches. With this approach we overcome media break problems. As we are at the beginning of our research, next steps include building and evaluating a prototype tool in order to assess the usefulness of such a tool and our approach.

References

1. Alvarado, C., Davis, R.: SketchREAD: a Multi-Domain Sketch Recognition Engine. In: 17th Annual ACM Symposium on User Interface Software and Technology. pp. 23–32. ACM, New York (2004)
2. Avola, D., Del Buono, A., Gianforme, G., Paolozzi, S., Wang, R.: SketchML: a Representation Language for Novel Sketch Recognition Approach. In: 2nd International Conference on Pervasive Technologies Related to Assistive Environments. pp. 1–8. ACM, New York (2009)
3. Black, A.: Visible Planning on Paper and on Screen: the Impact of Working Medium on Decision-Making by Novice Graphic Designers. Behavior and Information Technology 9, 283–296 (1990)
4. Chen, Q., Grundy, J., Hosking, J.: SUMLOW: Early Design-Stage Sketching of UML Diagrams on an E-Whiteboard. Softw. Pract. Exper. 38(9), 961–994 (2008)
5. Cherubini, M., Venolia, G., DeLine, R., Ko, A.J.: Let's Go to the Whiteboard: How and Why Software Developers Use Drawings. In: '07 SIGCHI Conference on Human Factors in Computing Systems. pp. 557–566. ACM, New York (2007)
6. Glinz, M.: Very Lightweight Requirements Modeling. In: 18th IEEE International Requirements Engineering Conference. pp. 385–386. IEEE Computer Society, Washington (2010)
7. Goel, V.: Sketches of Thought: a Study of the Role of Sketching in Design Problem-Solving and its Implications for the Computational Theory of the Mind. Ph.D. thesis, University of California at Berkeley, Berkeley (1991)
8. Gross, M.D.: Stretch-A-Sketch: a Dynamic Diagrammer. In: '94 IEEE Symposium on Visual Languages. IEEE Computer Society, Washington (1994)
9. Grundy, J., Hosking, J.: Supporting Generic Sketching-Based Input of Diagrams in a Domain-Specific Visual Language Meta-Tool. In: 29th International Conference on Software Engineering. pp. 282–291. IEEE Computer Society, Washington (2007)
10. Hammond, T., Davis, R.: Tahuti: a Geometrical Sketch Recognition System for UML Class Diagrams. In: '02 AAAI Spring Symposium on Sketch Understanding. pp. 59–68. AAAI Press, Menlo Park (2002)
11. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: a Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In: Constantopoulos, P., Mylopoulos, J., Vassiliou, Y. (eds.) Advanced Information Systems Engineering, LNCS, vol. 1080, pp. 1–21. Springer, Heidelberg (1996)
12. Plimmer, B., Freeman, I.: A Toolkit Approach to Sketched Diagram Recognition. In: 21st British HCI Group Annual Conference on People and Computers. pp. 205–213. British Computer Society, Swinton (2007)