Hammering Models: Designing Usable Modeling Tools

Ko-Hsun Huang, Nuno Jardim Nunes, Leonel Nobrega, Larry Constantine, Monchu Chen

Hammering Models: Designing Usable Modeling Tools

Ko-Hsun Huang[1,2], Nuno Jardim Nunes[1], Leonel Nobrega[1], Larry Constantine[1], and Monchu Chen[1],

[1] Madeira Interactive Technologies Institute and University of Madeira, Campus Universitário da Penteada, 9000-390 Funchal Portugal
[2] Institute of Applied Arts, National Chiao Tung University, Hsinchu, Taiwan
kohsun@csie.nctu.edu.tw, {njn, lnobrega, lconstantine, monchu}@uma.pt.

Abstract. A modeling tool not only helps users express their ideas and thoughts but also serves as a communication platform among domain experts, designers, developers, and others practitioners. Existing modeling tools have shortcomings in terms of supported functionality and situated usability or do not meet the needs of users of varying levels of expertise. To facilitate improvement of such modeling tools, this research begins by identifying common problems in existing tools and proceeds by borrowing concepts from grounded theory to develop a framework of redesign guidelines. A case study illustrates how this framework can be used by applying it to MetaSketch, a metamodeling tool. The study employs multiple user experience research methods, including usability tests with paper prototypes, observations, interviews, and contextual inquiries. A set of core tasks and two significant modeling approaches were identified that directly influence interface and interaction design for modeling tools.

Keywords: Model-Based Design, Interactive Systems, Usability and Software, User Experience Design, Metamodels, Metamodeling, Participatory Design,

1 Introduction

Software developers and designers have long complained that the "tools of the trade" are unsupportive and unusable. With the advent of model-driven development, the same concerns apply to modeling tools. A good modeling tool not only helps its users express their ideas and thoughts but also serves as a communication platform among domain-experts, designers, developers, and others practitioners. These various usages must be recognized and supported in varied ways. The primary purpose of this research is to identify and understand the significant work styles of potential tool users and to identify design implications for modeling tools [1-3]. We also reconsider the functionality a metamodeling tool can support, such as an iterative modeling process, giving domain-experts greater freedom to alter metamodel definitions based on knowledge emerging from case and field studies.

1.1 Modeling Tools

Software development is still dominated by so-called third-generation programming language (3GLs) introduced over a half-century ago. Computer-Aided Software Engineering (CASE), the first modeling tools, arose from the so-called software crisis and the assumption that better tools would help programmers create better software [4]. Despite the partial failure of such tools [3], CASE did bring more advanced tools to the basic toolset of editor, compiler and debugger. Integration of fourth-generation programming languages (4GLs) with CASE tools enabled creation of high-end tools that boosted productivity. However, editors, compilers and debuggers, now combined into Integrated Development Environments (IDEs), are still the primary tools of software developers.

It has been argued that models might improve software engineering as they have in traditional engineering disciplines [5]. Model-driven development is no panacea, and it is not without controversy [6]. However, models and modeling languages make possible raising the level of abstraction [7]. Models can also help bridge the gap between developers and other stakeholders, in particular making application domain experts active participants in development. The growth of UML (Unified Modeling Language) and UML-based tools suggests that modeling has potential in typical software development [6]. Despite debate about models in software engineering and the role of UML [8], it is widely accepted that UML lacks the expressiveness needed to address domain specific knowledge, and, therefore, new modeling languages are required. Domain-specific languages have proved to deliver gains when compared with general-propose languages [9]. Capturing domain specificities is crucial to understanding problem contexts and, therefore, finding correct solutions. Domain-specific languages enable creation of models derived from a domain and are more easily understood by domain experts. These languages enable domain experts to contribute directly to problem definition and, by better describing needs, improving software development. Defining domain-specific modeling languages is not easy [7], and the Software Language Engineering discipline [10] has emerged to help.

Multi-language IDEs are common, but multi-language modeling editors are unusual. Benefiting from UML standardization, most popular modeling editors support only UML and its concrete syntax instead of supporting different notations as before. To support effectively the definition of models using domain-specific modeling languages, we need the capability to define these specialized languages. The Generic Modeling Environment (GME) [11] and notably the MetaEdit+ Domain-Specific Modeling environment [12] brought this capability to Domain-Specific Modeling (DSM), where new modeling languages could be defined for particular domains. Approaches like Software Factories [13] and the Eclipse Modeling Framework (EMF) [14] also open new opportunities for DSM.

With the UML 2.0 specification, the Object Management Group (OMG) concluded a long standardization process that not only improved UML as a language, but also built the basis for OMG's Model Driven Architecture (MDA) initiative to support model-driven engineering (MDE)—the concept that software development methods should be focused on creating and exploiting domain models rather than the underlying computing or algorithmic concepts. MDE is meant to increase productivity by maximizing compatibility between systems, simplifying the process of design, and

promoting communication among individuals and teams developing a system [15]. The UML 2.0 language is the most popular implementation of the MDE approach, providing support for modeling the different aspects of a system using different levels of abstraction as proposed in MDA.

1.2 Structure of the MetaSketch Workbench

MetaSketch is a modeling language workbench that allows the rapid development of new modeling languages for model-driven engineering. MetaSketch is based on OMG standards including MOF 2.0, OCL 2.0, and XMI 2.1, and is specially tailored for creating new members of the UML family of languages. Currently, the workbench comprises three pre-beta release tools: the Editor, the Merger and the Designer:

- MetaSketch Editor. The editor, the heart of the workbench, is simultaneously a modeling and a metamodeling editor, meaning that the same editor can be used to create the metamodel and the models that conform to the created metamodel.
- MetaSketch Merger. This tool is only applicable at the metamodel level and basically resolves all the Package Merges used in the definition of a metamodel. In addition, the hierarchy of packages is flattened into a single package that contains all language construct definitions merged.
- MetaSketch Designer. This work-in-progress enables the graphical creation of the concrete syntax for each language. Currently, these definitions have to be done directly in the XML files.

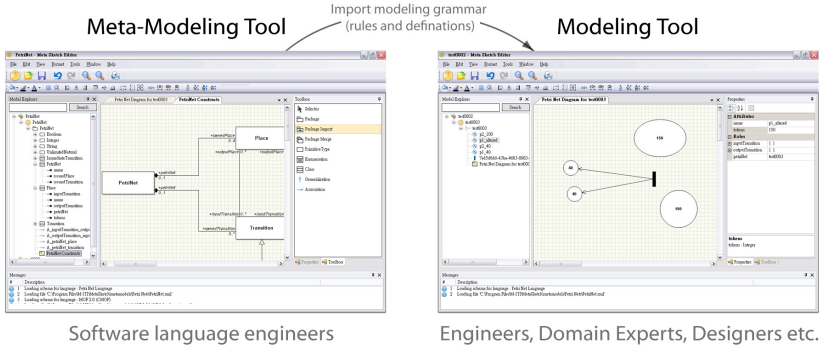

Fig. 1. Original structure of MetaSketch. On the left is the tool for software language engineers developing UML languages (metamodels). On the right is the tool enabling domain-experts, designers, developers, and others practitioners to develop models.

The original version of MetaSketch allows software language engineers to define metamodel languages and supports other users—such as designers, developers, and other practitioners—in creating models using a similar interface for metamodel definition and modeling (Fig. 1). This design does not comprehensively support users of different levels of expertise and varied work styles. Whatever kinds of models or metamodels are used, the interface is always the same.

To redesign MetaSketch, we started to reconsider who are the target users and what are their main usages. For software engineering, the current version of MetaSketch already supports quite well the standard metamodel development process. The main shortcoming is the lack of an advanced designing tool to help language engineers define the concrete syntax for new language constructs. For other users, especially field-study researchers and designers, the interface is too complicated. These users are concerned with creating models for expressing concepts and knowledge and would prefer an interaction style closer to conventional editing or drawing tools. Therefore the approach taken was to separate MetaSketch into several different parts to support different uses and to engage all potential user types.

To increase the flexibility and changeability of the metamodel and to support domain-experts in developing and revising metamodels from field cases, we reconsidered the relationship between modeling and metamodeling as an iterative process. We propose the comprehensive framework shown in Fig. 2.

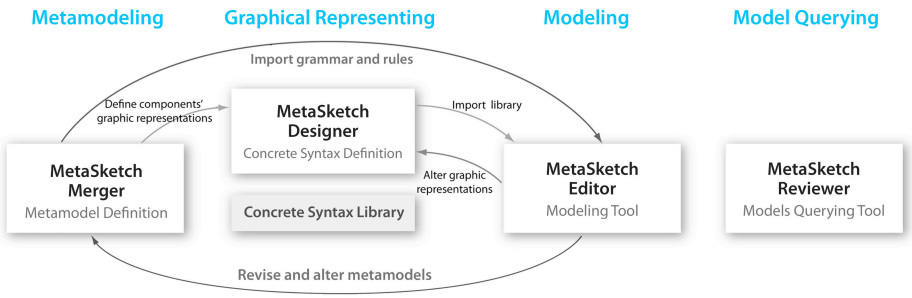

Fig. 2. Framework of modeling tool to support flexible and iterative modeling and metamodeling processes.

1.3 Supporting Iterative Modeling Processes: Case-Driven Theory

We undertook a case-driven theoretical development process, borrowing its main concept from Grounded Theory [16,17], which is a systematic qualitative research method commonly used in the social sciences. Grounded Theory incorporates a variety of methods and an iterative process to build a theory from field data, including extracting concepts from cases and categorizing concepts into a theory. It provides a specific theoretical framework to build domain knowledge and produce a model for application.

Software engineers also define grammars and metamodels for domain specific situations. Consequently, before an entire grammar and rules become well defined, case collections and rough concept sketches of modelers can be considered as a part of meta-modeling development. Therefore, three concepts inspired by Grounded Theory are suggested for designing modeling tools:

- To develop a modeling language employing a field-based or case-driven process. A modeling tool can serve as the sketchpad, which gives researchers and domain experts more freedom to express their conceptualizations of field data.
- To transform cases and raw data into grammar and rules. This is the process of conceptualizing and categorizing data and then generalizing it into the most important representative form (the deductive and inductive process [18]).
- To define a metamodel and to use it to create models for cases through an iterative process. Real cases help software engineers reconsider and revise their original metamodel, and concrete syntaxes can also be improved by the evaluations of designers, researchers, and practitioners.

To support these three basic concepts, there are several functions a modeling tool like MetaSketch should provide in the future:

1. supporting field data collection through an ill-defined modeling language (e.g., a sketch on a napkin);
2. supporting different representations (concrete syntax) for different field data and concepts, therefore providing a modeler with more flexibility and expressiveness;
3. supporting the automatic categorization and contextualization of cases from raw data for future rule definition (generalizing theories from cases [19]).

2 State of the Art

Many studies have analyzed software development practices, but qualitative studies of modeling work practices are relatively rare. Some authors report a gap between how tools represent and manipulate programs and the software developers' actual experiences [20]. This work is mostly based on workstyle models for collaborative software design and shows how a tool can fail to match an intended work context. However, research on practitioners' workstyles helps in understanding how we can create new tools that better support specific workstyle transitions. A survey of how practitioners worked and used tools offered insights into how developers can build human-centered tools but did not focus on modeling tasks [20].

To understand the potential usage and meaning of a tool, it is necessary to find out the needed and desired aspects [21] and to consider the future context of use in the whole design process [22,23]. Usage of modeling tools is hard to predict because they are involved in an uncertain and complex creative process. In addition, a modeling workbench, such as MetaSketch, aims to support multiple types of users, including designers, domain experts, engineers, practitioners, and field researchers. It supports communication and coordination of perspectives among all the users, facilitating a process where these stakeholders can jointly transform their knowledge [24,25]. The research methods used in this study not only have to cover the details of usage and interactions, but also need to reflect the different workstyles and attitudes of users of different levels of expertise [20,26]. To enhance the quality and usability of the system, and to satisfy all types of users, research approaches from both user-centered design and participatory design were considered jointly [25,27,28].

To determine the knowledge required for performing modeling tasks and to understand users' mental models, several techniques for tracing and modeling

cognitive and behavioral processes in system evaluation were applied, including task analysis, protocol analysis, observation, and interviews [29-31]. Techniques to describe how people actually work and to evaluate the current version of MetaSketch—such as, cognitive walkthrough, contextual inquiry, and rapid prototyping—were also considered [32-34].

There are two main challenges in this research. First, most research techniques for usability evaluation focus on interface or task-oriented planning [28,29]. However, modeling is an iterative and creative activity, which reflects the idea and thought of modelers and is an important material to help them think. Assigning users set tasks for observing and analyzing is not suitable in our case.

The second challenge is to investigate and to explore modeling behavior with low-level detailed analysis of interactions. Since the activities of modeling are not clear, to predefine an appropriate set of interactions for each task becomes very difficult. Prototyping at this stage may not reflect the richness of desired behaviors and lacks specification of low-level details of interactions [25,26].

Sketches, storyboards, and wireframes are widely used for interface and interaction design [35,36], and many varied materials and tools support designers in expressing ideas and simulating behaviors [26,37]. However, many techniques and materials are applicable to static interface design, such as the appearance, layout, or graphics for a website, but not for dynamic aspects or details of complex interactivity [38]. A recent survey of 259 practitioners found that 86% designers considered behavior more difficult to prototype than appearance [26]. The perceived and actual properties of a tool also affect possible usage [39,40].

We adapted the paper prototyping approach to make it more flexible for exploring possible functions users expect from a modeling tool. We made all components of the paper user interface independent, movable, and reusable, including system dialogs, tools, and other objects. Think-aloud and Wizard-of-Oz protocols were applied for understanding users mental models. Interactions were recorded by video and later interpreted for task analysis. After each usability test, participants were interviewed about their personal experiences and opinions about modeling tools. We also debriefed end users with the results and discussed how to refine the paper prototype and system response (see next section) [41-44]. The system responses to be simulated by the paper prototype were performed by a so-called “human computer” who was given a text description defining a series of default system responses as operational sequence diagrams covering all anticipated user actions and situations where users want to understand the system behaviors, including by trial and error interactions.

The adaptability of our paper prototype made the usability test an iterative process. In the end, all the tasks involved in modeling were elicited, and all potential interactions to achieve these goals were discussed. The results not only reveal the intentions and strategies that users have in building models but also indicate their expectations for a modeling tool.

The paper prototype is not merely an evaluations tool, but a material for designers to convey their design concepts as well as a platform for communication among domain experts, engineers and practitioners. In this research, we explore all the aspects of modeling activity, and show an integrated approach for usability testing.

3 Usability Test for MetaSketch Editor

MetaSketch Editor is the first component selected for redesign in our process. The original interface of the editor is separated into four components (Fig.3). The Explorer is a tree-view of all entities involved in the model; the Diagram is the main canvas for modeling. The Object Library and Properties share the same window and enable users to add new objects into models and perform advanced editing tasks.

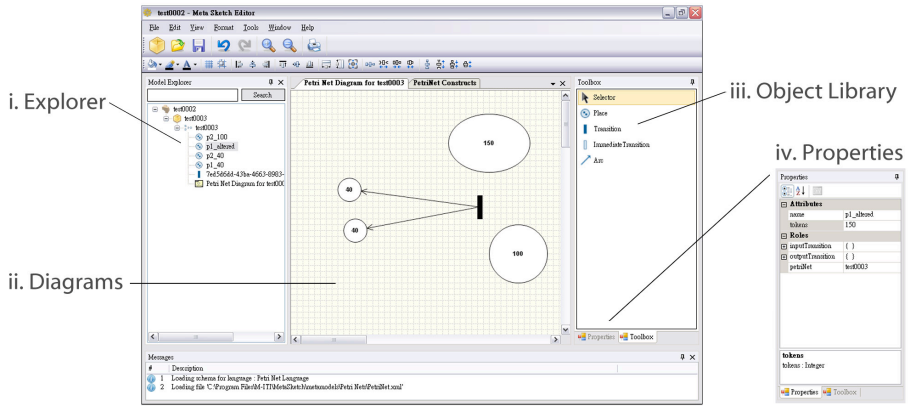

Fig. 3. The interface of MetaSketch Editor is divided into four components: Explorer, Diagrams, Object Library, and Properties.

Initially we conducted a pilot study to gain better understanding of current modeling tools and their interaction design. This study was based on literature reviews and heuristic evaluations of current modeling tools, including meta-modeling tools, such as Eclipse Modeling Framework and MetaEdit+, and common modeling tools, such as Oryx, Aris Express, and Intalio. Based on the results, we discussed a diverse set of design concepts over a period of months with designers and software language engineers [10]. We designed the paper prototype to resemble well-known image-editing tools, and developed several hints and guiding functions to assist users in modeling tasks [45,46]. The prototype (Fig. 4) included basic editing tools: Object Library, Component Explorer, Navigator, Properties Window, system dialogs and hints. There are several new functions:

- Explorer and Diagram. The prototype Explorer provides additional options for users to examine the partial model and the entities of each diagram.
- Object Library. A context menu enables accessing the Object Library and adding different types of entities on the diagram directly.
- Properties. Additional information is provided for each entity, including which diagram or model an entity is involved in.
- Navigator. For editing complex models, users can easily check the undefined or unlinked entities through different filters in Navigator.
- System dialogs and Hints. Instead of traditional text-based descriptions, hints and grammar explanations are provided with graphic examples.

In redesigning MetaSketch Editor, we looked to three types of users: i) experts, who are familiar with a specific modeling language; ii) designers, who explain their concepts and ideas through a model; and iii) researchers, who reveal their knowledge of a certain field or situation through modeling. To understand the different mental models and work styles in all areas, we applied different user experience research methods for usability testing, in particular, paper prototyping [34], thinking aloud [34][47], Wizard of Oz [48,49], observation and interviews [50]. To further analyze the data, we also applied contextual inquiry [33] and task analysis [29].

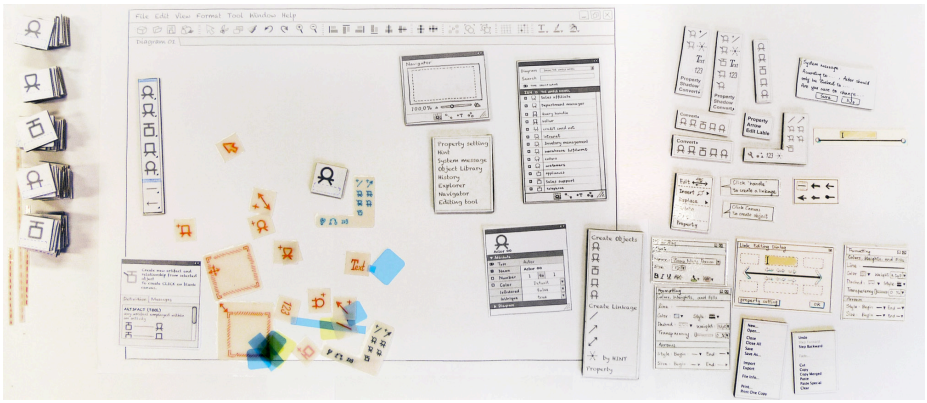

Fig. 4. Based on the pilot study, a flexible paper prototype was designed to support different types of users, including domain-experts, designers, developers, and others practitioners.

We defined a default set of responses to all possible user actions in order to integrate all functions of MetaSketch Editor with the new design concepts. During think-aloud tests, if users expected a different system response from the default, the "human computer" reacted in the way participants wanted, thus avoiding interrupting their current work process and helping to better understand their mental models.

For testing with engineers and designers we selected Activity Modeling and Participation Maps [51,52], since both were well understood by our potential subject group. Five participants were recruited, including four engineers and one designer. The four engineer subjects are experienced practitioners and domain experts, but only one was the software language engineer who developed Activity Modeling. All work in interaction design and are highly capable in usability tasks, such as, thinking-aloud and cognitive walkthroughs, and can give valuable feedback regarding low-level interaction details.

Two modeling strategies were distinguished in the pilot study: sequential and non-sequential. In sequential modeling, users build models one entity at a time. For instance, in flowcharting, users tend to add an entity, and then connect it immediately. In a non-sequential modeling strategy, the user first creates a number of entities and then considers potential connections among entities. Both approaches were observed in the pre-study. To understand if task order and human-computer interaction were influenced by different modeling approaches, the test was separated into two sections:

- Section I: A scenario was provided, which described a retail selling situation. Users were asked to create a Participation Map using the paper prototype. The scenario contained three paragraphs, each describing two or three entities and their relationships. In this section, users could be expected to model the situation using the sequential approach (Fig. 5).
- Section II: Users were given a Participation Map and were asked to recreate it using the paper prototype. In this section, users could be expected to use the non-sequential approach (Fig. 5).

In both sections, users were asked to think aloud to explain what they were looking for, thinking, doing, and what kind of system responses they expected. The purpose was to understand users' mental models and to evaluate our interaction design.

After each test, a brief, semi-structured interview was conducted. Users shared how they used materials or applications for modeling and gave their opinions on the design of the paper prototype [50]. All data was captured on video and later integrated into a task analysis table [41]. To represent the modeling approach applied by each participant in each section, we applied Sequence Model in Contextual Inquiry through a flowchart [53].

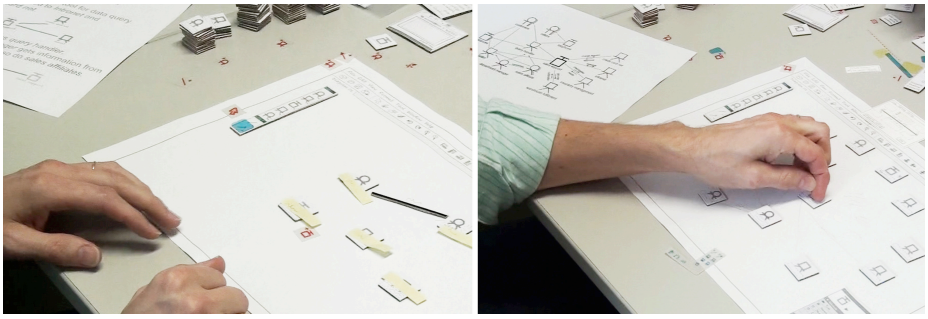

Fig. 5. The left picture shows the first section of the usability test, where the user was asked to create a model from a text-based scenario. On the right is the second section, where the user was provided with a sketch of a Participation Map to build using the paper prototype.

4 Primary and Secondary Modeling Tasks

From the pilot study and design concepts approximately a hundred potential tasks involved in modeling were identified, including adding new entities to the modeling canvas, connecting entities, editing properties or names of entities, and correcting errors. To help us order the frequencies of important interactions and make decisions for the interface design, we have to distinguish the primary and secondary tasks in the modeling activity.

Ten important modeling tasks were identified (Fig. 6). Primary tasks were: creating entities, naming entities, considering possible relationships among entities (creating links), annotating relationships (naming or editing links) and repositioning entities

(moving objects). Secondary tasks occur when users need to correct errors, which are not the goal of modeling but often interrupt their current process. Secondary tasks include: changing the type of an entity, changing the type of relationship, and changing the direction of relationship (changing endpoints of links).

Fig. 6. Left table: primary and secondary tasks in text-based section and in the sketch section. Right table: modeling approaches actually taken by each user in each section.

The primary modeling tasks are creating and connecting entities. We found out that the steps users took and their expectations of system response were highly dependent on the modeling approaches they applied.

- Creating Entities. In the text-based scenario, four of five users modeled the Participation Map using a sequential approach. A context menu enables users to choose entity type and create it on the canvas directly. However, most users still created entities from the Object Library. Users suggested making the context menu visible in the beginning or alternatively providing on-screen tips. For the non-sequential modeling approach, results show that the Object Library provides an easy and smooth way to create all the important entities on the canvas.
- Creating Links. The most diverse interactions were found in how users created links. All users selected the create-link tool from the Object Library in the beginning, but later actions and expectations of system responses depended on their modeling approaches and individual preferences. Three users wanted to create a link between entities by selecting the source and the target. While applying the sequential modeling approach, two users wanted the last clicked object to always become the source for the next action, thus allowing a link to be created by a single click: selecting the target entity. Other possible techniques for link creation

discussed in interviews included clickable handles on each entity to make connection to others and applying a hot key.

- Annotating. Interviews did not indicate that task priority was influenced by the modeling approach. Users were more focused on annotating links in the non-sequential approach (Fig. 6), where editing links becomes one of the primary tasks. According to the interviews, adding a link in the non-sequential approach is one of the important parts of modeling. While users are creating a link between two entities, they are considering the type and details of this relationship. Naming an entity was considered part of creating an entity, and most users tended to insert the name immediately after adding an entity.
- Repositioning Entities. Two users mentioned that repositioning entities actually helps them think. Looking at and reorganizing the canvas helps make sense of the whole model. Hence, repositioning is an important independent activity, quite different from simple revision or editing of the model.

After considering Primary Tasks, we focused on Secondary Tasks, the most important being correcting errors. Since it is not the purpose of modeling itself, the way users corrected errors and revised models was unrelated to the modeling approaches used but was related to individual preferences in handling interruptions.

- Changing Entity Type. It is sometimes difficult to decide the type of entities while adding them to the canvas. Most users agreed that converting an entity into a different type is a frequent modeling task. They wanted to convert an entity into any other type even if that might lead, temporarily, to a grammatical error that might later be corrected. By habit and personal preferences, some users also tend to correct all errors toward the end. Modeling tools should avoid interrupting users with repeated error messages or reminders.
- Changing Relationship Type. Some users did not consider changing the direction of a link as changing the type. Adding an arrowhead was to them a part of annotation or decorating, like changing color or line weight. Instead of creating a directional relationship, they first created an undirected line and later tried to add an arrow by selecting the line and choosing an arrow from the formatting palette (Fig. 7). Whether the interaction design for this task should reflect its semantic connotation in a particular modeling language is unclear, particularly as users will want to be able to easily correct errors in direction.
- Moving Endpoints. With complex models, it is common to create links by mistake. Most users understood that changing endpoints might lead to a grammatically incorrect result but still expected the freedom to alter the model in any way they want. Instead of deleting incorrect links and creating new ones, users want to manipulate links easily, such as by dragging endpoints to new targets. As with changing entity type, the system should not limit user actions.

Tasks like editing properties, reviewing system messages, and checking guidelines and grammar rules are activities independent from modeling. Theses functions are important and should be supported by any practical modeling tool but serve very different purposes from modeling itself.

- Advanced Editing. In creating entities and building a model, setting entity properties is independent from other tasks. Users start considering properties after getting an initial picture of the model as a whole, hence properties should be considered in the second level of information architecture.

- Exploring and Querying. In both the Navigator and Explorer, filter functions provide different views of a model, allowing users to browse by types of objects, unedited objects, and unlinked objects. Users explained that the canvas can become very complicated for large models. Although the Navigator and Explorer were not used in the tests, most users appreciated having the function.
- Visual Refinement. Editing is not the primary modeling task. Positioning of entities represents information and helps the user to think, but may not be the main purpose of modeling. Common functions supported by image editing applications, like aligning and resizing objects, are not considered as core features of a modeling tool but could be of vital importance for modeling as communication.
- System Hints and Guidelines. To help first-time users learn the grammar and rules of a certain model, the system could provide a guiding tool with graphic examples. However, it is also important to avoid interrupting users. Most users did not want to be disturbed by system hints and messages. However, they also said they would like a grammar verifier to help them identify modeling errors.

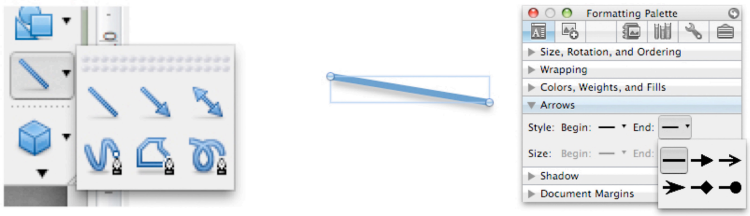

Fig. 7. Two ways to draw an arrow in common editing tools. Microsoft Word allows drawing an arrow directly with the drawing tool (A) or by drawing a simple line first, and then changing the style by adding an arrowhead from 'Formatting Palette' (B).

5 Modeling Approaches

The two modeling strategies from the pilot study, sequential and non-sequential, were both observed in the tests. As summarized in Fig. 6, three users took the sequential approach in the text-based scenario and the non-sequential approach in the sketch-based section. However, one user employed a non-sequential approach in the text-based section. Another used a sequential approach in the sketch-based test.

Fig. 8 shows the consolidated work models of sequential and non-sequential approaches. In the sequential approach, since connections are made after creating an entity, it becomes more difficult to change the type of any entity afterwards, which would require reconsidering all related connections. Moreover, it can take more steps to annotate existing relationships, not having considered meanings and direction.

In the non-sequential approach, the process is more complex. Users create a number of entities at once and consider their types and meanings afterwards. Later, they think about possible connections and the direction and specific meaning of

relationships. To one user, creating links was an independent part of modeling and a well-defined task. In the non-sequential approach, the frequency of correcting links and changing their types would be lower than in the sequential approach.

According to the interviews, the rules of a modeling language and its graphic notation directly influence the way users select their modeling strategy. The pilot study indicated that models like flowcharts, sequence diagrams, and activity diagrams in UML, incline users to use a sequential approach. In contrast, use case diagrams and component diagrams in UML, which do not represent workflow or procedure, encourage non-sequential approaches. Testing revealed that personal preferences and habits also influence selection of modeling approaches. For instance, users who sketch before using modeling software and users expecting efficiency may prefer a sequential approach independent of the form of models. Users who consider modeling applications as tools for helping them think tended to use a non-sequential approach.

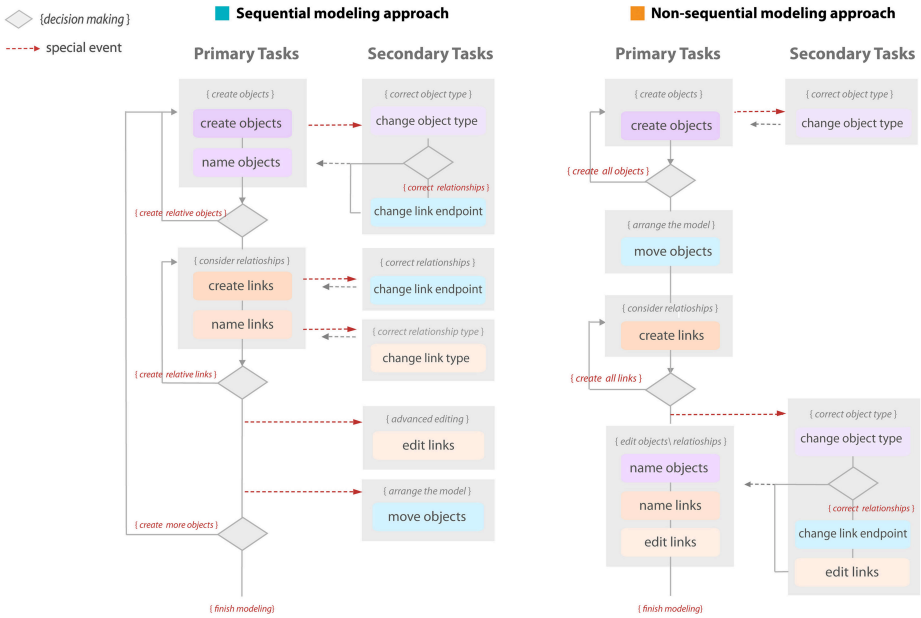

Fig. 8. Consolidated work model of modeling. In the sequential approach, the primary tasks are creating entities and links at once. In the non-sequential approach, users tend to create entities first and then consider relationships among those entities.

6 Design Implications

To design the information architecture and interface layout of a modeling tool, we argue that two different approaches and levels of user expertise should be considered. For instance, less-experienced users are drawn by strong affordance to the Object

Library to create new entities rather using the hidden feature of context menus that provide the same functions. Users favoring a sequential approach will not use context menus if unaware of them. For another example, a so-called sticky tool inclines users to create all the entities of one type at once for efficiency, which might lead users favoring a sequential approach to create models by the non-sequential approach.

The interaction design for primary and secondary task is directly influenced by the modeling approaches. We propose these design implications for modeling tools:

1. Creating Entities. For both approaches, a context menu for directly creating new entities on the canvas is essential. Our tests showed that not all users are comfortable with accessing this menu through right click. We suggest two additional ways to inform users of the functionality: (i) a screen tip the pointer hovers on the blank canvas and (ii) a ghost menu while the pointer hovers over a clickable object on the canvas for a few seconds.
2. Naming Entities. For both approaches, prompting users to name entities as created can reduce work in advanced editing steps. We suggest that once an entity is created, the name text field should be highlighted and have keyboard focus to enable straightforward insertion.
3. Changing Entity Type. Changing the type of an entity is a frequent task, and grammatically incorrect links and incorrect relationships may occur as a consequence. However, the system should not automatically revise or delete them. Modeling tools should guide users and help them identify and correct errors by highlighting incorrect relationships or providing a grammar checker.
4. Creating Links. Interaction design for creating links in support of both approaches is complicated. For the sequential approach, the efficient way is to let the modeling tool automatically create a link when a user creates an entity from another. For the non-sequential approach, creating links always combines two steps: selecting the source entity and selecting the target. If a modeling tool manages to support both approaches, it could allow users to change the previous target entity into the new source entity by using a hot key, which reduces the process by one step. In addition, users argued that modeling tools should allow users to create a temporary link without selecting a target entity.
5. Annotating Links. Annotating links and naming entities should be compatible processes, such as, by highlighting the text field of a link once created. Even though annotating links is not as frequent a task in most modeling situations, it should be easily accessible without interfering with the user's primary work focus.
6. Link Direction. Editing the direction of a relationship should be straightforward but also reflect the semantics of the model. If the direction of a link is not only a part of the notation for modelers but also contains information and presents a certain relationship type with semantics, a modeling tool has to help users of all levels of expertise understand this difference.
7. Inserting Entities. Allowing users to insert an extra entity into a link between two objects is important, especially for the sequential modeling approach. The function can be included in the context menu of editing links. Advanced interactions like dragging an entity and then attaching it into a link can also be considered.
8. Hints and Guidelines. Graphic images in grammar examples help users understand the rules of the modeling language (Fig. 9). Tools should make hints and guiding windows accessible but avoid distraction. Progressive (or cascading) screen tips

[54] offer simple object identification on hover, as with conventional screen tips, but more information and a link into the help system after an additional delay.

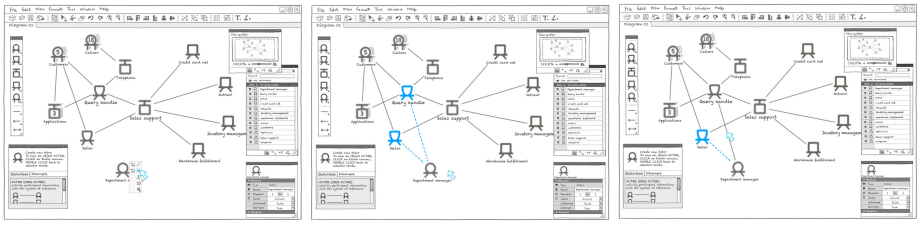

Fig. 9. Design concepts for guiding tools. To help users create links, grammar-based guiding hints (blue-dashed lines) and graphic examples (bottom left window) were proposed by users. Most agreed that graphic examples might help but that hints on the canvas interrupted work.

In modeling tools, simple operations like copy-and-paste or drag-and-drop take place in very complex situations. Since a model can be represented by more than one diagram and users may edit different models at the same time, to copy (or drag) an entity (or a group of entities) from one diagram to another might indicate two different situations. Either the user is trying to duplicate a concept for representing different instances, or the user actually wants to represent the same concept in different diagrams or to import a concept for the same instance into different models. Modeling tools need to remind users of the exact meaning and results of these interactions. An entity explorer and navigator need to clearly convey how components in diagrams and models are related. Finally, the tool should also provide users different viewpoints for merging and slicing the model.

Other important advanced functions, which should be considered for future work, are enabling users to use different graphical representations for the same instance and allowing users to merge different entities together. In our interviews, we found that these two functions are not well supported by current modeling tools, but are required when users try to clarify and reduce the complexity of models. To facilitate advanced editing, modeling tools could also allow users to group entities and components by separating them into different layers.

7 Conclusion

We present an iterative process to help improve tools for both domain-experts and software engineers define modeling languages. We consider a modeling tool as a workplace that helps users think, organize thoughts, and review concepts. We suggest several specific components should be considered in future modeling tools.

Through the contextual research described in this paper, we identified important primary and secondary modeling tasks. From this initial categorization of frequent modeling tasks, we presented and detailed the concerns and requirements of users with different levels of expertise. In addition, two modeling approaches were identified, and users' decisions to use either of these approaches were found to be

based on the type of model considered as well as personal preferences. Based on this contextual analysis, we propose a different interaction design to support the primary and secondary tasks of the two modeling approaches. We argue that the interaction should support different workflows and different mental models. Design implications for modeling tools are identified that support enhanced efficiency, advanced functionality, situated usability, and varying needs of modelers.

A modeling tool serves as a communication platform among domain-experts, designers, developers, and others practitioners. Through a case study of MetaSketch Editor, we argue for better understand of the diverse purposes of using a modeling tool with different work styles. Future research will continue to evaluate remaining parts of the modeling tool, including the platform for designing concrete syntax and querying the models.

Acknowledgments
Special thanks to those who participated in this research, shared their experiences, and offered excellent opinions on the design issues.

References

1. Myers, B., Hudson, S., Pausch, R.: Past, Present and Future of User Interface Software Tools. Computer-Human Interaction, vol. 7, no. 1, 3--28 (2000)
2. Jarzabek, S., Huang, R.: The Case for User-Centered CASE Tools. Communications of the ACM, vol. 41, no. 8, 93--99 (2004)
3. IIvari, J.: Why are CASE Tools not Used? Communications of the ACM, vol. 39, no. 10, 94--103 (1996)
4. Whittaker, J.A., Voas, J.M.: 50 Years of Software: Key Principles for Quality. IEEE IT Pro (Nov/Dec), 28--35 (2002)
5. Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software, vol. 20, issue 5, 19--25 (2003)
6. Hailpern, B., Tarr, P.L.: Model-Driven Development: The Good, the Bad, and the Ugly. IBM Systems Journal, vol. 45, issue 3, 451--462 (2006)
7. Ward, M.P.: Language-Oriented Programming. Software, Concepts and Tools, vol. 15, no. 4, 147--161 (1994)
8. Thomas, D.A.: MDA: Revenge of the Modelers or UML Utopia? IEEE Software, vol. 21, no. 3, 15--17 (2004)
9. Cuadrado, J.S., Molina, J.G.: Building Domain-Specific Languages for Model-Driven Development. IEEE Software, vol. 24, no. 5, 48--55 (2007)
10. Kleppe, A.: The Field of Software Language Engineering. In: Gaševic, D., Lämmel, R., van Wyk, E. (eds.), Software Language Engineering, LNCS, vol. 5452, pp.1--7. Springer, Heidelberg (2009)
11. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason IV, C., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The Generic Modeling Environment, Proc. Workshop on Intelligent Signal Processing, Budapest, Hungary (2001)
12. MetaCase, February 2011, http://www.metacase.com
13. Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, John Wiley & Sons (2004)
14. Budinsky, F., Steinberg, D., Ellersick, R., Merks, E., Brodsky, S.A., Grose, T.J.: Eclipse Modeling Framework. Addison Wesley (2003)

15.Miller, J., Mukerji J. (eds.): MDA Guide Version 1.0.1, OMG, Framingham, Massachusetts, June (2003)
16.Strauss, A., Corbin, J.: Basics of Qualitative Research. Sage, Newbury Park, CA (1990)
17.Charmaz K.: Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis. Sage, Thousand Oaks, CA. (2006)
18.Eisenhardt, K.: Building Theories from Case Study Research. Academy of Management Review, vol. 14, no. 4, 532--550 (1989)
19.Dey I.: Qualitative Data Analysis: A User Friendly Guide for Social Scientists. Routledge, London (1993)
20.Campos, P., Nunes, N.J. Practitioner Tools and Workstyles for User-Interface Design. IEEE Software, January/February, vol. 24, no. 1, 73--80 (2007)
21.Löwgren, J., Stolterman, E.: Design Methodology and Design Practice. Interact, 13--20 (1999)
22.Gerdenryd, H.: How Designers Work: Making Sense of Authentic Cognitive Activities. Lund University Cognitive Studies, Lund University, Sweden (1998)
23.Schön, D.A.: The Reflective Practitioner. Basic Books, New York (1983)
24.Carlile, P.: A Pragmatic View of Knowledge and Boundaries: Boundary Objects in New Product Development. Organization Science, vol. 13, 442--455 (2002)
25.Kursat Ozenc, F., Kim, M., Zimmerman, J., Oney, S., Myers, B.: How to Support Designers in Getting Hold of the Immaterial Material of Software. In: 28th International Conference on Human Factors in Computing Systems, pp. 2513--2522. ACM press, Atlanta, Georgia, USA (2010)
26.Myers, B.A., Park, S.Y., Nakano, Y., Mueller, G., & Ko, A.: How Designers Design and Program Interactive Behaviors. In: 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, pp.177-184. IEEE press, New York (2008)
27.Norman, D.A., Draper, S. (eds.): User Centered System Design: New Perspectives on Human-Computer Interaction. Erlbaum, London (1986)
28.Kensing, F., Blomberg, J.: Participatory Design: Issues and Concerns. Computer Supported Cooperative Work, vol. 7, 167--185 (1998)
29.Nielsen, J.: Usability Engineering. Academic Press, San Diego, CA (1993)
30.Woods, D.D.: Process-Tracing Methods for the Study of Cognition Outside of the Experimental Psychology Laboratory. In: Klein, G.A., Orasanu, J., Calderwood, R., Zsambok, C.E. (eds.) Decision Making in Action: Models and Methods, pp. 228--251. Ablex, Norwood, NJ (1993)
31.Millen, D.: Rapid Ethnography: Time Deepening Strategies for HCI Field. In: ACM Symposium on Designing Interactive Systems, pp.280--286, ACM Press, New York (1993)
32.Wharton, C., Bradford, J., Jeffries, R., Franzke, M.: Applying Cognitive Walkthroughs to More Complex Interfaces: Experiences, Issues, and Recommendations. In: ACM CHI '92 pp. 381--388, ACM Press, Monterey, California (1992)
33.Holtzblatt, K., Jones, S.: Contextual Inquiry: A Participatory Technique for System Design. In: Schuler, D., Namioka, A. (eds.), Participatory Design: Principles and Practices, pp. 177--210. Lawrence Erlbaum, NJ (1993)
34.Mayhew, D.J.: The Usability Engineering Lifecycle: A Practioner's Handbook for User Interface Design. Academic Press, San Diego, CA (1999)
35.Buxton, B.: Sketching User Experiences: Getting the Design Right and Right Design. Morgan Kaufmann Publishers, San Francisco, CA (2007)
36.Wong, Y.Y.: Rough and Ready Prototypes: Lessons from Graphic Design. In: Human Factors in Computing Systems: CHI'92, pp.83—84. Monterey, CA (1992)
37.Landay, J.A., Myers, B.A.: Interactive Sketching for the Early Stages of User Interface Design. In: SIGCHI Conference on Human Factors in Computing Systems, pp.43--50, Denver, Co (1995)

38.Bailey, B.P., Konstan, J.A., Carlis, J.V.: Supporting Multimedia Designers: Towards More Effective Design Tools. In: 8th International Conference on Multimedia Modeling, pp.267--286 (2001)
39.Gibson, J.J.: Reasons for realism: Selected essays of James J, Gibson. Reed, E., Jones, R. (eds.), Erlbaum, Hillsdale, New Jersey (1982)
40.Norman, D. A.: Psychology of Everyday Things. Basic Books, NY (1988)
41.Blomberg, J., Giacomi, J., Mosher, A. Swenton-Wall, P.: Ethnographic Field Methods and Their Relation to Design. In: Schuler, D., Namioka, A. (eds.), Participatory Design: Principles and Practices, pp. 123--156. Lawrence Erlbaum, NJ (1993)
42.Creswell, J.W.: Qualitative Inquiry and Research Design: Choosing Among Five Traditions. Sage, London (1998)
43.Lincoln, Y.S.: Emerging Criteria for Quality in Qualitative and Interpretive Research. Qualitative Inquiry, vol. 1, 275--289 (1995)
44.Lincoln, Y.S., Guba, E.G.: Naturalistic Inquiry. Sage, Beverly Hills, CA (1985)
45.Kirwan, B., Ainsworth, L.K.: A Guide to Task Analysis. Taylor and Francis, London (1992)
46. Benyon, D.: Task Analysis and System Design: the Discipline of Data. Interacting with Computers, vol. 4, no.1, 246--249 (1992)
47.Ericsson, K.A., Simon, H.A.: Protocol Analysis: Revised Edition. MIT Press, Cambridge, MA (1993)
48.Cordingley, E.S.: Knowledge Elicitation Techniques for Knowledge-Based Systems. In: Diaper, D. (ed.), Knowledge Elicitation: Principles, Techniques, and Applications, pp. 89--175. John Wiley & Sons, NY (1989)
49.Maulsby, D., Greenberg, S., Mander, R.: Prototyping an Intelligent Agent through Wizard of Oz. In: SIGCHI Conference on Human Factors in Computing Systems, pp. 277--284. ACM press, NY (1993)
50.Scott, A.A., Clayton, J.E., Gibson, E.L.: A Practical Guide to Knowledge Acquisition. Addison-Wesley Longman Publishing Co, MA (1991)
51.Constantine, L. L.: Human Activity Modeling: Toward a Pragmatic Integration. In: Seffah, A., Vanderdonckt, J., Desmarais, M.C. (eds.), Human-Centered Software Engineering: Software Engineering Models, Patterns and Architectures for HCI, pp. 27--51. Springer, London (2009)
52.Kaenampornpan, M.: A Context Model, Design Tool and Architecture for Context-Aware Systems Designs. Ph.D. thesis, University of Bath (2009)
53.Beyer, H., Holtzblatt, k.: Contextual Design: Defining Customer-Center Systems. Morgan Kaufmann Publishers, San Francisco (1998)
54.Constantine, L.L., Lockwood, L.: Instructive Interaction: Making Innovative Interfaces Self-Teaching. User Experience, Winter, 14--19 (2002)