Group Exercises for the Design and Validation of Graphical User Interfaces

Harald Störrle

Ludwig-Maximilians-Universität München
stoerrle@informatik.uni-muenchen.de

Abstract: This paper presents a simple method for the initial design and validation of graphical user interfaces by group exercises based on brainstorming and storyboarding with paper prototypes. It requires no technical skill on behalf of the participants, and may thus be used with end users. The method helps to initiate a group process and motivate them—by getting these people involved early, the acceptance of the overall development effort may be improved. The approach also proposes a syntactic variant of UML state transition diagrams as a User Interface Design Language (UIDL).

1 Introduction

1.1 Motivation

For most end users, the GUI is all they ever see of an information system. As far as they are concerned, the surface is all that matters. It is the screen they yell at, not their computer. So, when it comes to getting users to become personally involved, the GUI ist the right place to start at.

This is an important observations, since, frequently, users are often not exactly helpful during development: they may see the development as just another waste of time sent upon them by management, or they might even feel threatened by the impending changes to their working environment. Hostile or indiferent users can pose a serious threat to the overall success of a development effort. The best way to overcome such obstacles is getting the users personally involved in the development process. Since users have no understanding of the notations software engineers use, paper prototypes (visual mock ups) of the system to be is a viable way of communication between the two groups: on the one hand, designing a user interface is to a considerable degree understanding and eliciting the user‘s requirements. On the other hand, designing a user interface can be used as a device to attract the user‘s interest and involvement. This paper presents a method (i.e. notations and associated techniques) that can achieve this by means of group dynamic exercises yielding an initial design of a GUI for an information system.

1.2 Approach

Now look at different ways of creating a GUI-design.[1] Firstly, one may create an (explorative) prototype, that is, simply start coding the GUI. This way, all possible GUI-features are available at once, but creating such a prototype is rather expensive, and takes considerable time. There are only small numbers of users involved, if any at all. During the development, the amount of technical and graphical detail distracts the developer's—and the user's—attention from the purpose proper, namely, understanding the requirements and documenting them in a way the user understands them. Also, an explorative prototype usually can not be taken "as is" to be included in the final product.

So, secondly, one may use a GUI-builder instead. Here, much less effort has to be spent on creating the prototype, though there is still an unwanted focus on the graphical appearance. Using a GUI-builder is still a task for a skilled expert: only small numbers of users can participate, and it's a mostly passive role they play: it is very clumsy and tedious to modify a prototype over and over again interactively during a group discussion *by and with* the end-users. Assessing or integrating alternatives this way is virtually impossible.

Thirdly, thus, one may use so called paper-prototypes, that is, sheets of paper with more or less elaborated sketches of windows and menus on them. An end-user may then "select" or "drag" by saying so and finger-pointing, and a developer will reply by, say, replacing one sheet with another, giving some further explanations on-line. This technique is well suited for fast, interactive changes of a GUI-design, and it is simple enough for users to directly interact with it, and make their own contributions. Using overhead slides, it is also usable for interactive discussions with larger audiences. However, the dynamic aspect is not explicitly represented here—it exists only in the head of the developer or in ad-hoc annotations. The precise meaning of these tends to get lost in no amount of time. So, the transition from the design to the implementation is again open to interpretation.

Here, my approach comes in: I interpret a GUI-design as a (syntactical extension of) UML state-transition diagram, whose states are windows or menus and whose transitions are actions and messages. So, the benefits of paper prototypes are combined with the semantical framework of the UML. The dynamic aspect is captured unanimously. Such a GUI-design can be used directly as a blueprint for the implementation proper, for its validation, testing and for the creation of a manual.

Most important however (and that is the contents of this paper) are the techniques based on this notation, that is, the way of creating such a specification interactively. The approach is very simple indeed, and can be understood by almost any user. Large audiences can actively contribute, evaluate and compare concurrent designs interactively, ensuring a higher degree of involvement, and thus acceptance of the resulting design.

1.3 Related work

A method contains notations and techniques. Concerning notation, there are of course many user interface design languages (UIDLs) based on state machines (cf. [Jac86, FL89, Mye93]). Some UIDLs use hierarchical models (e.g. [Wel89]), and some follow a story-

[1] In the remainder, we will only consider **W**indow**I**con**M**enu**P**ointer-style graphical user interfaces (GUIs).

boarding approach with graphical elements as states/transitions (cf. [Lan96] and preceding papers). These approaches focus on the design *language* (the notations) rather than on the design *process* (the techniques), however.

But there are also contributions dealing with the latter issue (cf. [Bro97, MS00, Woo98]), or both issues (cf. [Was85] and works cited there, though this approach features only a classical, non-participatory process).

My approach, in contrast, defines a graphical UIDL based on UML state transition diagrams and an aligned participatory process using group exercises. The approach is targeted at the very early stages and serves to

- initiate a group dynamic process among larger groups of non-experts (e.g. end-users);
- thereby getting them involved in the overall development effort;
- eliciting elementary requirements from their point of view en route;
- and create a low-fidelity GUI prototype understandable to them.

The approach presented here is first described in [Stö00].

2 Notations: Window/Event Diagrams

A method consists of notations and techniques. I will introduce the Window/Event-diagram (WED) notation in this section, and the associated techniques in the next.

2.1 Concrete Syntax

The concrete syntax consists of two parts. First, there is simple system of sketches for basic GUI elements such as buttons, sliders, input and output boxes. Most of them are entirely self-explanatory: in Figure 1 I have given some examples (the inscriptions carry no significance). Starting from the top left and proceeding clockwise, there are

- a main window with two menus and a slider;
- a menu with four entries;
- a set of tabbed panes (note the use of concurrent regions);
- two different presentations of the same radio button with five values;
- a confirmation dialog;
- a menu with a pair of radio buttons, an optional input box, and three check buttons.

These GUI-elements are the static part of the notation. They are connected by arcs that show in which sequence the windows and menus may be accessed by the user. The arcs are inscribed by icons and text indicating the triggering event, a guard condition, and an effect (all of these are optional). In Figure 2, there are five kinds of transitions (top to bottom, then left): one triggered by a click with the left mouse button; one triggered by the escape-key, but only if the focus is in the appropriate window; one triggered by a time-out and resulting in a beep; one showing a drag-and-drop action (the icons stand for holding the right button depressed, and releasing it); and finally a doule click with a side effect. Instead of the icons for the escape-key and the mouse-buttons, textual abbreviations are often handier, e.g. LC for left-click and so on.

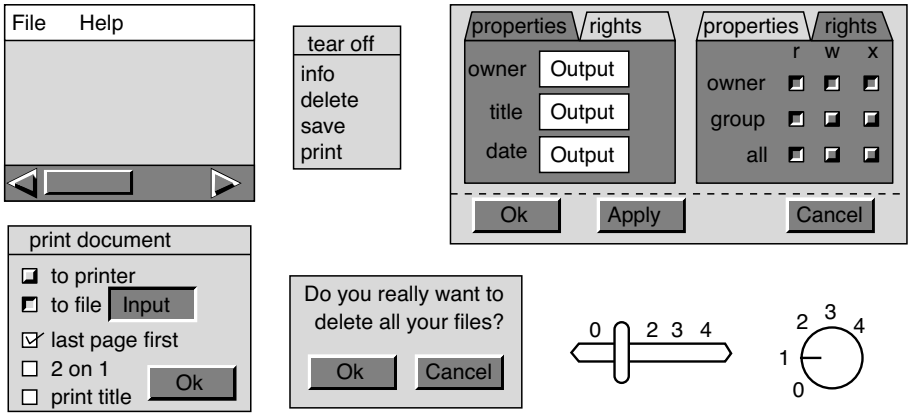

Figure 1: Examples for syntactic representations of ≪GUI-state≫ States.

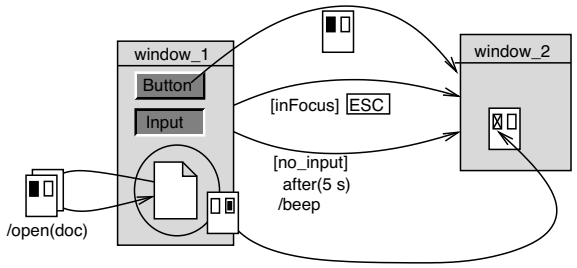

Figure 2: Examples for syntactic representations of ≪GUI-state≫ States.

2.2 Abstract syntax and semantic mapping

So far, WEDs are just a very intuitive, systematic way of describing paper-prototypes of GUIs. Most users understand them with very little explanation. For the techniques presented below, this is already enough. However, for the transition to implementation (that is, for programmers and for tool support), a more precise semantic definition is called for.

This is achived by providing a mapping from WEDs to UML state transition diagrams, or, to be more precise, from the concrete syntax of WEDs to the abstract syntax provided with the UML [OMG01]).[2] The main idea is to identify windows, menus, and menu entries as States. Then, user actions like mouse clicks or text input are Events that may trigger a Transition, resulting in messages, pop-up menus or new windows etc. as its effect. With this mapping, all the static elements shown in Fig. 2 map to States, and all the arcs in Fig. 1 map to Transitions.

This mapping is really slick – it is straightforward, and preserves both the intuitive meaning and the UML semantics. As a specialty, observe the use of a ConcurrentState for separating the dynamic and static parts of a set of tabbed panes in Fig. 1.

[2]When referring directly to elements of the UML Metamodel I use the helvetica font (e.g. StateMachine and StateVertex).

3 Techniques: Group exercises

After having introduced the notations, in this section, I will now turn to the techniques using them.

3.1 Step 1: Brainstorming and refining

As a first step, a representative sample of users is gathered for a brainstorming, moderated by the GUI developer (called moderator): in its first phase (the collection-phase), everybody may freely propose windows, menus, and mechanisms that are collected on a pin-board or similar.[3] Each proposal consists of a sheet of paper with a raw sketch. After some time, the collection phase is declared finished by consensus (or the moderator).

Now, an organization phase ensues. First, the proposals are grouped for similarity. Typically, there will be one or two proposals in a group that clearly dominate the others, and a few additional ideas in other proposals that are considered worthy. Then, the dominating proposals (and the scattered ideas) of each group are integrated into a single one for each group. If no consensus on the right solution can be reached, one may be determined by voting. During this process, small additions may be made to the proposals as needed. As the second step of the organization phase, the proposed windows, menus, and so on are connected by arcs that are then inscribed with triggers, guards, and effects.

Sometimes, what is on the blackboard right now is a rather coarse, incomplete, and possibly not even a proper WED. If that is case, it is necessary to have a third phase to refine it sufficiently. For instance, pop-up menus, key-shortcuts and additional elements and requirements imposed by a style guide may have to be considered.

3.2 Step 2: Validating design

Now we have reached a design of the GUI, including both the static and the dynamic aspect. It has to be validated by sample users, which may be either the same group that the first phase was conducted with, or, preferably, by another group. The validation can be done by an interactive white-board-simulation. All one needs is the WED on the white-board and a magnetic sticker to mark the window on which the focus is. The moderator marks the initial state, and someone from the audience suggests some action to be taken which is then executed by the moderator, that is, he announces what happens, and possibly moves the focus. If some inconsistency or shortcoming of the design is detected during the simulation, it may be fixed at once, on the white-board.

The qualities of a design may be determined by a dot-ballot. This is particularly useful, when there are competing designs that must be voted on. Suppose, the dimensions usability and functionality of three competing designs (called blue, red, and yellow) are to be determined. For each of these, a coordinate system is drawn on a flip-chart. Each participant gets one sticky dot of each of the three colors. Then, all participants go to the flip

[3] I propose you use a magnetic and writable white-board for this step: magnetic in order to attach sheets of paper in a movable way (or use A4 Post-Its), and writable for adding the arcs in the second phase, see below.

charts simultaneously and put their dots on the appropriate diagram. The result may look something like Figure 3. The general opinion on the respective qualities of the competing designs becomes very obvious this way.

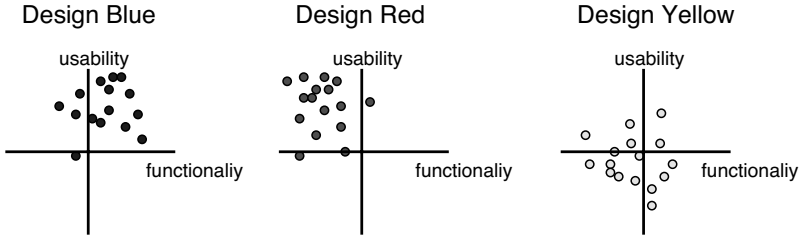

Figure 3: Determining different qualities of competing designs: three competing designs labelled blue, red and yellow.

3.3 Step 3: Creating a prototype

At this point, we have decided on a design laid down precisely as a WED. The design has been validated by a group of users, and a fairly high level of acceptance among those users has been ensured by a direct involvement (and usually a lot of fun). We (as developer) are now faced with the task of transforming this design into a (production) prototype.

This task involves mainly two aspects: those questions related to the graphical design (layout, visual effects, fonts, sizes, styles and so on), and those questions related to connecting the GUI-logic to the application logic. The GUI-logic, on the other hand, has already been defined by the WED (and the UML semantics for StateMachine). So, the GUI may be generated using a GUI-builder right away. In the example presented in the next section (see Figure 6), a few hints concerning the connection of GUI and application have already been included: for toy applications, and for explorative prototypes, this may already suffice.

3.4 General advice

In general, all issues and measures known from the literature on group exercises apply directly to the method I have presented, that is, dealing with shy or heckling participants, dealing with disruptions, obstructions and blockades, and concerning presentation techniques.

As with all group dynamic exercises, there are the distinct roles of moderator and scribe. Only for groups smaller than ten to six people the two roles may be played by the same person. Switching roles during a workshop only as a last resort to resolve personal animosities. It is vital, that the moderator leading the workshop has thorough training an experience moderating large groups.

As a rule of thumb, steps 1 through 3 should take between 2 hours and 2 days, with a 15-minute-break at least every two hours. The exact duration depends on several factors:

the group size, the exposure the participants have a had to the application domain, and the scope of functionality to be covered.

The most important thing to keep in mind, however, is that the whole exercise is supposed to be fun for the participants, and almost any device fit for this purpose is acceptable.

4 Example

I have so far presented a simple extension of UML state-transition-diagrams on a fairly abstract level. To make this all more concrete, consider now an example. Admittedly, this is a toy example and somewhat artificial, but it is abstracted from a real experience, and presenting a more detailed example would exceed the space available here.

4.1 Problem

Suppose that we are to implement an information system for course results at some university. The conceptual model of the problem would look something like Figure 4.

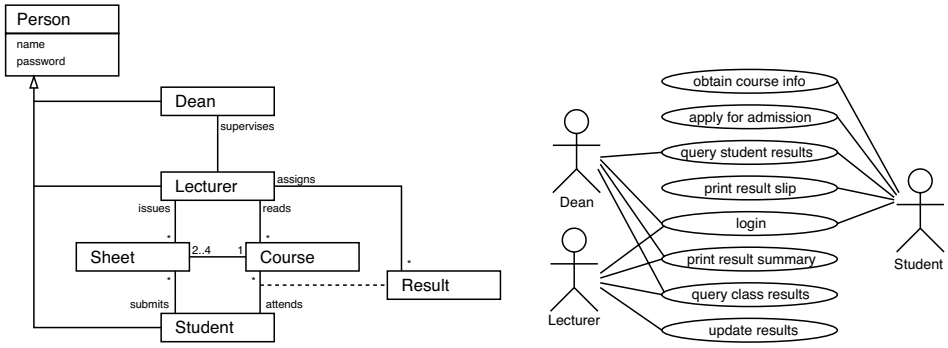

Figure 4: The underlying conceptual model in an UML-style static structure diagram (left); the functionality of the system in an UML-style use case diagram (right).

There are the usual persons with some attributes, there are courses to be read by some lecturer, and to be attended by students. In each course, there are between two and four sheets with exercises to be solved and submitted by the students attending that course. The lecturer issues these sheets, collects them again, and assigns results. There is a number of functionalities we would like to see realized in our toy information system, as summarized by the use case diagram in Figure 4.

These functionalities are further specified by a number of "scenarios" (or "stories"):

- First of all, there has to be a login step, in which the role an capabilities of a user are determined.
- Before the term starts, students may obtain information on offered courses, and may apply for admittance to some of them. The lecturer admits students based on their prior courses and/or grades.

- During the term, the lecturer updates the results database, which may be queried by the students. The lecturer may also query the results of individual students or the whole class, for instance, to prepare a report for the grim dean, who supervises our poor lecturer, and who is particularly keen on all students passing the exams no matter what.
- After the term, the students may print a result slip, either for their records (or parents or funding institution), and as an admission ticket for the final exam.

So far, we have described the requirements from the client, ordering the information system. This information is given to the developers, and they have to go on from here. Since the system is to be used by students (and staff) of all faculties, general computer literacy may be assumed, but no more than that. So assume for the sake of argument, that the conception, design and validation of the system is started by a GUI prototype using the technique described above.

4.2 A sample solution

In the first step, prospective states are proposed, collected and systematized. The first menu-paths are sketched by some arcs, resulting in a graph of windows and user-activities (see Figure 5).

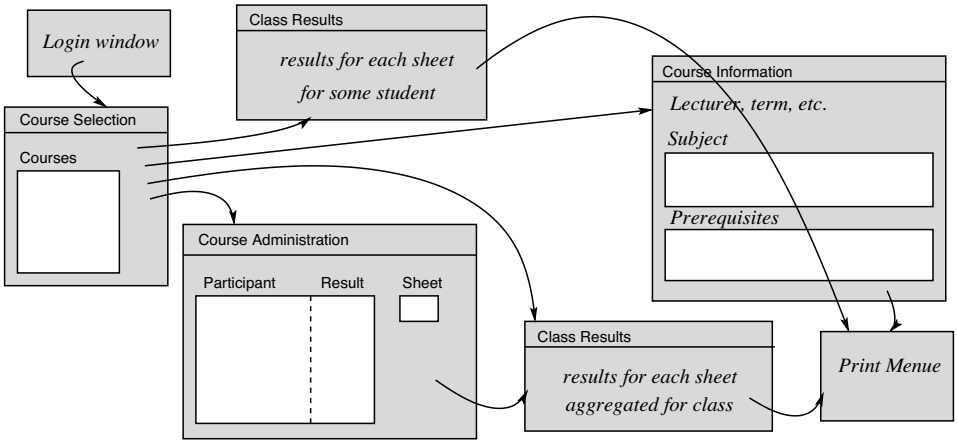

Figure 5: A first draft of the GUI, containing only the main windows and arcs.

Note that so far, almost no graphical detail has been put into the design. This design may now be refined into a proper WED, by spelling out the logical details of the windows, adding arcs and menus, and refining the interaction. For esthetic reasons, I have also added a little layout and graphical make-up—this is nice to have, but, at this stage, not essential. The result is shown in Figure 6

The second step is omitted here—it should be clear, anyway. In the third step, a prototype is created after the WED-blueprint. This is very easy indeed. In Figure 7, I have shown screenshots of the first two windows implemented straight after the diagram.

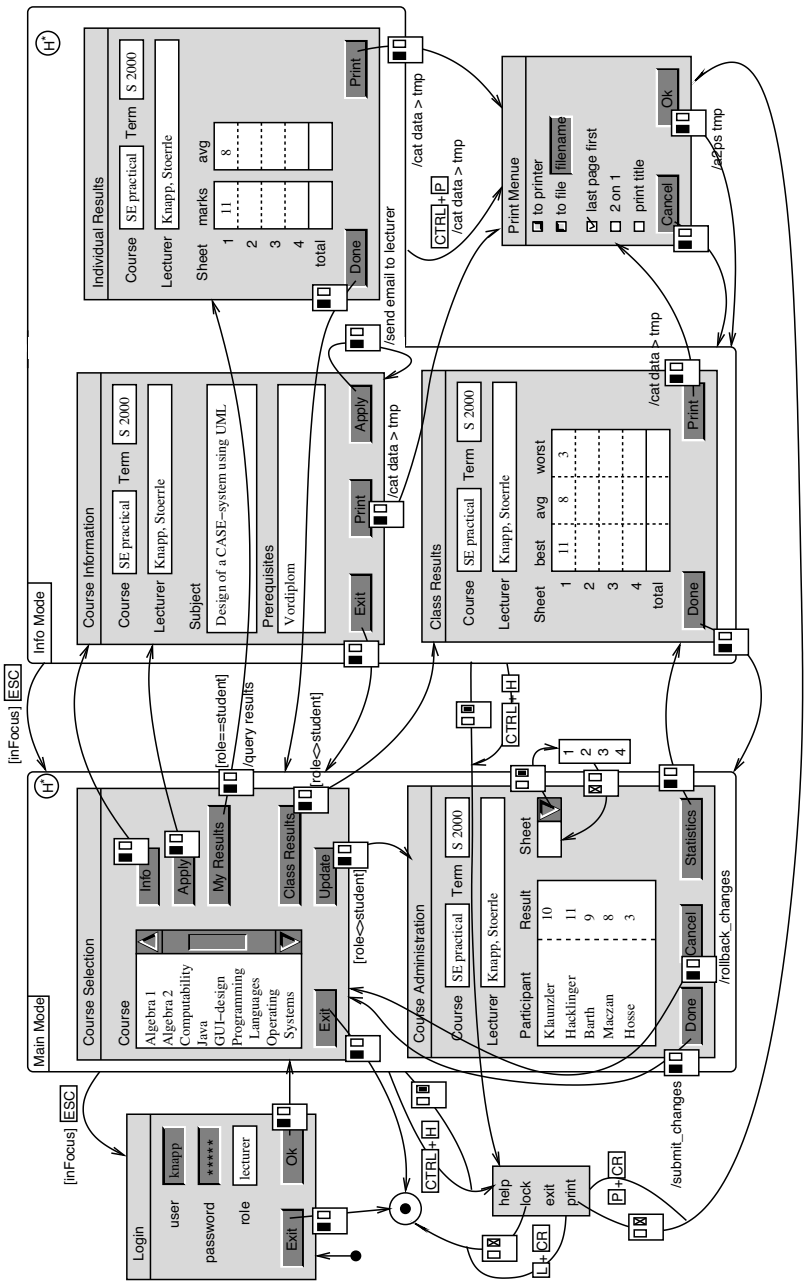

Figure 6: A complex WED as the design for the GUI of the course information system.

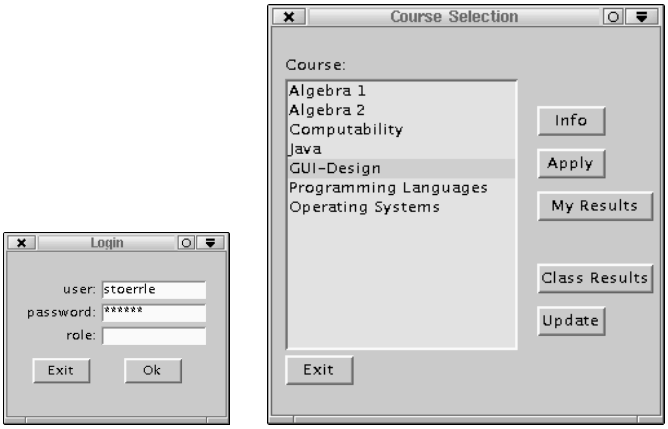

Figure 7: Screenshots of the first two windows coded using Swing.

5 Discussion

5.1 Summary and Contributions

It is a truism, that "*many UI deficiencies arise because the UI design process is ad hoc and the design is not communicated successfully to the programmer*" (cf. [MS00, p. 17]) and that "*user participation is very important in the early stages of the methodology*" (cf. [Was85, p. 699]). Still, however, in many software projects, these issues are not addressed adequately. This paper tries to improve on this situation by a method comprising a UIDL called Window/Event-diagram (WED) and a set of techniques based on it.

The WED notations is a syntactic variant of UML state-transition-diagrams that deemphasizes technicalities like layout or interaction details. Instead, it helps the users focus on the logic and functionality instead. It can be used to compare and discuss several drafts in parallel by groups of people, which facilitates e.g. the joining together of competing approaches. Note also, that a state can easily be elaborated and refined, that is, it needs not be complete, entirely precise, or operational right from the start.

WEDs allow to present a GUI-design in a simple and visual way which is easily understood and applied by programmers and users alike. Programmers often find WEDs convenient, as it is (an extension to) a general purpose design language they (ought to) know anyway rather than a new, specialized notation. Thus, the transition from inception to design and implementation is improved.

The techniques provided are based on classical group dynamic exercises like brain storming. They need no computer support and allow to design GUIs interactively with large groups of non-experts. By getting these people involved in the development, this technique increases acceptance of the system.

The approach presented here is used regularly for trainings in industry and academia and has also been used occasionally for production purposes.

5.2 Objections

A number of objections have been raised against the technique and notation presented here. Let us look at these in turn.

- "The notation is restricted"—true. It is supposed to: this approach is about user involvement, it must not be feature-loaden. Otherwise, the notation would distract the attention from questions of content, and direct it towards questions of presentation.
- "There are other notations that are better suited/more precisely defined"—special purpose notations: yes (see Section 1.3), but not general purpose notations. And the UML *is* the most widely used design notation by far.
- "The UML stinks"—well, it does have a strange odor at times, but then, are there realistic alternatives?
- "The techniques are just folklore"—well, yes and no. They are to the experts in the (respective) fields. They are not to the majority of people working in the area. Look at this approach as a way to spread the "folklore". Besides, there are some nifty extensions, and they are nicely aligned with the notation.

Particularly the last objection is somewhat difficult to counter. Indeed, my approach is to some extent an application of established ideas and best practices. The contribution of the combination and the extensions is easily underestimated. Look at Extreme Programming (XP) for an analogy: the techniques summarized there are also "just folklore", and still the combination has turned out to be quite helpful in certain contexts.

5.3 Limitations

Besides all benefits of this approach, there are also some shortcomings. First of all, when trying to integrate *all* aspects of an interaction, WEDs soon become quite complex and difficult to understand at first sight. Also, it is not always possible to incorporate all aspects of a style guide during the brainstorming process—here, changes may become necessary later on.

6 Further Work

This approach is subject to ongoing work. First, though a number of practical experiences has showed great promise and subjective experience reports are unanimously positive, there are no objective findings to support this opinion. More controlled experiments are necessary to exactly determine the limitations of the approach: for which kind of groups (size, qualification) and applications (size, application type and domain) is it best? Is there really an objective benefit? If so, how large is it?

Then, and as a prerequisite to such studies, one would like to have an interactive visual simulator, and a code generator. Currently, there is a fully-fledged UML StateMachine simulator ready, but it only has a textual interface. Of course, one would want such a tool to be integrated with a GUI-builder to generate as much as possible automatically.

Finally, there are open questions concerning the integration of this approach with other techniques: are there any interferences or additional synergies with established methodologies like use cases?

Acknowledgments

Thanks go to Florian Hacklinger and Alexander Knapp, three anonymous referees, and Gregor Engels. Thanks go also Martin Glinz and his collaborators for incessant support concerning the proceedings format.

References

[Bro97] J. Brown. Exploring Human-Computer Interaction and Software Engineering Methodologies for the Creation of Interactive Software. *SIGCHI Bulletin*, 29(1):32–35, January 1997.

[FL89] B. Fraser and D. Lamb. An Annotated Bibliography on User Interface Design. *SIGCHI Bulletin*, 21(1):17–28, July 1989.

[Jac86] R. Jacob. A Specification Language for Direct Manipulation User Interfaces. *ACM Trans. Graphics*, 5(4):283–317, 1986.

[Lan96] J. Landay. *Interactive Sketching for the Early Stages of User Interface Design*. PhD thesis, Carnegie Mellon University, December 1996.

[MS00] P. McInerney and R. Sobiesiak. The UI Design Process. *SIGCHI Bulletin*, 32(1):17–21, January 2000.

[Mye93] B. Myers. Report on the CHI'91 Workshop on Languages for Developing User Interfaces. *SIGCHI Bulletin*, 25(2):20–23, April 1993.

[OMG01] OMG Unified Modeling Language Specification 1.4. Technical report, Object Management Group, February 2001.

[Stö00] H. Störrle. *Models of Software Architecture. Design and Analysis with UML and Petrinets*. PhD thesis, LMU München, Institut für Informatik, 2000.

[Was85] A. Wasserman. Extending State Transition Diagrams for the Specification of Human-Computer Interaction. *IEEE Trans. Software Engineering*, 11(8):699–713, August 1985.

[Wel89] P. Wellner. Statemaster: A UIMS based on Statecharts for Prototyping and Target Implementation. In K. Bice and C. Lewis, editors, *Proc. Intl. Conf. Computer-Human Interaction*, pages 177–182. ACM Press, 1989.

[Woo98] L. Wood. *User Interface Design: Bridging the Gap*. CRC Press, 1998.