

Design and Validation of Graphical User Interfaces Using Window/Event-diagrams

Harald Störrle

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

Abstract. This paper presents a syntactic variant of UML state transition diagrams suitable for the design and validation of simple graphical user interfaces. The notation does not change the UML semantics in any way. It is particularly well-suited to support collaborative style of work, needs no computer support, and is easily adopted by users without technical expertise. We provide a brainstorming-based technique to create GUI-designs using this notation interactively with large groups of non-experts like users. By getting people involved early on in the design process, the acceptance of the overall development effort is improved. The approach has been used in the classroom with great success. This paper is based on [7].

Keywords. UML, GUI modeling, methodology, brainstorming, state transition diagram, metamodeling

1 Introduction

1.1 Motivation

For users, many functional and non-functional requirements of an application are defined via the GUI. Thus, designing a GUI is to a considerable degree understanding the user's requirements. Or, to put it in other words, designing a GUI can be used as a device to elicit the user's requirements. However, quite frequently, users are often not exactly helpful: 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. The best way to overcome such obstacles is getting the users involved in the development process. Now look at different ways of creating a GUI-design.

Firstly, one may create an (explorative) prototype, that is, simply start coding the GUI. This way, all GUI-features are available at once, but creating it 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 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 error-prone.

1.2 Approach

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 in a standardized way. Such a GUI-design can be used directly as a blueprint for the implementation proper, for its validation, and for the creation of a manual. 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

In [1], a methodology for the design of complex hypermedia systems is proposed, whereas I deal with simple graphical user interfaces for ordinary systems.

A very detailed approach to integrate a body of HCI concepts into the UML is presented in [2]. The approach does not deal with methodological questions, and focuses on the design phase rather than on the requirements analysis, as I do.

Both [8] and [5] provide methodological support for their respective notational extensions. The former uses use case diagrams in a similar style as I use state transition diagrams in this paper.

All of the above approaches have a very technical focus, and are thus not suitable as a basis to integrate end-users into the design process, neither in their notations nor in their techniques or overall methodology.

2 Window/Event Diagrams

In the remainder, we shall often reference directly to elements of the UML Meta-model. To make such references clear without impairing readability, we use the helvetica font, e.g. for metaclasses like `StateMachine`. The main idea is to identify menus 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**. We will use the abbreviation WED for Window/Event-diagram.

2.1 Syntax

First I introduce some auxiliary notation for basic elements such as buttons, sliders, input and output boxes. It is assumed that their default meaning and behavior is well-enough understood by all users.

I introduce a new kind of `SimpleState` with the `«GUI-state»`-stereotype. Its syntactic representation may take on any form suitable, that is, a window, tabbed pane, menu, confirmation dialog or other, see Figure 1.

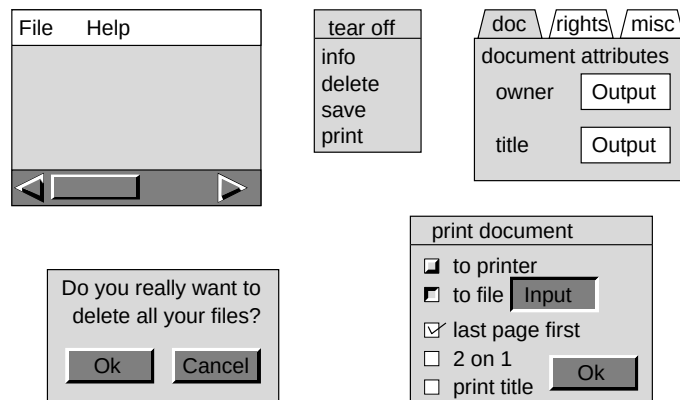


Fig. 1. Examples for syntactic representations of `«GUI-state»` States.

In Figure 1, I have given some simplistic, but typical examples. The inscriptions are irrelevant, they only serve to hint at possible applications. Now consider the four `«GUI-state»` in clockwise order starting from the top left. There are

1. a main window with two menus and a slider;
2. a menu with four entries;
3. a set of tabbed panes, the first of which presents two values;
4. a menu with a pair of radio buttons, an optional input box, and three check buttons;
5. a confirmation dialog.

All of these are states of the GUI, and directly correspond to `SimpleStates` in the sense of UML. If the basic elements are to be specified with a non-default semantics, one might use `concurrent CompositeStates` instead, with a region (that is, a `sequential CompositeState`) for those basic elements with additional semantics.

These drawings could be used directly as a paper-prototype. That is, this set of windows already captures the static aspect of a GUI-design. In order to also capture the dynamic aspect, these basic elements (that is, states) have to be connected with arcs. In the UML, these arcs are mapped to `Transitions`, featuring an `Event` as its trigger, an additional `Guard` as its guard, and an `Action` as its effect. For simplicity, only user actions and time-conditions are considered as triggers in the remainder. The guard may access variables from the context. The effect typically is a method call—in the remainder, it will be treated more or less like a comment. The notation we need to represent these is fairly straightforward (see Figure 2).

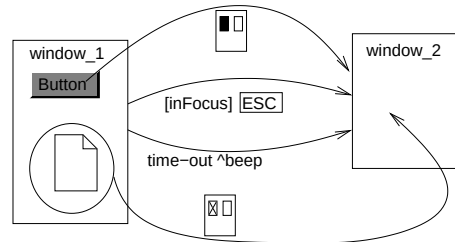


Fig. 2. Examples for syntactic representations of `<<GUI-state>> States`.

Figure 2 shows four kinds of transitions (top to bottom): one triggered by a click with the left mouse button; one triggered by the escape-key, but only if the focus is in the right window; one triggered by a time-out and resulting in a beep; and finally a drag-and-drop action. 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.

2.2 Mapping and semantics

The mapping from concrete to abstract syntax remains identical. Also, there are no changes to the standard `StateMachine`-semantics as laid down in the UML standard (cf. [3]).

Sometimes, a non-standard semantics is intended (e.g. additional logical constraints on sets of buttons, thresholds for sliders etc.). Then, this has to be specified explicitly, with the means introduced above.

2.3 Techniques

Step 1: Brainstorming and refining As a first step, a representative sample of users is gathered for a brainstorming (cf. [4]), 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.¹ 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 even unsyntactical 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.

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 are to be determined. For each of these, a cross as in Figure 3 (top) is drawn on a flip-chart, and each voter gets one sticky dot with the right color for each of the designs. Then, all participants go to the flip charts simultaneously, and put their dots wherever they like. The result may look something like Figure 3 (bottom). The general opinion on the respective qualities of the competing designs becomes very obvious this way.

¹ 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, and writable for adding the arcs in the second phase, see below.

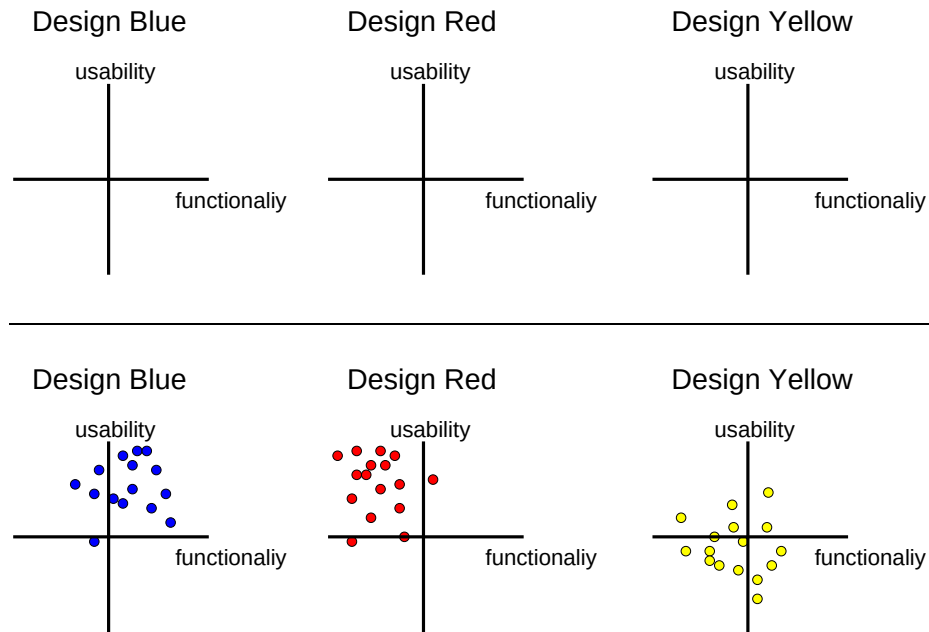


Fig. 3. Determining different qualities of competing designs: schemas before and after voting (top and bottom, respectively).

Step 3: Create 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 7), a few hints concerning the connection of GUI and application have already been included: for toy applications, and for explorative prototypes, this may already be sufficient.

3 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.

3.1 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.

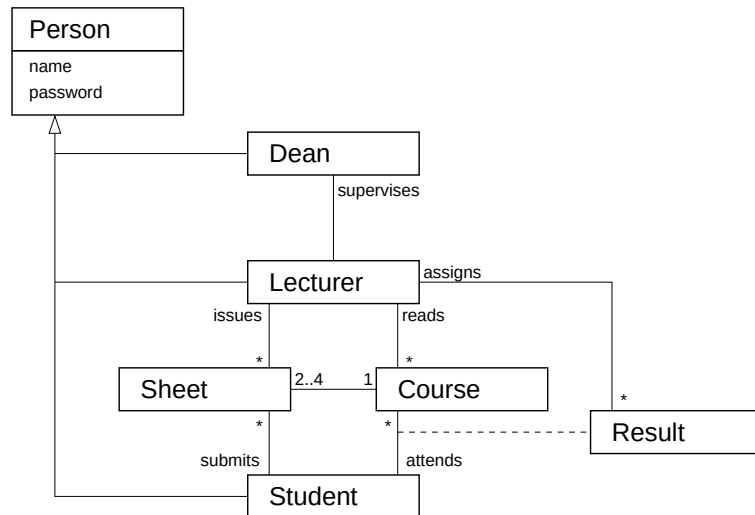


Fig. 4. The underlying conceptual model in an UML-style static structure diagram.

There are 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 5.

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 and 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.

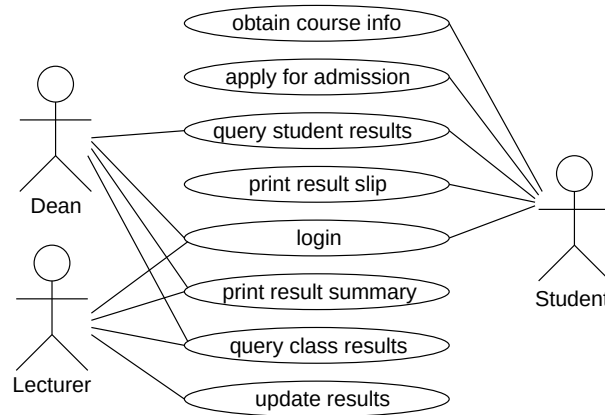


Fig. 5. The functionality of the system in an UML-style use case diagram.

- After the term, the students may print a result slip, either for their records (or parents), 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.

3.2 A sample solution

In the first step, prospective states are proposed, collected and systematized. The first menue-paths are sketched by some arcs, resulting in a graph of windows and user-activities (see Figure 6).

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 menues, 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 7

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 8, I have shown screenshots of the first two windows implemented straight after the diagram.

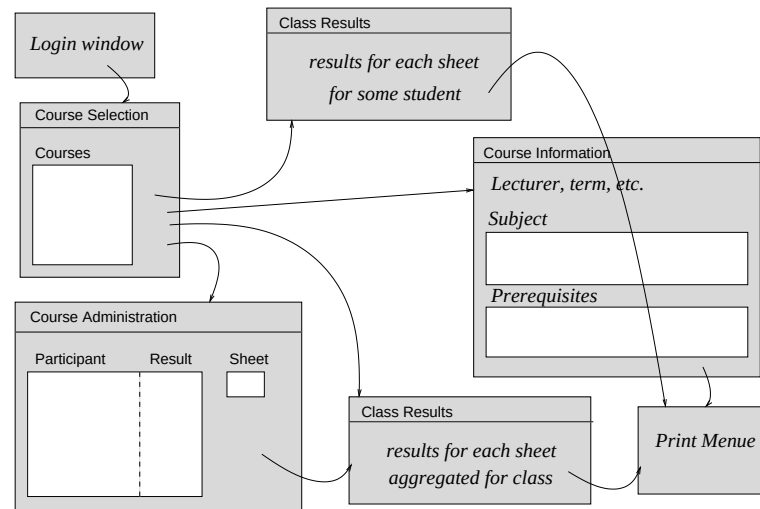


Fig. 6. A first draft of the GUI, containing only the main windows and arcs.

4 Discussion

4.1 Summary

In this paper, a syntactic variant of UML state-transition-diagrams has been proposed under the name Window/Event-diagram. A brainstorming-based technique has been described that allows to design GUIs using WEDs interactively with large groups of non-experts. By getting these people involved in the development, this technique increases acceptance of the system.

4.2 Advantages

In my experience, WEDs and the technique presented above have the following advantageous features:

- The WED notation presents all menus and their relationships simultaneously, that is, the static as well as the dynamic aspect of the GUI (rather than just the static aspect, as paper prototypes).
- The GUI-design is presented in a simple and visual way which is easily understood and may even be applied by non-experts. The notation and the technique need no computer support.
- The WED-notation deemphasizes technicalities like layout or interaction details, and helps the users focus on the logic and functionality instead.
- It can be refined consecutively, that is, it needs not be complete, entirely precise, or operational right from the start.
- 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.

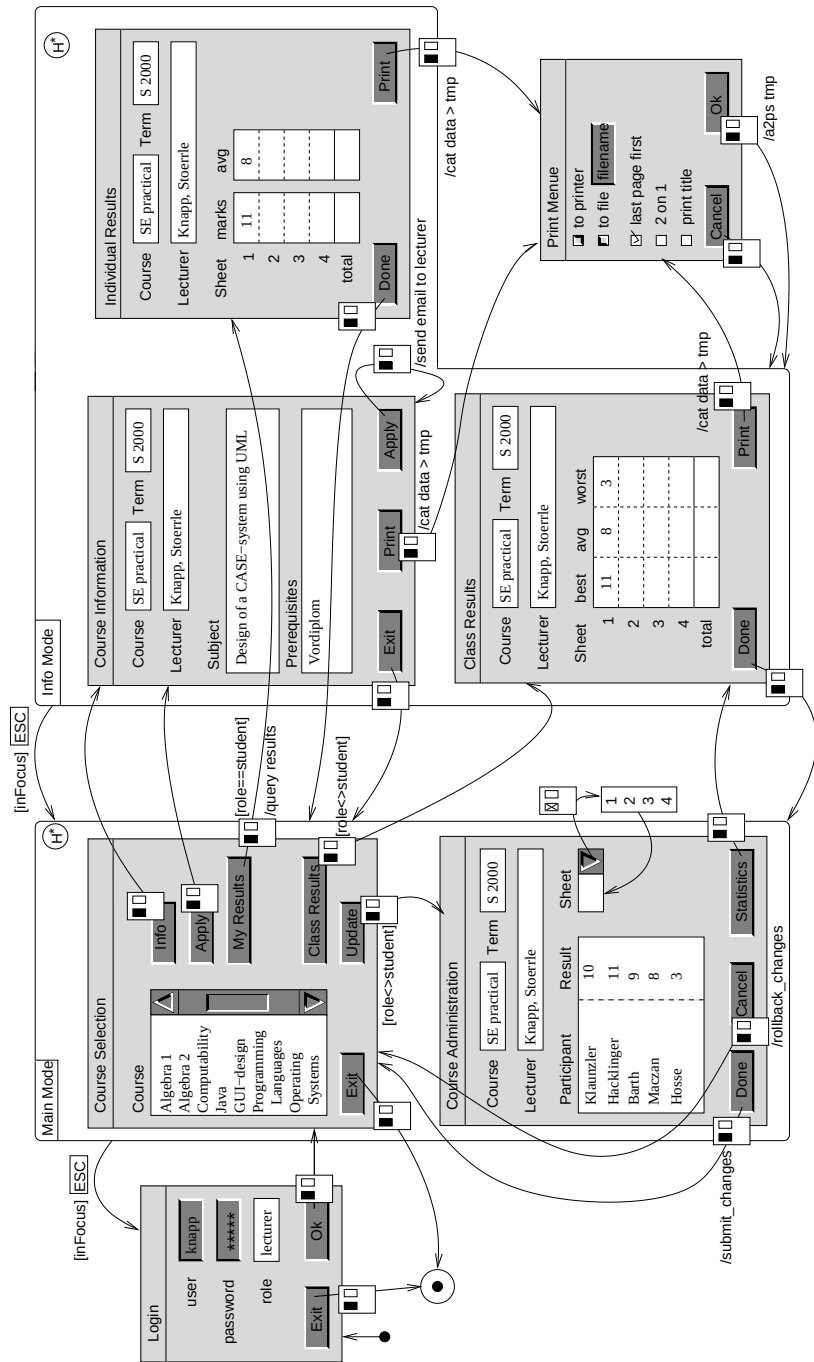


Fig. 7. A complex WED as the design for the GUI of the course information system.

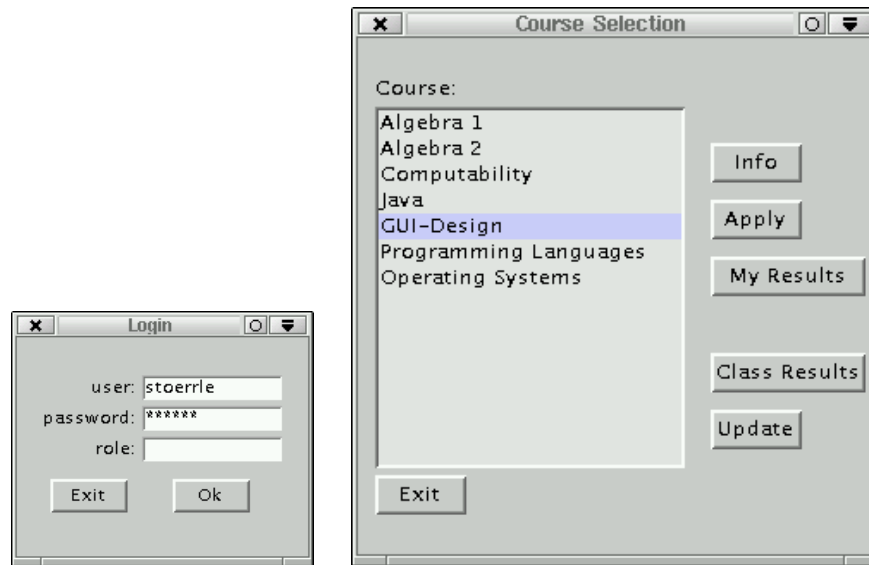


Fig. 8. Screenshots of the first two windows coded using Swing.

- There are no semantic changes to the UML necessary, just a conservative notational extension. So, a WED is easily understood by anyone who speaks UML. Grace to the precise semantics of UML, a WED can also be used as specification for coding (i.e. a kind of a contract), and to some degree probably even for automatic code generation.

4.3 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.

5 Further Work

This approach is also subject to ongoing work. First, 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. Then, one would of course want such a tool to be integrated with a GUI-builder to generate as much as possible automatically. Finally, though this approach has been used successfully with small examples, it is not clear whether it scales up for industrial-strength applications, too.

Acknowledgements Thanks go to Florian Hacklinger and Alexander Knapp.

References

1. Rolf Hennicker and Nora Koch. A UML-Based Methodology for Hypermedia Design. In Selic et al. [6], pages 410–424.
2. Nuno Jardim Nunes and João Falcão e Cunha. Towards a UML Profile for Interaction Design: The Wisdom Approach. In Selic et al. [6], pages 101–116.
3. OMG Unified Modeling Language Specification (draft, version 1.4). Technical report, Object Management Group, February 2001. Available at <http://cgi.omg.org/cgi-bin/doc?ad/01-02-14>.
4. A. Osborn. *Applied imagination*. Scribner, New York, 1957.
5. Paulo Pinheiro da Silva and Norman W. Paton. UMLi: The Unified Modeling Language for Interactive Applications. In Selic et al. [6], pages 117–132.
6. Bran Selic, Stuart Kent, and Andy Evans, editors. *Proc. 3rd Intl. Conf. <<UML>> 2000—Advancing the Standard*, number 1939 in LNCS. Springer Verlag, October 2000.
7. Harald Störrle. *Models of Software Architecture. Design and Analysis with UML and Petri-nets*. PhD thesis, LMU München, Institut für Informatik, December 2000. In print, ISBN 3-8311-1330-0.
8. Patricia Vilain, Daniel Schwabe, and Clarisse Sieckenius de Souza. A Diagrammatic Tool for Representing User Interaction in UML. In Selic et al. [6], pages 133–147.