From Pen-and-Paper Sketches to Prototypes: The Advanced Interaction Design Environment

Störrle, Harald

Published in:
Demonstrations at MODELS 2014

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Störrle, H. (2014). From Pen-and-Paper Sketches to Prototypes: The Advanced Interaction Design Environment. In T. Yue, & B. Combemale (Eds.), Demonstrations at MODELS 2014: Proceedings of the Demonstrations Track of the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (DEMO-MODELS 2014), October 1st and 2nd, 2014 (CEUR Workshop Proceedings, Vol. 1255).

From Pen-and-Paper Sketches to Prototypes: The Advanced Interaction Design Environment

Harald Störrle

Dept. of Applied Mathematics and Computer Science
Technical University of Denmark (DTU), hsto@dtu.dk

Abstract. Pen and paper is still the best tool for sketching GUIs. However, sketches cannot be executed, at best we have facilitated or animated scenarios. The Advanced User Interaction Environment facilitates turning hand-drawn sketches into executable prototypes.

1 Introduction

Graphical user interfaces (GUIs) have two important, independent aspects: appearance and affordances (i.e., visuals vs. behavior). Existing techniques focus mostly on visual appearances, providing tools for a limited scope of visual fidelity (e.g., from GUI-builders at the high end via drawing tools such as Photoshop, Visio or PowerPoint, to sketching tools like Balsamic at the low end). The simplest possible tool for creating sketches of the visual appearance of a UI is, of course, pen and paper (PaP). It turns out, that PaP is hard to beat in terms of usability and cost/benefit ratio; thus it is our gold standard of drawing.

On the other hand, there is the behavioral aspect of GUIs. Most tools completely abstract from this aspect, restricting designers to simple mock-ups of individual scenarios made from hyperlinked pictures, or manually facilitated paper prototypes. A notable exception is Flowella (see `http://www.youtube.com/watch?v=xmuJwKYjiW0`). An early approach of combining rough sketches with interactive executability (to a degree) is the Silk/Denim line of work by Landay et al. [3,2], where users would input a sketch with a digital pen. Both Flowella and Silk/Denim are limited to very small UIs as the complexity of designs grows dramatically with the number of behavioral variants and details added.

AIDE aspires to overcome this limitation by a number of mechanisms, most notably using hierarchical state machines with concurrent substates, and syntactic layers.[1] Also, AIDE allows inputting UI designs by PaP, thus allowing to include diverse audiences in the creation process and reducing overhead and extraneous load caused by inadequate tooling. Such rough sketches (i.e., PaP input) may then be complemented by more elaborate input in the form of XUL (XML User Interface Language, see `https://developer.mozilla.org/en/XUL`). PaP and XUL may be mixed freely, allowing scalable fidelity, offering the following advantages.

[1] See `https://www.youtube.com/watch?v=vbMblOWTRko&feature=youtube.be` or the AIDE homepage `www.compute.dtu.dk/~hsto/tools.html` for a demo.

- **Inclusive Technology** We all learn to use pen and paper from early childhood, so it is safe to assume everyone has sufficient dexterity in sketching; artistic skills are not required. In contrast, operating a computer-based tool often demands substantial expertise and/or training, which end users and domain experts may lack. Thus, using simple pen-and-paper sketches as AIDE does allows us to include virtually everybody in the UI design process.
- **Continuous Workflow** Graphic designers appreciate sketching tools: their low viscosity makes them ideal for exploring the design space (cf. [1,7,8]). However, exploration has to turn into engineering eventually, at which point developers take over from designers and visionary sketches give way to formal models and code. Often, the overall development process is greatly affected by this discontinuity of people, cultures, and methods. AIDE supports a continuous workflow integrating initial sketching with subsequent elaboration.
- **Comprehensive Design** While it is relatively easy to specify the *appearance* of an interface by a drawing, specifying its *behavior* is much more difficult (cf. [6]). In fact, the only way to completely describe arbitrary complex interface behaviors is by programming them. Obviously, integrating drawings and code in a harmonious way is difficult, further disrupting the interface design workflow. Most people are *either* good at graphic design *or* at programming, but rarely at both. Storyboards only offer a partial solution, since they allow to specify a very limited degree of behavior only (basically only linear sequences, see [5, p. 105]). So, in order to create comprehensive interface designs, we need a way to integrate both aspects of an interface, appearance and behavior.
- **Scalable Abstraction** Even small UIs may offer a large number of affordances, all of which act together to create the overall user experience. Capturing them in a prototype is expensive, time consuming, and inhibiting change. Capturing them in a more abstract specification will lead to a bloated and/or fragmented design that is difficult to reintegrate, maintain, and communicate. Establishing and maintaining consistency is an arduous and complex task [4]. So, we need a way of managing the complexity of large interface designs in such a way, that neither the clarity of the initial vision, nor the details of the interactions get lost.

2 Example

Consider an example Fig. 1. It shows a small portion of an interface design from a teaching example, the Library Management System (LMS). This toy case study is about searching for media in a library, lending them, prolonging, them, and and so on. Fig. 1 shows an interface design for the process of issuing a new reader card. The numbers in red dots are not part of the notation but have been added to improve the presentation here.

First, a dialog for entering some data appears. It contains a text field, two groups of radio buttons with two choices each, and two buttons to proceed and abort. The user inputs a reader's name, selects a few options, and eventually

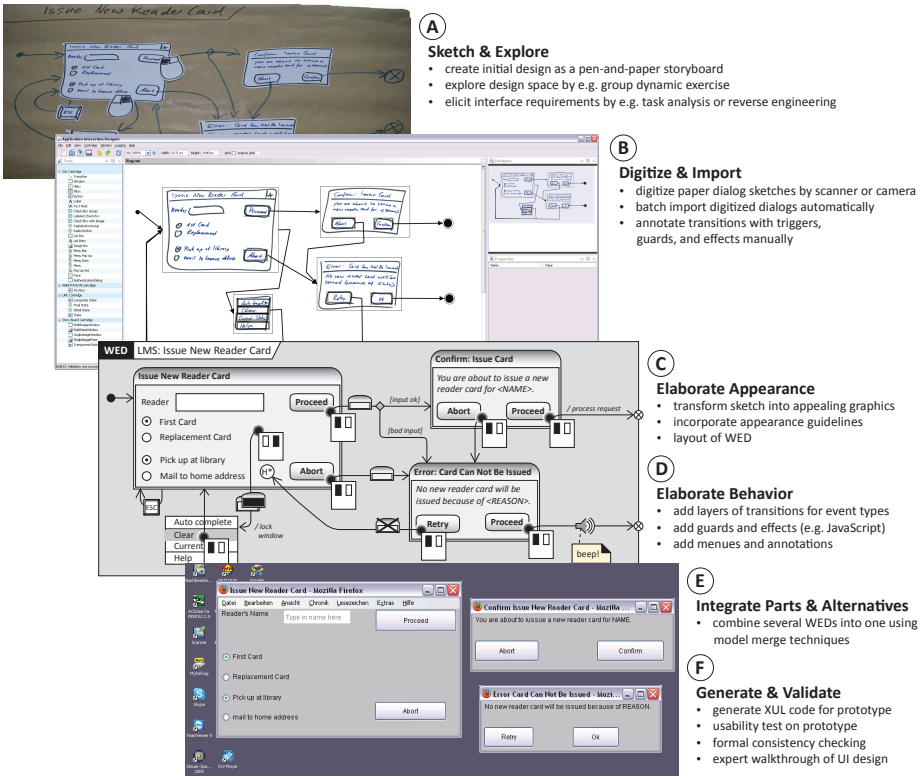

Fig. 1. Stages in the UI development life cycle using WEDs and AIDE: Starting from a traditional pen-and-paper wallpaper prototype, subsequent steps of elaboration yield a prototype with executable XUL code. Note that XUL-elements, clickable areas, and transitions have to be annotated manually in the current version of AIDE.

pressed "Proceed". If the data validation is successful, the user must acknowledges or aborts. If the process is aborted or the validation was not successful, the user may either revise the inputs or terminate the whole process. Using the right mouse button on the window "Issue New Reader Card" will open a pop-up menu with four options.

States UI widgets like text boxes or buttons are represented as simple states. Groups of widgets and complete dialogs are modeled as composite states. As a default, only one widget or dialog can be active at any time, which in UML maps to sequential composite states. In order to achieve different behavior, concurrent composite may be used. Not all states need be visible in a design. For instance, grouping radio buttons together can be achieved by an invisible compound state. The same applies for layout elements such as vertical boxes.

Triggers Positioning a pointer over a WED element is interpreted as putting the focus on that element. Technically, the corresponding state configuration tree

is activated. Any user events issued subsequently will be interpreted by that tree, bottom up. For instance, positioning the mouse pointer over the "Abort" button and pushing the left mouse button (a) issues the event "single left click" in the state "IssueNewReaderCard.Abort" and triggers the transition emanating that state. Likewise, moving the pointer over "Issue New Reader Card" and pressing the Escape key (b) will reset the corresponding state. Any (user) actions the UI affords may be used as triggers.

Guards Transitions may carry a guard that enforces the respective condition to be true before the transition is taken. Guards may refer to environment variables that may be used to represent a hidden UI state such as a mode.

Effects Then, the effect of the transition (modeling a UI action) is executed and its target state is entered. Effects may be described by plain text (a), code snippets, invocation of library functions, or maybe visualized by an icon (b). Probably the most common effects are opening a new window (a), closing one (b), or opening a modal dialog (c).

Entering States When entering a composite state C for the first time, the substate to be entered is determined by the initial state. Reentering C will reset its state configuration unless a history state is added to C, which restores the state configuration in effect at the time of exiting C. Exit Points (and and Entry Points) help achieving a modular design (this is regular UML 2.2 syntax). Exiting a state (or state machine) automatically exits all sub states, i.e., corresponding windows are closed by reaching a final node.

Secondary Notation Annotations and comment boxes may be used freely; they are represented as UML PseudoStates.

There is no semantic difference between UML 2 state machines and a UI design in this form: every construct in a UI design may be mapped to a UML state machine construct. These mappings are typically very straightforward, but have to be added manually in the current version of AIDE.

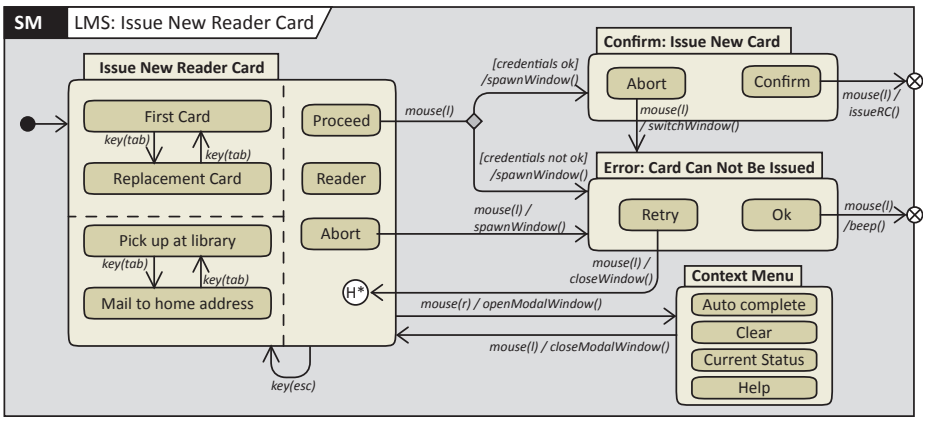

Fig. 2. This UML state machine is yielded by stripping all appearance cues and elaborating effects to procedure calls.

3 The AIDE Tool

The Advanced Interaction Design Environment (AIDE) is a highly modular, platform-independent, direct-interaction tool for creating WEDs, refining and elaborating them in a methodical fashion, supporting distributed concurrent group work, and generating working prototypes from WEDs. AIDE has been under development since 2006, with 25 students at Innsbruck University, Munich University, and the Technical University of Denmark contributing a total of approximately 10,000 work hours to it. Step Ⓑ in Fig. 1 is actually a screen-shot of the AIDE tool, step Ⓔ shows the resulting XUL code executed in Firefox.

AIDE is created using pure Java, using Piccolo2D, jEdit, JGoodies Looks, the Tango Iconset, VLDocking, and JAXB for persistency. AIDE follows a strict separation of logic and presentation. Extensibility of AIDE is ensured by a cartridge mechanism that encapsulates the visual appearance of elements and functions associated with them. Cartridges may be dynamically loaded or unloaded. Apart from the basic cartridges of UML state machines and annotations, there are currently cartridges for the XML User Interface Language (XUL), and for importing hand-drawn storyboards. XUL is used e.g., for defining the UIs of Mozilla projects such as Firefox and Thunderbird. AIDE provides XUL export and integrated simulation. Finally, AIDE also offers unbounded Undo/Redo, user definable roll-back points, tear-off-menus to support very large screens, a locator map, sophisticated zoom and scrolling functions, and multiple views on elements.

References

1. Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann, 2007.
2. James A. Landay and Brad A. Myers. Interactive Sketching for the Early Stages of User Interface Design. Technical Report CMU-HCII-94-104, Carnegie Mellon University, July 1994.
3. James A. Landay and Brad A. Myers. Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer*, 34(3), March 2001.
4. Francisco J. Lucas, Fernando Molina, and Ambrosio Toval. A systematic review of UML model consistency management. *Inf. Softw. Technol.*, 51(12):1631–1645, December 2009.
5. Scott McCloud. *Understanding Comics: The Invisible Art*. HarperPerennial, 1993.
6. Brad A. Myers, Sun Young Park, Yoko Nakano, Greg Mueller, and Andrew J. Ko. How Designers Design and Program Interactive Behaviors. In Paolo Bottoni, Mary Beth Rosson, and Mark Minas, editors, *Proc. IEEE Symp. Visual Languages and humand Centric Computing (VL/HCC'08)*, pages 177–184. IEEE Press, 2008.
7. Mark W. Newman and James A. Landay. Sitemaps, Storyboards, and Specifications: a Sketch of Web Site Design Practice. In *Proc. 3rd Conf. Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS'00)*, pages 263–274. ACM, 2000.
8. Y.Y. Wong. Rough and Ready Prototypes: Lessons from Graphic Design. In *SIG Computer Human Interaction (SIGCHI'92)*, page 685, 1992.