A Visual Language for Sketching Large and Complex Interactive Designs

James Lin [1], Michael Thomsen [2], James A. Landay [1]

[1] Group for User Interface Research
Computer Science Division
University of California
Berkeley, CA 94720-1776, USA
{jimlin, landay}@cs.berkeley.edu

[2] Department of Computer Science
University of Aarhus
Aabogade 34
8200 Aarhus N, Denmark
miksen@daimi.au.dk

ABSTRACT

Informal, sketch-based design tools closely match the work practices of user interface designers. Current tools, however, are limited in the size and complexity of interaction that can be specified. We have created an advanced sketch-based visual language that allows for easy prototyping of large, complex interactive designs. In its current embodiment in the DENIM web design tool, the visual language allows designers to sketch reusable components for recurring page elements, such as navigation bars, as well as conditionals to illustrate and test transitions that depend on a user's input. Designers can also specify sites that accept richer user input than simple clicking. Our informal evaluation shows that these features allow designers with little programming experience to quickly create prototypes of large, complex web sites while still working inside an informal, sketch-based environment.

KEYWORDS

Visual language, DENIM, user interface design, web design

INTRODUCTION

Designers of web sites typically go through a process of progressive refinement [12]. They tend to think about the larger picture, such as the overall site architecture, early on, and then progressively focus on finer details, such as the specific look of page elements, typefaces, and colors.

The design process often includes rapid exploration early on, with designers creating many low-fidelity sketches on paper. There are several benefits of sketching during this phase of design. Sketches are inherently ambiguous, which allows the designer to focus on basic structural issues instead of unimportant details. The ambiguity also allows multiple interpretations of the sketch, which can lead to more design ideas [3]. Sketching is quick, so designers can rapidly explore different ideas and iterate on those ideas. In addition, user tests using rough prototypes tend to find the same usability problems as do tests with more finished prototypes [7, 17].

A few computer-based web site and user interface design tools offer the benefits of sketching by using it as their primary interaction technique. These tools include DENIM [10] (see Figure 1) and SILK [9]. DENIM and SILK also allow the designer to specify interface behavior through *storyboarding*, where the designer draws arrows from one page to another to denote page transitions.

However, using these tools to create prototypes of larger and more complex interfaces is cumbersome. If a designer wants every page to have a navigation bar at the top, for example, the designer must redraw the navigation bar and link its contents on every page. Needing to redefine common elements like this leads to an explosion of pages and arrows, which becomes hard to manage. Also, a page transition in DENIM and SILK cannot depend on the state of other interface elements in the page, such as whether a check box is checked, making it difficult to create prototypes with this common type of behavior.

After several design iterations, we have created a visual language, employing a sketching metaphor, which addresses these problems. The target audience of the language is designers, who are not likely to know program-

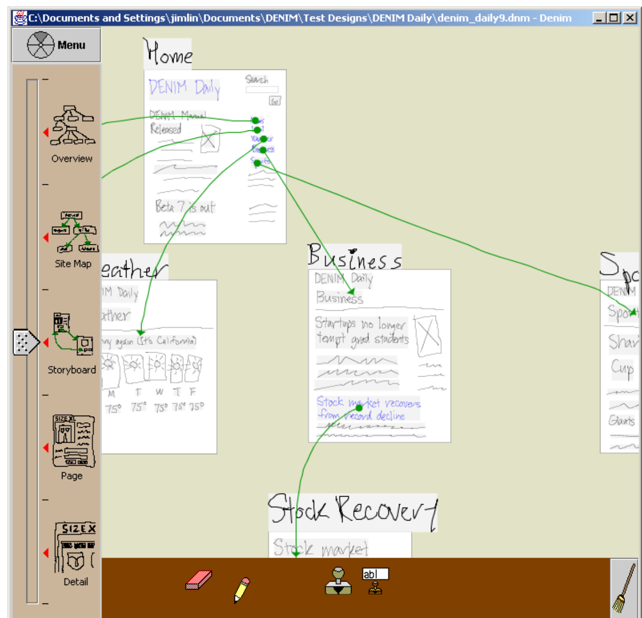


Figure 1. DENIM displaying a sketch of five web pages and six transition arrows.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CHI 2002, April 20-25, 2002, Minneapolis, Minnesota, USA.
Copyright 2001 ACM 1-58113-453-3/02/0004…$5.00.

ming, but who nevertheless wish to prototype large, interactive designs during the early stages of design. The language uses pages and arrows to represent simple interfaces, like in DENIM and SILK, but it also includes *components*, which allow designers to reuse storyboard fragments, *enhanced arrows*, which allow for more event types, and *conditionals*, which allow page transitions to depend on the state of interface elements in a page. We have implemented it within a new version of DENIM[1]. The visual language can be applied to areas other than web design, such as desktop graphical user interface design.

The rest of the paper is organized as follows. First, we discuss work related to the visual language. Next, we give an overview of DENIM and describe the core functionality that lays the foundation for the visual language. Following this is a discussion of the visual language itself in detail. A brief discussion of the implementation follows. Next we describe our evaluation of the language, and finally we describe our future work and conclusions.

RELATED WORK

There is work related to the visual language in the areas of storyboarding, components, and conditionals.

Storyboarding

Storyboards illustrate interactive behaviors by showing what the user interface looks like before and after an end-user event occurs. Besides SILK and DENIM, other design systems that use storyboarding include Anecdote [4] and PatchWork [16]. These systems only support left click events for a designer's custom widgets, whereas our new visual language supports several input events.

DEMAIS [1] is a system for prototyping multimedia applications. It allows designers to rapidly sketch out a storyboard of their design, add layers to their designs for easier organization, and draw arrows between storyboard elements to denote interaction and timing. It does not have a mechanism for packaging objects and behavior together to be reused in other parts of the storyboard.

Chimera [8] and Pursuit [11] are programming-by-demonstration systems based on a before-and-after comic strip metaphor. These systems infer what action causes a state transition from examples, while in our system the designer states explicitly what user action causes a transition. We call this type of explicit specification *programming by illustration*.

Components and Conditionals

As mentioned before, our visual language includes components to support recurring page elements. Our design is reminiscent of statecharts [5], an extension of finite state machines. However, statecharts are more general, supporting, for example, concurrency and message broadcast. Since the user can only interact with one interface "state" at a time, our components have no such concepts, and hence our design is somewhat simpler and easier to use. We have also created an innovative interface for specifying and using components.

The visual language also includes conditionals, which allow a transition between web pages to depend on the state of other elements in the page. Our design was influenced by rule-based visual programming languages such as AgentSheets [14], Stagecast Creator [15], and KidSim [2]. The foundation of such languages is a collection of if-then rules. However, the design paradigms of our new language and rule-based languages differ. A path that a user can take through the interface has a direct visual representation in a DENIM design as a path through a network of web pages. In AgentSheets and Stagecast Creator, there is no direct visual representation of a user's path; the closest representation would be a history of which rules had been executed. Visual representations of conditionals are easier for novices to use [13].

DENIM

As mentioned above, we have implemented the new visual language within DENIM, an electronic tool for the early stages of web site design. DENIM supports sketching input, allows design using three different representations, and unifies the representations through zooming.

DENIM is part of our research on *informal user interfaces*. Informal user interfaces support natural human input, such as speech and writing, while minimizing recognition and transformation of the input. These interfaces, which *document* rather than *transform*, better support a user's flow state. Unrecognized input embraces nuanced expression and suggests a malleability of form that is critical for activities such as early-stage design[2].

Since DENIM was designed with a pen interface in mind, we use pen-based terms for describing the interaction between the designer and DENIM. For example, *tapping* means to tap the pen onto the digitizing tablet. This corresponds to clicking the primary button on a mouse.

DENIM has one window (see Figure 1) with three main areas. The center area is a canvas where the user creates web pages, sketches the contents of those pages, and draws arrows between pages to represent their relationship to one another. On the left is a slider that is used to set the current zoom level. The bottom area is a toolbox that holds tools for drawing, panning, erasing, and creating and inserting reusable components.

Instead of pull-down menus, DENIM uses techniques geared towards pen interaction. Pie menus are used for executing commands. Alternatively, pen gestures can be

[1] An earlier version of DENIM and videos demonstrating the new visual language can be downloaded from:

http://guir.berkeley.edu/denim

[2] Given DENIM's focus on the early stages of design, the designer's raw sketches are left rough. Once the site design is finished, the final code for the site is produced using traditional tools.

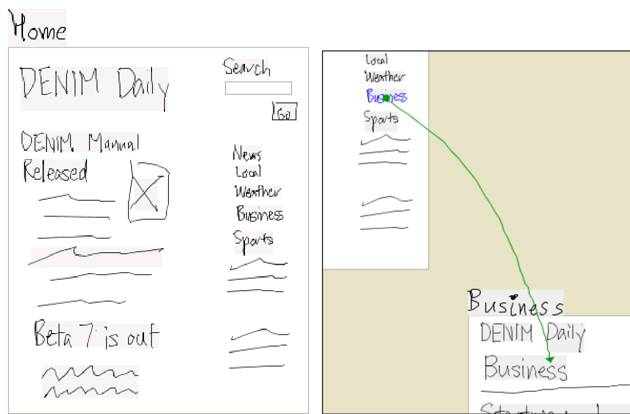


Figure 2. a) A DENIM page with the label "Home" b) An arrow, whose source is a blue hyperlink, "Business."

Figure 3. The Checkout page for an e-commerce site.

used for quickly executing the most common commands, such as copying, pasting, and panning.

Designers test out the interaction of their designs in DENIM'S Run mode. Opening a pie menu over a page and selecting File→Run opens a separate DENIM browser window with the page loaded. The designer can navigate through the site design exactly like in a web browser, clicking on links and using the Back and Forward buttons.

THE FOUNDATION OF THE VISUAL LANGUAGE

Our visual language has its foundation in two of DENIM's core concepts: *pages* and *arrows*.

Pages represent the web pages in a site. A page consists of two parts: a label describing the page, and a sketch representing the physical appearance of the web page (see Figure 2a). A designer can sketch or type in a page.

An arrow between two pages represents a relationship between those pages (see Figure 2b). To create one, the designer draws a stroke between two pages. If an arrow starts from a particular item in a page, such as a word, image, or button, then the source of the arrow becomes blue, like a hyperlink in a web page. Furthermore, in Run mode, the user can click on the item to transition to the destination page—the source of the arrow is a hyperlink.

THE VISUAL LANGUAGE'S ADVANCED FEATURES

Using pages and arrows, designers can fully describe a simple web site consisting solely of web pages and hyperlinks. However, these basic constructs are not sufficient for more advanced sites.

An Example Scenario of an Advanced Design

Consider a designer of a shopping web site who is prototyping the checkout procedure. The designer wants to prototype the following behavior:

- The checkout page presents a shopper with the current contents of her shopping cart, two check boxes for optional Gift Wrapping and Gift Card, and a Next button (see Figure 3).
- If she checks neither Gift Wrapping nor Gift Card, then clicking Next takes her to the shipping page.
- If she checks only Gift Wrapping, then she will be taken to a page for selecting the theme of the wrapping (e.g., birthday, graduation, or sympathy).
- If she checks only Gift Card, then a page for selecting the theme and text of the card will be next.
- Finally, if she selects both Gift Wrapping and Gift Card, then she will be taken to a page that only shows those themes for which there are appropriate matching wrapping and cards.
- Clicking Next in any of the gift card or gift wrapping pages leads to the shipping page.
- Additionally, if the user does not make any selection in the checkout page within five minutes, then the session will timeout for security purposes, and the browser will transition automatically to a login page.

If the designer tries to create an interactive prototype of this design using only pages and arrows, he or she would encounter several problems.

- To simulate the behavior of the check boxes, the designer would have to draw four pages, one for each possible combination of check box states. The designer would also have to draw arrows from a checked box to an unchecked box for each of the

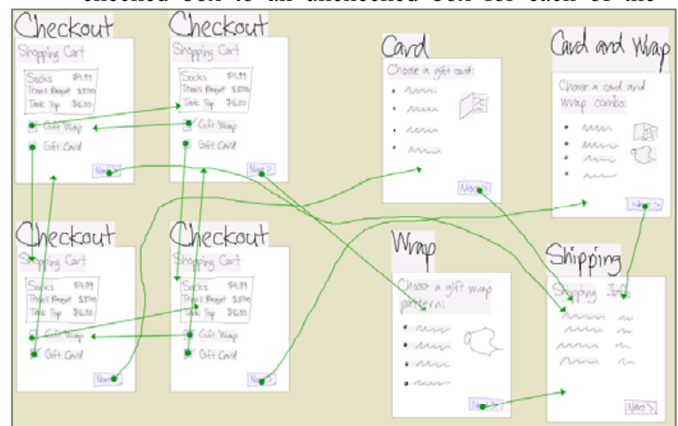


Figure 4. Combinatorial explosion: transitions depending on two states leads to four pages and eight arrows.

check boxes. As illustrated in Figure 4, this leads to combinatorial explosion, where two check boxes lead to eight arrows and so on, a "visual spaghetti" that is clearly not desirable.

- If the designer wanted to use check boxes elsewhere, he would have to redraw this complex sequence again, which is time consuming and error prone.
- Since there is no way to specify page transitions on anything other than a left-click event, prototyping the automatic timeout transition is impossible.

We have overcome these problems while still maintaining the sketching paradigm. By sketching out components, the designer can create recurring elements like a check box. By using conditionals and components, the designer can avoid the combinatorial explosion of explicitly defining all combinations. By using enhanced arrows, the designer can show page transitions that trigger on events other than simple left-clicks, such as timeouts. These powerful language features are discussed in detail below.

Components

Components provide a mechanism for the designer to create and use reusable widgets and fragments of interface designs. There are two types of components in DENIM: *intrinsic* and *custom*. Intrinsic components are standard widgets or page elements built into the visual language. Currently, we have implemented text fields, but we plan to add more intrinsic components, such as buttons, radio buttons, and scroll bars. Custom components differ from intrinsic components in that they are defined by designers, allowing them to create their own "building blocks."

Inserting a Component Instance

Every component has a "rubber stamp" tool associated with it (see Figure 5). Each rubber stamp has the name of the component or an icon that represents what the component looks like. The one plain rubber stamp is used for *creating* custom components.

Figure 5. Rubber stamps for a text field intrinsic component, check box custom component, and creating custom components, respectively.

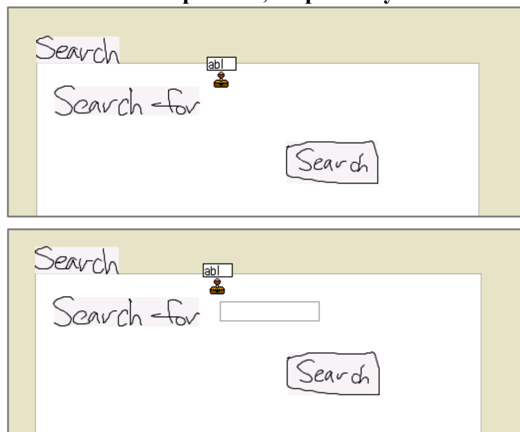


Figure 6. a) Inserting an instance of the text field component into a page. b) The result.

To insert an instance of a component into the design, the designer picks up the component's stamp and then taps on the desired location in the design (see Figure 6).

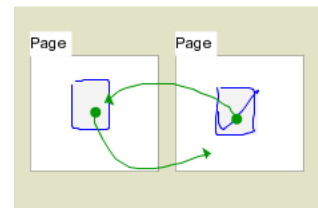
Once the component instance has been inserted, the designer can make adjustments to it, such as changing its position by dragging it, or changing its initial state by opening a menu on the instance.

Custom Components

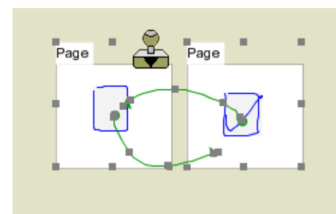
As we stated above, designers can create their own components for reusable page elements. For simplicity, custom components are completely self-contained—instances of custom components cannot know about or refer to other component instances used within the same page. Although this limits the expressive power of custom components, we do not believe this is a serious limitation in the context of low-fidelity prototyping.

In our example, the designer can create a custom *check box* component for his checkout pages. This involves the following steps:

1. First, the designer draws two small pages: one showing the checkbox unchecked, and one showing it checked. He then draws an arrow representing a left click transition from the page with the unchecked box to the page with the checked box and vice versa.



2. Next, he selects the pages by drawing a circle around them while holding down the pen's barrel button.
3. He then picks up the blank rubber stamp and taps the selected pages to create the new component.



4. The pages now disappear, and the designer is asked

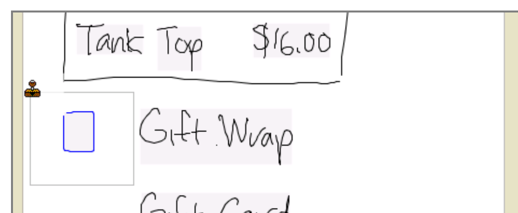


Figure 7. Inserting an instance of the Check box component.

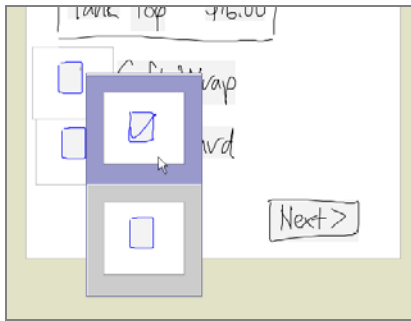


Figure 8. Changing the state of a check box component instance.

for the name of the component. A new rubber stamp with that name is created and placed in the toolbox.

The designer can now insert instances of this component into his design. To do so, he picks up the check box stamp and taps it in the canvas to stamp instances of the component (see Figure 7). The designer can also use a menu to change the state of a check box (see Figure 8).

When the designer interacts with his design in Run mode, the check boxes will check and uncheck when clicked, as defined by his check box component.

Editing a Component

The designer can edit a custom component by opening a separate pane containing the component's definition (see Figure 9).

For example, suppose the designer decides to use Macintosh-style check boxes that have an X for the checked state instead of a check mark. To do this, the designer brings up the component pane, erases the check mark, draws an X, and then closes the pane. This illustrates another advantage of using custom components: this changes the appearance of all instances of this component throughout the design automatically.

Creating Global Transitions

There are also cases in which a designer wants all instances of a custom component to always transition to a specific page in the design after a certain event. In the case of our example design, consider adding a navigation bar to each checkout page, where clicking on "Cart" in the navigation bar would bring the user back to the first checkout page, and clicking "Home" would take the user to the home page. Drawing the navigation bar on each page would lead to an unmanageable number of arrows.

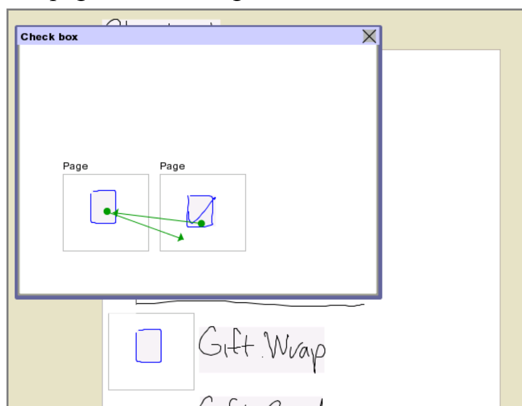


Figure 9. The component pane allows editing the definition of a custom component.

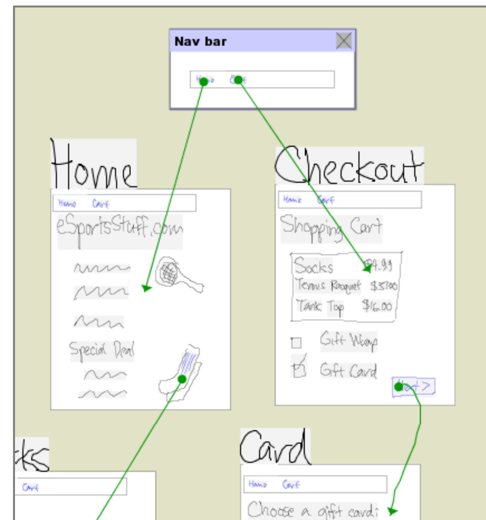


Figure 10. A navigation bar component with outgoing arrows (top), and four pages with instances of the component.

Instead, the designer uses components with *global transitions*. First, he draws a navigation bar and makes a component. He then opens the definition pane for the navigation bar component, and draws outgoing arrows from the component definition to specific pages outside of the component pane. For example, Figure 10 shows the navigation bar component with global transitions to the home page and the first checkout page.

Notice that no arrows are needed from any individual instance of the navigation bar component. Because of the component's definition, clicking on "Cart" in Run mode will take the end-user from any page with the navigation bar back to the first page of the checkout process.

Handling Conditional Transitions

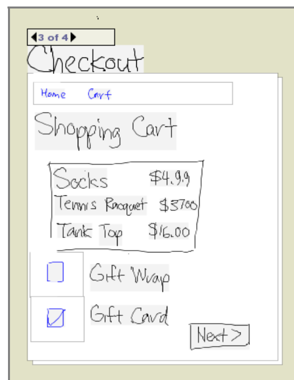
While components solve some of the designer's problems in the checkout scenario, we still have not completely solved the problem of "spaghetti-style" links, which plagued the initial checkout page design (see Figure 4). The first Checkout page (see Figure 3) contains the cart contents, two check boxes, and a "Next" button. The designer would like to link each of the four check box combinations to the appropriate pages in a clearer way.

The designer also needs a way to specify transitions that depend on the state of component instances, such as the two check boxes. *Conditionals* address these issues.

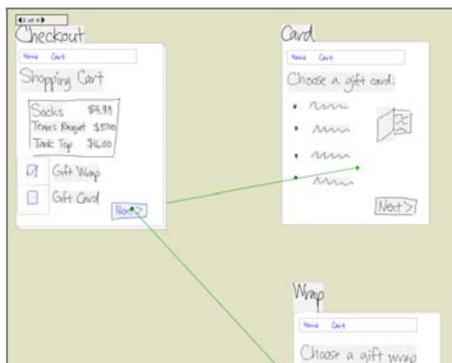
1. To make the checkout page a conditional, the designer opens the pie menu on the page, and selects Page→Add New Condition.

2. The page becomes a *stack* of conditions, initially holding two conditions. A bar appears at the top of the page showing which condition the designer is editing. The designer needs four conditions (two states raised to the power of two component instances). He uses the pie menu two more times, resulting in a conditional stack with four conditions.

3. The designer now needs to specify the state of the check boxes in each of the conditions. To switch the condition that the designer is editing, the designer simply taps on the left or right arrow in the bar above the stack. In this example, he goes to the second condition, and toggles the state of the first check box, then goes to the third condition and toggles the state of the second check box, and so on.



4. With each of the four conditions specified, the designer is ready to add the transitions. This task is simplified by the fact that there is no need to add the transitions that make each of the check boxes toggle—since the check boxes are instances of a custom component, this has already been specified. Only the arrows from the "Next" button in each condition have to be added. The designer goes through each of the four conditions, drawing one outgoing transition from the "Next" button to the appropriate page.



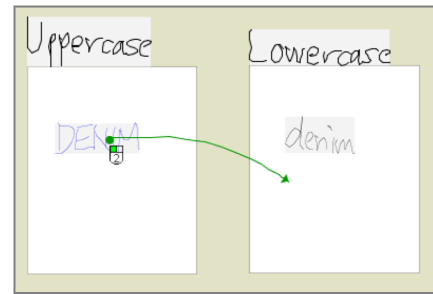


Figure 11. Arrow representing a double left-click.

Conditionals relieve the designer of a lot of work, and the resulting design is also much simpler to understand without the spaghetti of arrows. As another added benefit, making changes to the checkout page is also easy: a change to the contents of *one* of the conditions in the stack, for example, changing the word "Cart" to "Basket," is automatically reflected in *every* condition in the stack.

In some cases, a conditional transition should not depend on the total state of the origin page. In these cases, the designer can specify with the pie menu which user interface elements do not matter for evaluating a condition.

Enhanced Arrows

The last task for the designer in our scenario is to prototype automatic timeout on the Checkout page.

Originally, transition arrows represented clicking on a link with the left mouse button. Arrows in our new visual language can support other events common to desktop and web user interfaces, such as double-clicking and rollovers. *Enhanced arrows* display the type of event they represent near the source of the arrow (see Figure 11).

Enhanced arrows are drawn like normal arrows (see Figure 12a), except when the designer reaches the destination of the arrow, she does not lift up the pen. After a short delay, a pie menu opens with the events that the source of the arrow can handle (see Figure 12b). The designer then taps the desired event type. Normal arrows still represent a single click of the left mouse button.

Currently, the events that enhanced arrows support include single-clicking and double-clicking with either mouse button, mouse enter and mouse exit for rollover effects, and timers that lead to a new page after a certain amount of time after the first page is loaded.

Using a timer, a designer can easily prototype the timeout page in our checkout scenario: he simply draws an enhanced arrow from the first checkout page to a timeout

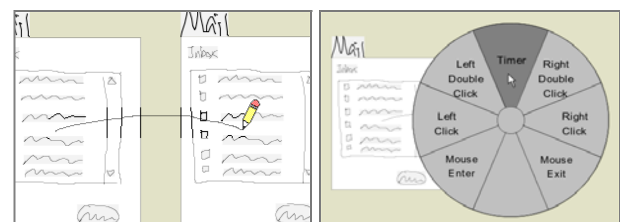


**Figure 12. a) Drawing an enhanced arrow.
b) Selecting the event type of the arrow.**

page, selects Timer in the menu, and then enters a value of 300 seconds in the property box that appears.

ITERATIVE DESIGN AND EVALUATION

After the initial design, but before the full implementation, we performed a brief, informal study on the visual language. The study had six participants. Three of them were professional user interface designers with moderate to advanced programming ability, and the other three were computer science undergraduates at UC Berkeley who were taking a human-computer interaction course. After a short introduction to the visual language, we gave them several interface design tasks and asked them to do these tasks on paper. Overall, the informal study validated our design, and thus motivated us to implement the visual language additions.

Evaluation

To evaluate the final design, we performed an informal task-based, usability test. The participants were introduced to DENIM and the visual language, and then asked to create elements of a simple e-commerce site.

Participants

There were a total of four research participants, two males and two females, from 26 to 31 years in age. All were final year master's students majoring in Information Science or Multimedia Design at the University of Aarhus. They were chosen to have backgrounds similar to professional web designers. According to pre-study questionnaires, they all had extensive knowledge of various drawing tools such as Adobe Photoshop but "little" or no knowledge of programming.

Apparatus and Procedure

The subjects performed the study on a 700 MHz Pentium III PC with Windows 2000, a 21″ CRT color screen (1280×1024 resolution) for output, and a Wacom Intuos A4 regular tablet with both pen and mouse for input.

Each subject started out by filling out an informed consent form and a pre-study questionnaire. They were then allowed to use Microsoft Paint with the tablet for about 5–10 minutes to acquaint themselves to this input device. Next, they were introduced to basic DENIM interaction followed by task 1 (see below). Then they were given a two-page description of the visual language, which the experimenter guided the participant through for five minutes to accelerate the learning process. After this they were asked to complete tasks 2 through 4.

Tasks

The tasks centered on the goal of prototyping a site selling products for household pets. Task 1 involved creating the pages *Index*, *Dog products*, *Shopping basket*, and *Order confirmation* with initial content and links in between. The task was designed to teach the participants basic visual language functionality, and to provide a basis for the following tasks. Task 2 involved adding an *Include gift card* option and was designed to test their understanding of components (in this case, creating a check box component). Task 3 involved adding a *Choose gift card page* and was designed to test if they were able to add a condition to the *Shopping basket* page and then link to the correct pages. Finally, task 4 asked them to add a timeout security feature, and was designed to test their understanding of enhanced arrows.

Results

All four participants completed all four tasks in 20 to 30 minutes[3]. Two completed them without any help, whereas the other two were offered a little help (e.g., "have you looked in the manual" or "components should be created on the background"), primarily when they got confused because of bugs in the implementation. Given that the participants had only five minutes of training and no programming experience, we find these results impressive.

The post-study questionnaires indicated that two users were moderately happy with the system, whereas the two others were very happy and gave comments like: "*It seems much more informal, but you still have all the functionality,*" "*I feel like I [can] focus more on the design,*" "*better than paper and pencil!*" and "*intuitive construction of interactivity*." These comments suggest that the subjects found the tool useful and especially liked the informal style of interaction.

Problems encountered during the evaluation were mostly due to programming bugs, but two design problems were also discovered: two participants wanted to be able to create a component "in place" on a page, and another two suggested that the number of conditions that a conditional initially has should be determined by the number of possible component states. Finally, three subjects complained that the Wacom tablet was hard to use.

IMPLEMENTATION

We implemented the visual language within a new version of DENIM, which is built with the Java 2 SDK version 1.3, using SATIN [6], a toolkit for building informal, pen-based applications. Most of the implementation details are pretty standard, so here we only discuss the implementation of components, conditionals, and events.

There are two base classes that form the heart of the component subsystem. `DenimComponent` represents the definition of a component. `DenimComponentInstance` represents a usage of a component, which is created when a designer stamps with a component's rubber stamp. All `DenimComponents` have a list of events, such as right-click and left double-click, that their corresponding `DenimComponentInstances` listen to in Run mode.

When an arrow of a particular event type is drawn from a `DenimComponentInstance` within one page to another, DENIM associates the event type and the current condition

[3] During the execution of one test, one participant accidentally selected "Quit" in a menu and exited the system. The experimenter recovered by quickly redrawing part of his design, and then continued the test.

of the page with the destination page in the event table of the `DenimComponentInstance`.

When the user interacts with a design in Run mode, an event within a `DenimComponentInstance` is handled by detecting the page's current condition and looking up the associated destination in the component instance's event table, and replacing the contents of the browser window with the destination page.

FUTURE WORK

We have designed but have not yet implemented a few additional aspects of the visual language. We plan to implement the following features soon.

Text Variables

Currently, there is no way for a value that an end-user inputs into a text field to be used in other pages. We have designed a mechanism in which the designer can give a name to a text field, and then insert the name in other pages. At run time, the name would be replaced by the contents of the text field with that name.

Page Masters

Pages within a web site often use the same general layout, such as a logo in the top left corner. Currently, there is no easy way of specifying this. Designers should be able to create and edit *page masters* in a manner similar to creating and editing components. These could then be used as templates when creating a new page, or they could be applied to existing pages to allow designers to experiment with various layout alternatives.

CONCLUSION

Our visual language includes advanced concepts that are necessary for designing large, interactive web sites and user interfaces. Components reduce the complexity and "visual spaghetti" of large designs by letting the designer define and reuse common interface elements. Conditionals allow the designer to specify transitions that depend on user behavior. Enhanced arrows allow the designer to specify user behavior besides simple clicking. All of this is accomplished using a familiar sketching metaphor, enabling designers to keep the benefits of informal representations along with the advantages of electronic tools.

The visual language allows interface designers to prototype sophisticated interfaces for more advanced and larger sites, facilitating evaluations of these at an earlier stage. Finally, the design of the visual language is a good fit for the original audience of DENIM, web site designers with little or no programming background, and, as we have successfully evaluated, is directly usable after even very little training.

ACKNOWLEDGEMENTS

Thanks to Jason Hong and Orna Tarshish for helping us with DENIM and SATIN, and Qualcomm for their financial assistance of this work. Also thanks both to Anoop Sinha, who coined the term *programming by illustration*, and Francis Li for their valuable comments on this paper.

REFERENCES

1. Bailey, B.P., J.A. Konstan, and J.V. Carlis. DEMAIS: Designing Multimedia Applications with Interactive Storyboards. In Proceedings of *ACM Multimedia 2001*. Ottawa, Canada. pp. 241-250, Sept. 30-Oct. 5, 2001.
2. Cypher, A. and D.C. Smith. KidSim: End User Programming of Simulations. In Proceedings of *Human Factors in Computing Systems: CHI '95*. Denver, CO. pp. 27-34, May 7–11, 1995.
3. Goel, V., *Sketches of Thought*. Cambridge, MA: The MIT Press. 279, 1995.
4. Harada, K., E. Tanaka, R. Ogawa, and Y. Hara. *Anecdote*: A Multimedia Storyboarding System with Seamless Authoring Support. In Proceedings of *ACM International Multimedia Conference 96*. Boston, MA. pp. 341-351, November 18-22, 1996.
5. Harel, D., Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 1987. **8**(3): pp. 231-274.
6. Hong, J.I. and J.A. Landay, SATIN: A Toolkit for Informal Ink-based Applications. *CHI Letters: Proceedings of User Interfaces and Software Technology: UIST 2000*, 2000. **2**(2): pp. 63-72.
7. Hong, J.I., F.C. Li, J. Lin, and J.A. Landay. End-User Perceptions of Formal and Informal Representations of Web Sites. In Proceedings of *Human Factors in Computing Systems: CHI 2001 Extended Abstracts*. Seattle, WA. pp. 385-386, March 31-April 5, 2001.
8. Kurlander, D. and S. Feiner, A History of Editable Graphical Histories, in *Watch What I Do: Programming by Demonstration*, A. Cypher, Editor. MIT Press. pp. 405-413, 1993.
9. Landay, J.A. and B.A. Myers, Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer*, 2001. **34**(3): pp. 56-64.
10. Lin, J., M.W. Newman, J.I. Hong, and J.A. Landay, DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design. *CHI Letters: Proceedings of Human Factors in Computing Systems: CHI 2000*, 2000. **2**(1): pp. 510-517.
11. Modugno, F. and B.A. Myers, Graphical Representation and Feedback in a PBD System, in *Watch What I Do: Programming by Demonstration*, A. Cypher, Editor. MIT Press: Cambridge, MA. pp. 415-422, 1993.
12. Newman, M.W. and J.A. Landay. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In Proceedings of *DIS 2000: Designing Interactive Systems*. New York, New York. pp. 263-274, August, 2000.
13. Pane, J.F. and B.A. Myers. Improving User Performance on Boolean Queries. In Proceedings of *Human Factors in Computing Systems: CHI 2000 Extended Abstracts*. The Hague, the Netherlands. pp. 269-270, April 1-6, 2000.
14. Repenning, A. and W. Citrin. Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction. In Proceedings of *IEEE Symposium on Visual Languages (VL'93)*. Bergen, Norway: IEEE Computer Society Press. pp. 77–82, September, 1993.
15. Stagecast, *Stagecast Creator*, 1997. Stagecast Software, Inc. http://www.stagecast.com/
16. van de Kant, M., S. Wilson, M. Bekker, H. Johnson, and P. Johnson. PatchWork: A Software Tool for Early Design. In Proceedings of *Human Factors in Computing Systems: CHI 98 Summary*. Los Angeles, CA. pp. 221-222, April 18-23, 1998.
17. Virzi, R.A., J.L. Sokolov, and D. Karis. Usability Problem Identification Using Both Low- and High-Fidelity Prototypes. In Proceedings of *Human Factors in Computing Systems: CHI '96*. Vancouver, BC, Canada. pp. 236-243, April 13–18, 1996.