Tangible User Interface Input: Tools and Techniques

by

Scott Robert Klemmer

B.A. (Brown University) 1999

M.S. (University of California, Berkeley) 2001

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor James Anthony Landay, Chair
Professor Jennifer Mankoff
Professor Terry Winograd
Professor Paul K. Wright

Fall 2004

The dissertation of Scott Robert Klemmer is approved:

_____

Chair Date

_____

Date

_____

Date

_____

Date

University of California, Berkeley

Fall 2004

Tangible User Interface Input: Tools and Techniques

by

Scott Robert Klemmer

ABSTRACT

Tangible User Interface Input: Tools and Techniques

by

Scott Robert Klemmer

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor James Anthony Landay, Chair

Tangible user interfaces (TUIs) augment the physical world by integrating digital information with everyday physical objects. Developing tangible interfaces is problematic because programmers are responsible for acquiring and abstracting physical input. This is difficult, time-consuming, and requires a high level of technical expertise in fields very different from user interface development—especially in the case of computer vision. Consequently, only a small cadre of technology experts can currently build these UIs. Based on a literature review and structured interviews with TUI researchers, we created Papier-Mâché, a toolkit for building tangible interfaces using computer vision, electronic tags, and barcodes. Papier-Mâché introduces high-level abstractions for working with these input technologies that facilitates technology portability. We evaluated this toolkit through a laboratory study and longitudinal use in course and research projects, finding the input abstractions, technology portability, and monitoring facilities to be highly effective. This dissertation contributes new software tools and interaction techniques for tangible user interface input. It comprises *Papier-Mâché*, a toolkit for building tangible UIs; *Books with Voices*, a system providing

barcode-augmented paper transcripts for random access to digital video; and *The Designers' Outpost*, a tangible UI for collaborative web site design.

Papier-Mâché's design has been deeply influenced by my experiences building physical interfaces. Web designers use pens, paper, walls, and tables for explaining, developing, and communicating ideas during the early phases of design. These practices inspired The Designers' Outpost. With Outpost, users collaboratively author web site information architectures on an electronic whiteboard using physical media (Post-it notes and images), structuring and annotating that information with electronic pens. This interaction is enabled by a touch-sensitive electronic whiteboard augmented with a computer vision system. We conducted several studies of this system that validated that Outpost supports information architecture work practice and led us to develop support for design history, remote collaboration, and fluid interoperability with other design tools.

The second major TUI we developed focused on oral historians. While oral historians consider interview recordings a central historical artifact, these recordings sit unused after a written transcript is produced. We hypothesized that this is largely because books are more usable than recordings. Therefore, we created Books with Voices: barcode-augmented paper transcripts enabling fast, random access to digital video interviews on a PDA. Our evaluation found this lightweight, structured access to original recordings to offer substantial benefits with minimal overhead. Both The Designers' Outpost and Books with Voices could have been built in a fraction of the time if the Papier-Mâché toolkit had been available.

Professor James Anthony Landay
Dissertation Committee Chair

For my parents,

Cheryl and Chris Klemmer,

who have always encouraged me

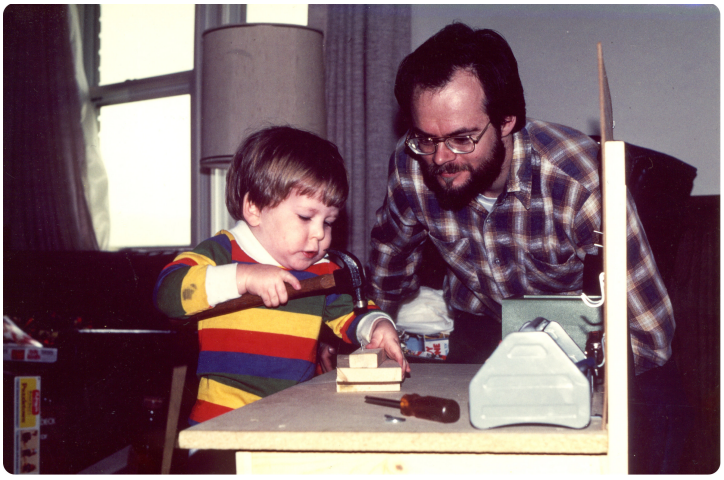


Table of Contents

List of Figures

List of Tables

Acknowledgements

A dissertation is created with the support and inspiration of family, friends, and colleagues: through close collaboration, moral support, distraction, and happenstance discussions at conferences, coffee shops, bars, and BART stations.

I am intensely grateful to all of the individuals that have donated their time as participants in my fieldwork and user evaluations, as well as the early adopters of the Papier-Mâché toolkit. While the need for anonymity prevents me from expressing my appreciation publicly to each individual, I would like to thank in aggregate: the twenty-seven web design professionals who participated in the Designers' Outpost studies, the nine oral historians and four book club members who participated in the Books with Voices evaluation, the nine tangible UI researchers I interviewed for Papier-Mâché, and the seven computer science graduate students who participated in the laboratory evaluation. I would also like to thank Richard Candida Smith and his colleagues at Berkeley's Regional Oral History Office for our discussions on technology support for oral history, and my colleagues who were early adopters of the Papier-Mâché toolkit in their research: Chris Beckmann, Edward De Guzman, Kun Gao, Nathan Good, Jeff Heer, Gary Hsieh, Scott Lederer, Tara Matthews, Ana Ramírez, and Rusty Sears.

This dissertation research was supported through a grant from the National Science Foundation (IIS-0084367), a summer internship at Ricoh Innovations, and equipment donations from SMART Technologies and the Intel Corporation.

My time at Berkeley was spent in the Human-Centered Computing laboratory in Soda Hall and later in the Berkeley Institute for Design in the Hearst Mining Building. I thank John Canny for his research advice and for his impressive ability in securing this space, the most precious commodity on any university campus. I have been fortunate to

have a wonderful group of colleagues to work with in these labs—Francesca Barrientos, Danyel Fisher, Eric Paulos, and Dan Reznik were my comrades in Soda Hall, and I joined Chris Beckmann, Scott Carter, Dan Glaser, Jeff Heer, Jonathan Hey, Matt Kam, Scott Lederer, Alan Newberger, David Nguyen, and Jeremy Risner as the original occupants of BID.

I've had an amazing group of collaborators in the Group for User Interface Research—Mark Bilezikjian, Kate Everitt, Ryan Farrell, Robert Lee, Jack Li, Jimmy Lin, Mark Newman, Ethan Phelps-Goodman, Anoop Sinha, Michael Thomsen; and at Ricoh Innovations—Jamey Graham, Jonathan Hull, and Gregory Wolff. I have also benefited from all of the other GUIR-heads and HCI colleagues, especially Marc Davis, Anind Dey, Ame Elliott, Marti Hearst, Jason Hong, and Sarah Waterson. I owe a great deal to my dissertation committee: Jen Mankoff, Terry Winograd (who I am now fortunate to have as a colleague!), and Paul Wright. My advisor, James Landay, has been a tremendous mentor to me over the past five years. He created a wonderfully tight-knit research group and home for HCI at Berkeley. With his boundless energy and charisma, he taught me self-confidence as a researcher and teacher.

Midway through graduate school, I spent four months living and working in Barcelona. I thank Josep Blat and his Grup de Tecnologies Interactives at the Universitat Pompeu Fabra for hosting me. My housemates Javier López Prieto and Marçal Nebot made Barcelona a home for me. As did Sofía Gamallo, Rocío García Robles, Dai Griffiths, Toni Navarrete, Patricia Puentes González, and Núria Reixach from the UPF.

Thank you to my friends who have shared their lives with me as my Bay Area family—Aaron Watson, my close friend for many years and housemate in 0-town for the last two; Daniel Scarpelli and Dorick Scarpelli, my wicked good friends; Jane van Gunst, one of the most thoughtful people I know; Phil Buonadonna and Jason Hill, for

all the mountain biking we've shared in the summer and snow in the winter. Also to Rozy Fredericks, Anders Fristedt, Elizabeth Goodman, Ian Jones, Nalini Kotamraju, Jacob Tonski, Becca Weider, Kamin Whitehouse, and Amanda Williams.

Lastly, to my sister Sandy, and my parents Cheryl and Chris. I have been incredibly fortunate to have your love, friendship, and wisdom. Thank you.

1 Introduction

> *"We interact with documents in two separate worlds: the electronic world of the workstation, and the physical world of the desk. Interaction styles in these two worlds do not resemble each other, functions available are different, and interaction between the two is limited"*
>
> —Pierre Wellner, 1993 [260, p. 87].

While people's interaction with tools in the real world is highly nuanced and heterogeneous, traditional graphical user interfaces map all of our tasks in the electronic world onto a small set of tools. Tangible user interfaces (TUIs) [117] address this by employing the physical tools we have facility with to control digital information.

While many research groups have developed compelling applications, nearly all of these have been developed from scratch without the benefit of user interface development tools. Developing tangible interfaces is problematic because programmers are responsible for acquiring and abstracting physical input. This is difficult, time-consuming, and requires a high level of technical expertise in fields very different from user interface development—especially in the case of computer vision. Consequently, only a small cadre of technology experts can currently build these UIs. The difficulty of technology development and lack of appropriate interaction abstractions make designing different variations of an application and performing comparative evaluations unrealistic. These difficulties echo the experiences of developing GUIs twenty years ago, when substantial raster-graphics expertise was required. One of the earliest GUI toolkits,

MacApp [221], reduced application development time by a factor of four or five. Similar reductions in development time, with corresponding increases in software reliability and technology portability, can be achieved by a toolkit supporting tangible interaction.

1.1 Thesis Contributions

This thesis offers contributions in three areas.

1 Toolkit support for tangible user interface input. The Papier-Mâché toolkit introduces a novel software architecture that:

   A Lowers the threshold for developing applications that employ tangible user interface input. This is accomplished with high-level abstractions of input technologies.

   B Supports switching input technologies with minimal code changes. The architecture structures input from all devices in a similar fashion.

   C Makes debugging easier through monitoring facilities that include Wizard of Oz control.

2 Interaction techniques that employ tangible user interface input to support professional work practices. The difficulty of implementing these applications inspired our research on toolkit support for tangible input.

   A The Designers' Outpost integrates wall-scale, paper-based design practices with novel electronic tools to better support collaboration for early-phase design. This integration is especially helpful for fluidly transitioning to other design tools; access and exploration of design history; and remote collaboration.

   B The Books with Voices system introduces an augmented paper UI providing fast, random access to digital video while retaining the paper-based transcripts preferred by oral historians.

3 Improved user-centered methods for the design and evaluation of software tools.

   A Fieldwork with developers as a basis for the design of software tools, in order to better learn what software developers are really doing and what tool support would be beneficial.

   B A triangulation method comprising controlled laboratory study, monitoring of longer-term use in projects, and external metrics such as performance and code size for evaluating the usability of software APIs.

1.2 Solution Overview

In this section, we outline how this dissertation addresses the thesis contributions.

1.2.1 Toolkit support for tangible user interface input

Developing tangible interfaces is problematic because programmers are responsible for acquiring and abstracting physical input. This is difficult, time-consuming, and requires a high level of technical expertise in a field very different from user interface development—especially with computer vision. These difficulties echo the experiences of developing GUIs twenty years ago. An early GUI toolkit, MacApp, reduced application development time by a factor of four or five [187]. Similar reductions in development time, with corresponding increases in software reliability [94] and technology portability, can be achieved by a toolkit supporting tangible interaction.

The central piece of this dissertation research is Papier-Mâché, a toolkit that lowers the threshold for developing tangible user interfaces. This toolkit enables programmers who are not input hardware experts to develop TUIs, as GUI toolkits have enabled programmers who are not raster graphics experts to build GUIs. Papier-Mâché's library supports several types of physical input: interactive computer vision, electronic tags, and barcodes. Through technology-independent input abstractions, Papier-Mâché also

improves application flexibility, allowing developers to retarget an application to a different input technology with minimal code changes. This support is important because it enables application developers to explore different versions of an application and perform comparative evaluations based on technology cost, performance, usability, and other important metrics.

A significant difficulty in program debugging is the limited visibility of application behavior [62] (§ 7.2). The novel hardware used in tangible interfaces, and the algorithmic complexity of computer vision, only exacerbate this problem. To support debugging, Papier-Mâché provides monitoring facilities that display the current input objects, image input and processing, and behaviors being created or invoked. The monitoring window also provides Wizard of Oz (WOz) [128, 172] generation and removal of input; its graphical WOz support is the richest of any post-WIMP toolkit. WOz control is useful for simulating hardware when it is not available, and for reproducing scenarios during development and debugging.

1.2.2 TUI input supporting professional work practices

The design of Papier-Mâché has been deeply influenced by my experiences building physical interfaces over the past several years. Experiential knowledge is very powerful —toolkit designers with prior experience building relevant applications are in a stronger position to design truly useful abstractions. As background research for this dissertation, we designed and built two significant tangible user interfaces.

For three decades, pundits have touted the imminent arrival of the paperless office. However, paper remains a central artifact in professional work practices, and use of paper is steadily increasing. It is tangible, portable, readily manipulable, and easily editable. Newman and Landay's study of web design practice found that pens, paper, walls, and tables were often used for explaining, developing, and communicating ideas

during the early phases of design [191]. Designers prefer these tools because they are flexible, immersive, and calm. Additionally, a wall-scale workspace allows multiple people to simultaneously view, discuss, and modify a design (see FIGURE 1.1). However, when using paper and walls for design, it is difficult to maintain structuring marks (such as links and annotations), create multiple versions, or collaborate with designers at another location (these fieldwork results are described in § 3.1.1).

These wall-scale, paper-based practices inspired The Designers' Outpost, a tangible user interface that combines the affordances of paper and large physical workspaces with the advantages of electronic media to support information design. With Outpost, users collaboratively author web site information architectures on an electronic whiteboard using physical media (Post-it notes and images), structuring and annotating

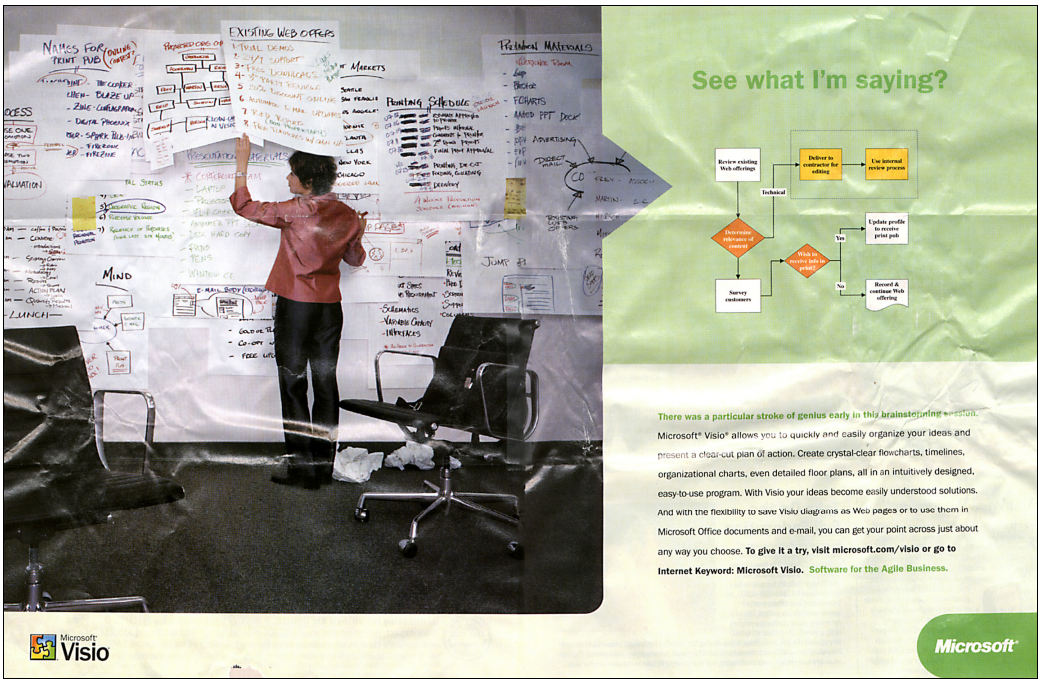


FIGURE 1.1 A magazine advertisement for the Microsoft Visio software. This ad shows an office professional after a meeting; she is left with the task of migrating the physical artifacts produced in the meeting to electronic form. The ad suggests that abandoning physical artifacts for electronic ones eliminates this migration step. While true, it also removes all of the fluid interactions available with physical artifacts. This dissertation work provides tools and techniques that couple fluid physical interaction with the capture and access advantages of electronic tools.

that information with electronic pens. This interaction is enabled by a touch-sensitive SMART Board [21] augmented with a computer vision system. Thus, paper in the physical world becomes an input device for the electronic world. A rear-mounted projector outputs electronic information onto surfaces in the physical world. Through its electronic capture of designs, Outpost supports the transition from this early representation to later electronic tools. Over the three years of the project, we conducted in-lab evaluations of Outpost with 27 users. These studies were part of our iterative design process where we assessed the usability of the system at each stage of development, and gathered information that fueled subsequent development.

The Designers' Outpost supports informal history capture and retrieval. Its history interface comprises three novel visualizations for collaborative early-phase design. The most important of which, the main timeline, is a visually navigable, time-ordered sequence of design thumbnails. Our focus on early-phase design led us to fluid, informal techniques that capture information that users produce in the normal course of their activities, structuring this information for later retrieval.

To better support remote collaboration, The Designers' Outpost provides an interaction paradigm where objects that are physical in one space are electronic in the other space, and vice-versa. This system also introduced two remote awareness mechanisms: transient ink input for gestures and a shadow of the remote collaborator for presence.

Our second major TUI resulted from a contextual inquiry I conducted into the practices of oral historians. This study unearthed a curious incongruity: while oral historians consider interview recordings a central historical artifact, these recordings sit unused after a written transcript is produced. This is largely because paper books are more usable than recordings. Therefore, we created Books with Voices: barcode-augmented paper transcripts enabling fast, random access to digital video interviews on a PDA. I conducted a quantitative evaluation of Books with Voices with 13 oral

historians and book-club members. Their response was overwhelmingly positive. The study showed that this calm interface allows users to concentrate on the task and stay in the flow. Participants repeatedly accessed recordings while reading, finding a level of emotion in the recorded video interview not available in the corresponding printed transcript. The video also helped readers clarify the text and observe nonverbal cues. We often look to new technologies, like Books with Voices, to save labor. Participants accomplished with just a few button presses what would have otherwise consumed hours. More importantly, this system makes a richer practice tractable. Books with Voices augments reading with an audiovisual experience previously unavailable.

The Designers' Outpost and Books with Voices could each be implemented in a few person-months of software development time if the interaction was purely graphical. Additionally, most professional software developers would have little difficulty implementing either of these systems as a GUI from a specification. Introducing a tangible interface substantially extended the development time, and required much more technical expertise.

1.2.3 User-centered methods for the design and evaluation of software tools

This dissertation offers two methodological contributions. 1) This is the first research to employ fieldwork *with developers* as a methodological basis for toolkit design. A toolkit is software where the "user interface" is an API, and the users are programmers. As part of our user-centered design process, I conducted structured interviews with nine researchers who have built tangible interfaces. 2) This dissertation introduces a method of "triangulating" the usability of an API through the application of multiple evaluation methods. Different usability methods provide different results [175]. For example, a laboratory study offers substantial control over the evaluation; however, it is an artificial task. Monitoring the use of a software tool in practice is a highly authentic method;

however, the experimenter has little control over the task. Our triangulation method comprises controlled laboratory study, monitoring of longer-term use in projects, reimplementing existing applications, and traditional software engineering metrics such as performance and code size for evaluating the usability of software APIs. Aggregating data from these diverse methods affords a much richer picture of the usability of a programming tool.

1.3 Dissertation Roadmap

Chapter 2 covers related work, beginning with the motivations for tangible interaction (§ 2.1, 2.2, and 2.3). It then discusses related work in user interface software tools (§ 2.4), GUI input models (§ 2.5), ubiquitous computing tools (§ 2.6), the 24 applications that inspired the toolkit needs of Papier-Mâché (§ 2.8), technology considerations for TUI design (§ 2.9), and prior work on user-centered evaluation of programming languages (§ 2.10).

1.3.1 Interaction techniques for tangible user interface input

In chapter 3 through chapter 6, we describe the motivating fieldwork, iterative designs, and evaluations of two applications we have built that employ paper as a tangible interface. Chapter 3 introduces the Designers' Outpost system, which provides a tangible user interface for collaborative web site design. It begins with the current wall-scale practices that inspired our system (§ 3.2). It then discusses our formative design studies (§ 3.3), Outpost's interaction techniques (§ 3.4), the study we conducted with web design professionals (§ 3.5), and the design implications resulting from the study (§ 3.6). The last third of the chapter discusses the computer vision prototypes (§ 3.7), and the current vision system and physical tools input (§ 3.8).

Chapter 4 discusses Outpost's support for design history. It begins by reviewing prior research in design rationale and history-enabled applications (§ 4.2), and how current design practices and designers' experiences with the basic Outpost system motivated us to research tool support for design history (§ 4.3). It then presents the design history interface (§ 4.4), four scenarios that illustrate the utility of design history (§ 4.5), and the software architecture for the design history implementation (§ 4.6). The chapter closes with the design study that we conducted (§ 4.7).

Chapter 5 describes our remote collaboration research with Outpost, beginning with prior work in distributed media spaces (§ 5.2) and methods that designers currently use for remote collaboration (§ 5.3). It then presents the interaction techniques we have developed (§ 5.4) and the software infrastructure that enables these techniques (§ 5.5). The chapter closes with feedback from users of the system (§ 5.6).

Chapter 6 presents Books with Voices, which provides barcode-augmented paper transcripts for random access to digital video. It begins with the fieldwork we conducted into oral history practice (§ 6.2) and related work (§ 6.3). It then describes the paper mock-ups that we designed (§ 6.4) and the interactive system (§ 6.5). It concludes with the evaluation of Books with Voices (§ 6.6).

1.3.2 Software tools for tangible user interface input

Based on this experience, we created the Papier-Mâché toolkit, described in chapters 7 through 9. Chapter 7 discusses the findings of our structured interviews with nine tui developers. It begins with broad practices and goals: that acquiring and abstracting input is difficult (§ 7.1), that development is conducted iteratively with familiar tools (§ 7.2), and that often, user experience goals motivated the interest in tangible interaction (§ 7.3). It then describes concrete development issues: code reuse (§ 7.4), the appropriateness of events (§ 7.5), a declarative programming model (§ 7.6), choice of

programming language (§ 7.7), choice of input technology (§ 7.8), distributed applications (§ 7.9), and the need for improved feedback (§ 7.10).

Chapter 8 describes the Papier-Mâché software architecture, beginning with an overview (§ 8.1). It begins with how input is abstracted and high-level events are generated (§ 8.2), discusses the relationships between events and application behavior (§ 8.3), the flow of control in an application (§ 8.4), how input technologies can be easily interchanged (§ 8.5), and how the Papier-Mâché's input model is distinct from GUI input models (§ 8.6). Next, we discuss Papier-Mâché's visual interface for monitoring application behavior (§ 8.7) and creating application behaviors visually (§ 8.8).

Chapter 9 presents the evaluation of Papier-Mâché. It begins with a discussion of the relative merits of different evaluation techniques (§ 9.1). We then describe our evaluation of Papier-Mâché in terms of performance (§ 9.2) and lines of code (§ 9.3). The chapter then covers the laboratory evaluation of Papier-Mâché (§ 9.4), observations of its longitudinal use in class and research projects (§ 9.5), and three of the inspiring applications rewritten using Papier-Mâché (§ 9.6).

Chapter 10 concludes the dissertation with a summary of contributions and general research directions for the future of user interface design tools.

2 Related Work

This dissertation research builds on prior work in a number of areas. This chapter describes this prior work, how it has inspired this dissertation, and the contributions that this dissertation offers beyond existing research.

The first three sections of this chapter address the field of tangible interaction as a whole. Section 2.1 overviews the benefits of the physical world, the digital world, and motivations for and difficulties with combining these. Section 2.2 introduces seminal research on tangible interaction. Section 2.3 summarizes fieldwork and laboratory evaluations illustrating the advantages of tangible interaction.

The next three sections address software tools. Section 2.4 outlines the benefits of user interface software tools. Section 2.5 reviews software architectures for graphical user interface input. Section 2.6 presents tool support for ubiquitous computing.

The following three sections address TUI taxonomies. In section 2.7, we present taxonomies for tangible UIs. Section 2.8 offers our taxonomy of tangible input, and places 24 applications that have inspired Papier-Mâché into this taxonomy. Section 2.9 takes a technology-centric perspective, looking at the various input technologies that are used in tangible interfaces and their relative merits and drawbacks.

The chapter closes with section 2.10, which reviews existing methods for evaluating programming tools, programming languages, and software architectures.

2.1 Introduction

In the *Myth of the Paperless Office* [222], Sellen and Harper describe their multi-year field research on document use in the office. The central thesis of the book is that while paper is often viewed as inefficient and passé, in actuality it is a nuanced, efficient, highly effective technology. The authors are not asserting that paper is superior to digital technologies or vice versa, but that the naïve utopia of the paperless office is mistaken. Digital technologies certainly change paper practices, but they rarely make paper irrelevant. An example the authors give is document distribution. Prior to networked computers, nearly all documents were centrally printed and distributed in *physical form*, through the postal service. Today, lengthy documents are still generally read on paper, however *electronic* distribution through email is becoming more common.

The paper-saturated office is not a failing of digital technology; it is a validation of our expertise with the physical world. We use paper, and writing surfaces more generally, in their myriad forms: books, notepads, whiteboards, Post-it notes, and diagrams. We use these physical artifacts to read, take notes, design, edit, and plan.

There are excellent reasons for researchers to embrace, not abandon, our interactions with everyday objects in the physical world. Paper and other everyday objects:

- Leverage our prehensile abilities for grasping and manipulating physical objects [161]
- Allow users to continue their familiar work practices, yielding safer interfaces [156, 158]
- Are persistent when technology fails, and thereby more robust [173]
- Enable lighter-weight interaction [107]
- Afford for fluid collocated collaboration [34, 191, 192]
- Are higher resolution, and easier to read than current electronic displays [220, 228 §13.3]
- Provide a verifiable record of transactions for tasks such as voting [66]

However, "tangible computing is of interest precisely because it is not purely physical" [67, p. 207]: it is the computational augmentation that defines the field. Researchers have electronically augmented paper and other everyday objects to offer:

- An interactive history of an evolving physical artifact (see chapter 4)
- Collaboration among physically distributed groups (see chapter 5)
- Enhanced reading [29, 228 §13.2, 237] (also see chapter 6)
- Physical links to electronic resources [29, 109, 204, 208, 237, 255] (also see chapter 6)
- Physical handles for fluid editing of electronic media [250]
- Automated workflow actions [96, 107, 122]

Additionally, data in the physical world (*e.g.*, post-it notes on a wall, cards in a card catalog, specimens in a biology collection) can only be organized according to one schema at any point in time. (Physical redundancy offers multiple views, but it can be time-consuming to produce and difficult to maintain.) Electronic augmentation offers the automatic generation of multiple perspectives on a physical data source.

However, there are difficulties in employing paper and everyday objects as a user interface. When paper is used as an interactive dialog, the update cycle (printing) is much slower than with electronic displays. When multiple actors (computational or human) control the physical data (*e.g.*, Post-it notes), the application needs to reconcile the physical objects representing an inconsistent view. This can be handled by either prohibiting such actions, or by electronically mediating them. Physical sensors (especially computer vision) require substantial technological expertise to be built robustly. While many developers have excellent ideas about how physical computing can better support a task, few have this technological expertise. The art of designing tangible interfaces is leveraging the unique strengths that the physical and electronic

worlds have to offer, rather than naïvely replicating the interaction models of one paradigm in the other [130].

2.2 Origins of Tangible Interaction

In the 1970's, Xerox PARC transformed the computing world with the graphical user interface [121, 231]. In the early 1990's, another sea change was afoot at PARC. Looking beyond the desktop, Mark Weiser and colleagues embarked on research into a future of ubiquitous computing. In contrast with virtual reality, which attempts to recreate the physical world electronically, ubiquitous computing provides "embodied virtuality": it embeds the electronic world in the physical one [257]. This PARC research group created "computing by the inch, foot, and yard." This yielded ParcTabs [256] (handheld computers), ParcPads [125] (tablet computers), and LiveBoards [72] (electronic whiteboards), respectively. While this vision provided the catalyst for a worldwide research effort on off-the-desktop, highly networked computing, these early research prototypes provided direct stylus input for small-, medium-, and large-scale raster-graphics displays: the fundamental method for user input was largely similar to the desktop computing standard.

Weiser's vision of integrating computation into the physical world also encouraged researchers to begin exploring interactions that integrate the physical and electronic worlds. Three seminal research projects in this direction are Wellner's DigitalDesk [260], Fitzmaurice *et al.*'s Bricks [81], and Ishii and Ullmer's Tangible Bits [117]. We discuss each of these in turn.

2.2.1 Interacting with paper on the DigitalDesk

Pierre Wellner and colleagues at Rank Xerox EuroPARC introduced the idea of an interface that bridges the physical and electronic worlds. Their DigitalDesk system comprises a ceiling-mounted projector that projects onto a physical desk and an electronic camera that tracks the movements of user's hands on the desk (see FIGURE 2.1). The camera also captures objects on the desk at standard video resolution, and a second camera is zoomed in on a special area of the desk, affording higher resolution capture. With this system, users can select areas of physical documents to be copied and pasted electronically. The video capture of the physical desk can also be displayed on a remote user's desktop. As a proof-of-concept demonstration, the authors use this for a game of tic-tac-toe between remote participants. This system helped inspire our interest in this area in general, and our work on The Designers' Outpost in particular. We also used this application to help shape the computer vision requirements for Papier-Mâché.

2.2.2 Bricks: laying the foundations for graspable user interfaces

Fitzmaurice, Ishii, and Buxton introduced the idea of physical bricks that can be used as handles for manipulating electronic content [79-81] (see FIGURE 2.1), offering a true,

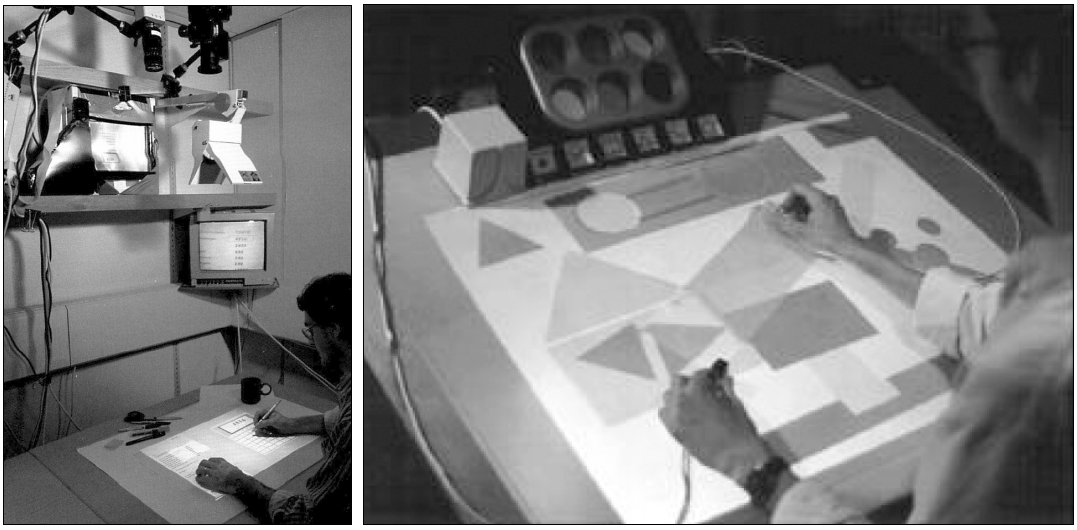


FIGURE 2.1 Wellner's DigitalDesk (*left*) and Fitzmaurice *et al.*'s Bricks (*right*).

direct manipulation interface. They observed that, "today an accountant, animator and graphic designer, all use the same input device set-up (*i.e.*, a keyboard and mouse) for performing their very diverse activities. This 'universal set-up' seems inefficient for users who work in a specific domain. The mouse is a general all-purpose weak device; it can be used for many diverse tasks but may not do any one fairly well" [80] (p. 50). They offer the distinction between *space-multiplexed* input and *time-multiplexed* input. An audio mixing board is space-multiplexed: there is a one-to-one mapping between controls and functions. A mouse is time-multiplexed: a single input device controls all functions, and the function controlled changes over time. Bricks exhibit properties of both paradigms. It is space-multiplexed in that multiple elements can be operated in parallel, and each has their own controller; it is time-multiplexed in that the mapping between controls and functions is reconfigurable. Papier-Mâché supports both space-multiplexed input and time-multiplexed input, as well as facilities for attaching a physical input to an electronic control.

The bricks work included explorative evaluations of physical prototypes. A design study found a graspable interface (a stretchable box) was roughly an order of magnitude faster than the traditional MacDraw interface for positioning, rotating, and stretching rectangles. The authors implemented a bricks interface to Alias Studio, a high-end 3D modeling and animation program. The evaluation found that, "all of the approximately 20 users who have tried the interface perform parallel operations (*e.g.*, translate and rotate) at a very early stage of using the application. Within a few minutes of using the application, users become very adept at making drawings and manipulating virtual objects" [81, p. 447]. The experimental psychology literature (*e.g.*, [161]) has shown the impressive dexterity of human hands in working with physical objects. Bricks leverage this dexterity; the authors' direct comparison of bricks and graphical UIs provides empirical justification that tangible manipulation can be both faster, and more

intuitively bimanual than graphical manipulation. In the commercial world, applications such as Photoshop are often used in a bimanual fashion: the non-dominant hand selects tools using the keyboard and the dominant hand performs precision, positional input.

2.2.3 Tangible bits

In 1997, Ishii and Ullmer introduced their Tangible Bits research agenda [117]. This paper and the group's substantial subsequent work [115] offer a broad array of techniques for physically interacting with the digital world, and was one of the inspirations for this dissertation. Their work includes physical interaction techniques for both foreground (graspable) and background (ambient) interaction. The Papier-Mâché input toolkit provides excellent support for the foreground techniques described in this paper. Several of these systems are included in our 24 inspiring applications; see SECTION 2.8 for descriptions of systems by the tangible media group. Papier-Mâché does not support ambient interfaces, as these UIs are display (output) only.

2.3 Motivating and Evaluating Tangible Interaction

The emerging literature evaluating tangible UIs is highly encouraging. In this section, we present both fieldwork motivating the need for tangible interaction, as well as a comparative evaluation of a tangible interface with the paper practice that inspired it.

2.3.1 Walls for collaborative design

Bellotti and Rogers conducted a study on web publishing workflow [34]. They too discovered a tension between paper-based practices and electronic practices. In particular, they found that people were often more comfortable working on paper, but felt that electronic tools were beneficial for stronger communication and awareness among distributed teams. One site director commented, "What I would love would be a flat panel I could hang on a wall… For the tacked up paper and string setup we have, a

video wall could be really useful, not just for the sake of more expensive equipment, but for working with remote group members, for ease of modification, and for keeping a better record of the evolution of the site" [34, p. 284]. This study helped motivate our interest in combining the physical and electronic worlds to gain these benefits.

2.3.2 Web site design practice

Newman and Landay's investigation into web design practice [191, 192] found that pens, whiteboards, paper, walls, and tables were the primary tools used for explaining, developing, and communicating ideas during the early phases of design. Later-phase design, where detailed page mockups are generated, occurs mostly on the computer. This finding is consistent with work practice studies across many design and engineering domains [34, 114, 262].

In one common early-phase practice, designers collect ideas about what should be in a web site onto Post-it notes and arrange them on the wall into categories. This technique, often called affinity diagramming [37], is a form of collaborative "sketching" used to determine the site structure. We have built a tool, The Designers' Outpost, that supports this practice (see chapters 3, 4, and 5).

2.3.3 Paper flight strips

Mackay *et al.* [156, 158] conducted fieldwork with air traffic controllers, a well-studied domain in cscw. Controllers manage and share information through paper-based flight strips (see figure 2.2). While the current system was safe, "No fatalities have ever been attributed to French civilian controllers," [158, p. 558] increased air traffic was encouraging automation, and the computers that were part of the process were old and needed to be replaced, providing a window of opportunity for new technologies.

While most researchers, "acknowledge the importance of flight strips, they generally concentrate on the information they contain, rather than the controllers' interactions with them." These attempts to replace the physical world of ATC with a GUI "have ultimately been rejected by the controllers." Mackay found "Automation need not require getting rid of paper strips. We suggest keeping the existing paper flight strips as physical objects, with all their subtlety and flexibility, and augmenting them directly by capturing and displaying information to the controllers." She also felt this provided a more domain appropriate interface than "the old mouse and keyboard designed for office automation."

Mackay has built several other augmented paper systems, including a DigitalDesk-based system for video editing [159] and, more recently, a system for augmenting biologists' laboratory notebooks [160]. This rigorous fieldwork shows there is nuance in the physical world that electronic systems should preserve. Papier-Mâché facilitates rapid development of systems such as these three, enabling iterative design with end-users.

FIGURE 2.2 Air traffic controllers working with paper flight strips.

2.3.4 Comparing paper and tangible multimodal tools

McGee *et al.*'s Rasa system extends the physical wall-mounted map and Post-it note tools currently used in military command post settings with a touch-sensitive SMART Board behind the map, gesture recognition on ink strokes written on the Post-it notes, and speech recognition on verbal commands [57, 174]. They performed a comparative evaluation of Rasa and traditional paper tools with six users [173]. To ascertain Rasa's robustness, the authors shut off the computer halfway through the study (without telling the users, or even letting on that they comprehended what happened). They found that users were able to keep working through a power failure, and with minimal effort, return the system to a consistent state after power was restored, showing "that by combining paper and digital tools, we have constructed a hybrid system that supports the continuation of work in spite of power, communications, and hardware or software failures."

However, the domain of technology support for military activity hides several shortcomings with recognition- and transformation-laden interfaces. These systems leverage existing military practices of a precise, consistent grammar for conveying information, and this highly constrained vocabulary and notation greatly eases recognition tasks. As such, tool support for military activity has little bearing on the much more fluid and ad hoc practices of most professional and domestic life. Over the next decade, recognition will play a larger role in human-computer interaction, and our group's research agenda of informal interfaces [145] is part of this trend. While informal interfaces incorporate more recognition technologies (*e.g.*, gesture recognition, or vision recognition of Post-its) than WIMP UIs, good informal interface designers work hard to minimize the recognition and most importantly, minimize the transformation of users' input.

2.4 User Interface Software Tools

The difficulties involved in building tangible interfaces today echo the experiences of the GUI community of twenty years ago. In the early 1990s, Myers and Rosson found that 48% of code and 50% of development time was devoted to the user interface [187]. One of the earliest GUI toolkits, MacApp, reduced Apple's development time by a factor of four or five [221]. We believe that similar reductions in development time, with corresponding increase in software reliability and technology portability, can be achieved by a toolkit supporting tangible interaction.

While the research community has shown the substantial benefits of tangible interaction, these UIs are currently very difficult and time consuming to build, and the required technology expertise limits the development community. The difficulty of technology development and lack of appropriate interaction abstractions make designing different variations of an application and performing comparative evaluations unrealistic. In each of the twenty-four research systems we have studied, at least one member of the project team was an expert in the sensing technology used. Contrast this with GUIs, where developers are generally experts in the domain of the application, not in raster-graphics manipulation.

GUI tools have been so successful because, "tools help reduce the amount of code that programmers need to produce when creating a user interface, and they allow user interfaces to be created more quickly. This, in turn, enables more rapid prototyping and, therefore, more iterations of iterative design that is a crucial component of achieving high quality user interfaces" [188, p. 5]. Our analysis of the Outpost code base (see SECTION 3.9) is inspired by this analysis.

2.5 GUI Input Models

Papier-Mâché's software architecture for input is inspired by software design patterns for handling input in graphical user interfaces. The elements of graphical user interfaces such as buttons, check boxes, and scroll bars are called *widget*s: small interactive objects that perform some editing or input task [199].

2.5.1 Model-View-Controller

Model-View-Controller (MVC) [137, 199] is a software design pattern for developing GUIs that separates each widget into three pieces. In MVC-style user interfaces, a *controller* (input handler) sends input events to a *model* (application logic), and the model sends application events to a *view*. There are two reasons for separating this functionality. First, there may be multiple methods for manipulating and/or presenting a particular piece of application logic. For example, a set of nodes may be displayed as both a graph and a list. Second, this separation enables developers to more easily change the way that an element is manipulated or displayed.

In practice, the communication between the view and the controller is highly implementation dependent, and substantially altering one piece generally requires altering the other as well. For this reason, most contemporary GUI toolkits provide widgets that comprise both the controller and the view. In many toolkits, such as Java Swing [3], widgets also include a mini-model that stores the state of the widget. In these systems, the widget's mini-model is responsible for communication with the application model; events are used for this communication.

2.5.2 Interactors

The Garnet toolkit [184] introduced Interactors [182], which are an extension of the MVC architecture providing higher-level input events. In Garnet, Interactors are the

controller, object-oriented graphics are the view, and Lisp code is the model. Interactors are also featured in Garnet's successor, Amulet [183, 186]. Interactors leverage Foley and Wallace's observation that graphical interaction with a computer has a small number of basic input types [82, 254]. The six original Interactors are *menu*, *move-grow*, *new-point*, *angle*, *text*, and *trace*. This higher-level API shields application developers from implementation details such as windowing systems. It also separates view and controller more cleanly than the original MVC. Interactors are highly parameterized, minimizing the need to write custom input-handling code. Parameters include the start event (such as pressing the left mouse button), interim and final feedback to show, and a command object [86, pp. 233-242] to execute upon completion. These six Interactors fully span the space of traditional WIMP interaction techniques [186]. It is possible to attach multiple Interactors to a view object (*e.g.*, pressing the left mouse button operates a window, and pressing the right button moves that window), and it is possible for multiple Interactors to operate simultaneously (an important benefit of this is support for bimanual input).

It should be noted that Interactors, like most input architectures, does not cleanly handle keyboard shortcuts. In most GUI architectures, all input is routed to a focus widget (such as a button), and the Interactors attached to the widget handle that input. Keyboard shortcuts operate globally, and therefore routing to the focus object is not correct. To avoid this, most architectures introduce an additional component that monitors keyboard input for keystrokes that are shortcuts, and handle this input separately.

Additionally, the Interactors architecture loses elegance for input beyond traditional WIMP interfaces. Interactors have been extended to include gesture [143, 183], and other modalities, such as speech, have been proposed [186]. Two important axes of input include the input *mode* (*e.g.*, WIMP, gesture, or speech) and the type of input *action* (*e.g.*,

select, create, or move). The original six Interactors are a set of *action* types that span the WIMP *modality*. WIMP, Gesture, and speech are distinct *modality* types, each of which can be used for many *actions*. The conflation of these axes with Interactors indicates that a more elegant architecture should separate mode from action. Papier-Mâché offers this separation.

2.5.3 SubArctic

Hudson and Smith's SubArctic is a constraint-based GUI toolkit written in Java. Its input handling [112] is similar to that in Garnet and Amulet: all three employ an Interactors architecture where all Interactors are stored in a tree organized by the spatial containment of widgets on the screen. It is somewhat more flexible than Garnet or Amulet in that it provides an extensible policy architecture for input dispatch. The two most common policies are *positional* (send input to the Interactor directly beneath the cursor) and *focus* (send input to the Interactor that has the application's focus). Dispatch agents translate low-level input events (e.g., *mouse press* and *mouse move*) into higher-level events (e.g., *dragging*). A manager provides a priority list of *policies* and *agents*. Policies and agents can receive events they are interested in, and *absorb* these events if policies and agents further down the list should not receive them. An example of an input policy that would use but not absorb input is a system that logs all application events. In general, interaction with widgets is absorbed by the relevant Interactor.

2.5.4 Specifying non-WIMP user interfaces

In [119], Jacob *et al.* introduce a software input model and specification language for non-WIMP user interfaces, such as virtual environments. Like Papier-Mâché, this input model is designed to support parallel, continuous, multimodal input. The system offers

a dataflow programming model for continuous input and an event model for discrete input. Input and application data are connected using one-way constraints.

Dataflow programming and constraints are most commonly encountered in spreadsheets. A simple example of a constraint is a spreadsheet cell whose value is defined (constrained) to be twice that of its neighbor. Introduced in the SketchPad system [240], constraints have been used in UI toolkits such as Garnet [184], Amulet [186], SubArctic [113], and Bramble [90]. Most contemporary UI toolkits, such as Java Swing [3], offer rudimentary constraints for widget layout. However, in general, constraints are a promising method that has not achieved widespread adoption [188, § 2.3.3].

Jacob *et al.* argue that dataflow programming and constraints offer a conceptual model that is more appropriate for programming non-WIMP UIs than events. While an interesting hypothesis, there is little evidence to support the claim. There is a range of programming models between completely event-based and completely constraint-based systems for coupling input with application behavior. In practice, nearly all systems, Jacob's and Papier-Mâché included, use constraints in some places and events in others. Papier-Mâché uses constraints for defining the mapping between an input object and application logic, and events for allowing the application logic to flexibly respond to the input (see CHAPTER 8). For post-WIMP user interfaces, the balance between events and constraints has not been rigorously explored, and is a fruitful area for further research.

2.6 Tool Support for Ubiquitous Computing

In the last five years, the research community has developed several tools to support ubiquitous computing applications. This section outlines existing tools for tangible interfaces, as well as other related ubicomp tools.

2.6.1 Phidgets: programmable physical widgets

The work most related to Papier-Mâché is Phidgets [99]. Phidgets are physical widgets: programmable ActiveX controls that encapsulate communication with USB-attached physical devices, such as a switch, pressure sensor, or servomotor (see FIGURE 2.3). Phidgets are a great step towards toolkits for tangible interfaces. The graphical ActiveX controls, like our monitoring window, provide an electronic representation of physical state. However, Phidgets and Papier-Mâché address different classes of tangible interfaces. Phidgets primarily support tethered, mechatronic TUIs that can be composed of powered, wired sensors and actuators. Papier-Mâché supports TUI input from untethered, passive objects, often requiring computer vision.

Papier-Mâché provides stronger support for the "insides of applications" [186] than Phidgets. Phidgets facilitate the development of widget-like physical controls (such as buttons and sliders), but provide no support for the creation, editing, capture, and analysis of physical input, which Papier-Mâché supports.

One of the reasons for Phidgets' success is that the Greenberg's original research

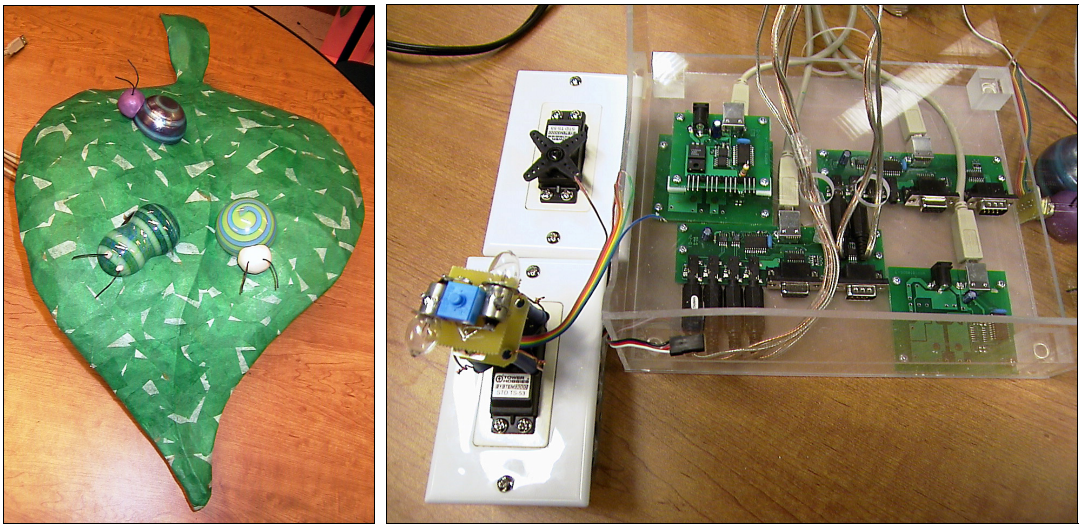


FIGURE 2.3 BuddyBugs (*left*) is a tangible interface for instant messaging [176]. Each bug represents an IM contact; the contact's status is represented by the bug's orientation. To initiate contact, a user taps on a bug avatar. It was created using Phidgets pressure sensors and servomotors *(right)*.

agenda was not a toolkit. The authors were originally interested in building applications (*e.g.*, [98]). It was the difficulty and frustration of continuing with a research agenda of building and evaluating one-off applications that led the authors to develop tools. The Shiloh's Teleo toolkit [225] is quite similar to Phidgets in concept, and also arose out of the difficulty of repeatedly building custom applications. In the Teleo case, the original applications were controllers for the robotic performance art of Survival Research Laboratories [201]. The primary distinction between Teleo and Phidgets is that Teleo is programmed with Max/MSP [19] or Macromedia Flash [10], rather than the Microsoft Visual Studio environment.

Experiential knowledge is very powerful; toolkit designers with prior experience building relevant applications are in a much better position to design truly useful abstractions. We are in a similar position of experience with Papier-Mâché.

2.6.2 IStuff

IStuff [30] introduces compelling extensions to the Phidgets concept, primarily support for wireless devices. IStuff, in conjunction with the Patch Panel [31], provides fast remapping of input devices into the iRoom framework [120], enabling standard GUIs to be controlled by novel input technologies. There are two main differences in our research agenda: First, like Phidgets, iStuff targets mechatronic tangible interfaces, rather than TUIs controlled by everyday objects. For example, it is not possible to build computer vision applications using iStuff or Phidgets. Conversely, because Papier-Mâché only provides input support, it cannot be used to control a servomotor or other physical output device. Second, iStuff offers novel control of existing applications, while Papier-Mâché does not. Unlike iStuff applications, the tangible interfaces Papier-Mâché supports do not use a GUI input model.

2.6.3 Image processing with crayons

Fails and Olsen have implemented a highly successful interaction technique for end-user training of vision recognizers, Image Processing with Crayons [75]. It enables users to draw on training images, selecting image areas (*e.g.*, hands or note-cards) that they would like the vision system to recognize. They employ decision trees as their classification algorithm, using pixel-level features. The resulting recognizers can be serialized for incorporation into standard Java software. Crayons complements our work well, offering a compelling interaction technique for designating objects of interest. Papier-Mâché's recognition methods (*e.g.*, edge detection and perspective correction) are higher-level than the pixel-level processing employed by Crayons. We also offer higher-level object information (*e.g.*, orientation and aspect ratio), and most importantly, an event mechanism for fluidly integrating vision events into applications. Papier-Mâché's classifiers also supports ambiguity [166], an important feature unavailable in Crayons.

2.6.4 Hardware toolkit

Hudson and colleagues [28, 148] have taken a clever approach to supporting TUI prototyping with the hardware toolkit (HWTK); it integrates RFID buttons and other wired and wireless devices with Macromedia Director. This enables the interaction design community, already familiar with Director, to prototype physical devices such as remote controls and game controls. Fluid integration with physical mock-ups is aided by the small form factor of the devices. Like Phidgets and iStuff, HWTK is for mechatronic UIs. Java is a more appropriate language for Papier-Mâché because the target audience is software developers. If we were to support interaction designers, it would be beneficial to follow the HWTK and integrate our work with Director.

2.6.5 Context toolkit

The Context Toolkit (CTK) [64, 65] makes context-aware applications easier to build. Like Papier-Mâché and Phidgets, this work came from a research group that had previously spent several years building applications in the toolkit's domain. The main architectural similarity is that it does not just provide a software interface to physical sensors (*a la* Phidgets); it "separates the acquisition and representation of context from the delivery and reaction to context by a context-aware application" [65, p. 100]. In CTK, a *widget* provides a device-independent interface to an input or output device; similar to Papier-Mâché's *InputSource*. Papier-Mâché provides more monitoring and WOZ facilities than CTK, and it supports interactive tangible interfaces, which CTK does not.

2.6.6 Tools for augmented reality

One use of computer vision in HCI is augmented reality (AR): electronic images are overlaid onto the real world, usually through a head-mounted display or handheld computer [76]. The ARToolKit [126] provides support for building applications that present geo-referenced 3D graphics overlaid on cards marked with a thick black square. The ARToolKit provides support for 3D graphics, while Papier-Mâché does not. However, the ARToolKit does not provide general information about objects in the camera's view, only the 3D location and orientation of marker cards. It is not a general input toolkit; it is tailored for recognizing marker cards and presenting geo-referenced 3D graphics through a head-mounted display.

The DART tool [155] offers designers a Macromedia Director [5] environment for authoring AR content. Like Papier-Mâché, it abstracts technology issues such as sensor input (from 3d trackers or vision-based marker recognition through the ARToolKit). The primary difference between DART and Papier-Mâché is that DART is intended for media design (and media designers); Papier-Mâché is intended for application design

(and software developers). Consequently, the DART abstractions are implemented in C, while designers work in Director. This visual tool enables rapid design to the extent that the provided technology support is sufficient. With Papier-Mâché on the other hand, the toolkit is written in Java and application developers also work in Java. Using the same language for both application and tool development has a higher threshold for application development, but a lower threshold for moving between application development and toolkit extension. The Papier-Mâché visual authoring system (see SECTION 8.8) provides users a low-threshold environment, with a lower wall in between creating applications and extending the tool. The goal of tools in this area should be similar to the goal of web authoring tools such as Macromedia Dreamweaver [7], where (for the most part) users can move fluidly between textual and visual authoring modes.

2.6.7 OOPS: Supporting mediation and ambiguity

The OOPS toolkit [164-166] provides architectural support for *ambiguous* input (input that requires interpretation) and for *mediating* that input (methods for intervening between the recognizer and the application to resolve the ambiguity).

OOPS extends the input-handler in the SubArctic toolkit [112]. Traditional UI software architectures only support input from a keyboard and mouse (or device that can generate equivalent events, such as a stylus). The OOPS architecture can theoretically support any device that can generate discrete events; the OOPS library includes speech [166] and context [65, 216] input. Ambiguity information is maintained in the toolkit through the use of hierarchical events [185]. The hierarchy models the relationship between raw input (such as mouse events), intermediate input (such as strokes), and derived values (such as recognition results). The hierarchy also models ambiguity through hierarchical *ambiguous* events. These events are considered tentative, and may be later recalled. Mediation provides a method for resolving ambiguity. An input event

may be mediated in different ways; two examples are 1) the user can explicitly *choose* one interpretation, or 2) she can *repeat* the input.

Ambiguity is an essential property of recognition-based input support. Papier-Mâché offers a lightweight form of ambiguity to illustrate that the architecture is ambiguity-aware (see section 8.1). A production implementation of Papier-Mâché should more fully support the ambiguity and mediation model introduced in OOPS.

2.6.8 Distributed interaction architectures

There is a rich literature on distributed programming models. Two successful examples of flexible distributed programming for ubiquitous computing are Grimm *et al.*'s one.world [100] and SRI's OAA [169]. Both of these systems employ a data-centric view of applications: computation is carried out on heterogeneous devices performing heterogeneous tasks, and the event model is a blackboard architecture. A blackboard architecture provides a central server that all services and agents post information to and receive information from. The newer one.world system uses XML as its data storage language. The advantage of a blackboard is that it is highly flexible: different services can provide the same data, multiple recipients can read the data, and the posting and reading can be asynchronous. Papier-Mâché could be integrated with a blackboard architecture to support distributed tangible interfaces (see section 7.9 for examples from our interviews).

2.7 Taxonomies of Tangible Interfaces

As the field of tangible interfaces has matured, a few researchers have begun to develop taxonomies useful for describing the behavior and experience of using these systems. In this section, we summarize the contributions of three efforts in this direction, and

discuss the similarities between these efforts and the taxonomy presented in this dissertation.

2.7.1 Emerging frameworks for tangible user interfaces

Ullmer and Ishii provide an excellent taxonomy of existing tangible interfaces [248]. We have drawn heavily on both this taxonomy and the innovative ideas of their Tangible Media Group [115] in creating our list of inspirational applications. They also propose MCRpd [248] as analogue to MVC for physical UIs (see FIGURE 2.4). The difference between MVC and MCRpd is that the view is split into two components: *Rp*, the physical representation, and *Rd*, the digital representation. As a means of describing the user experience of interacting with a TUI this has some merit. However, from an implementation standpoint, it is unclear whether explicitly separating physical and digital outputs is beneficial. In fact, for reasons of application portability, it is important that the event layer be agnostic to whether the implementation is physical or digital (*e.g.*, for prototyping and evaluation purposes, it would be useful to create and compare physical and

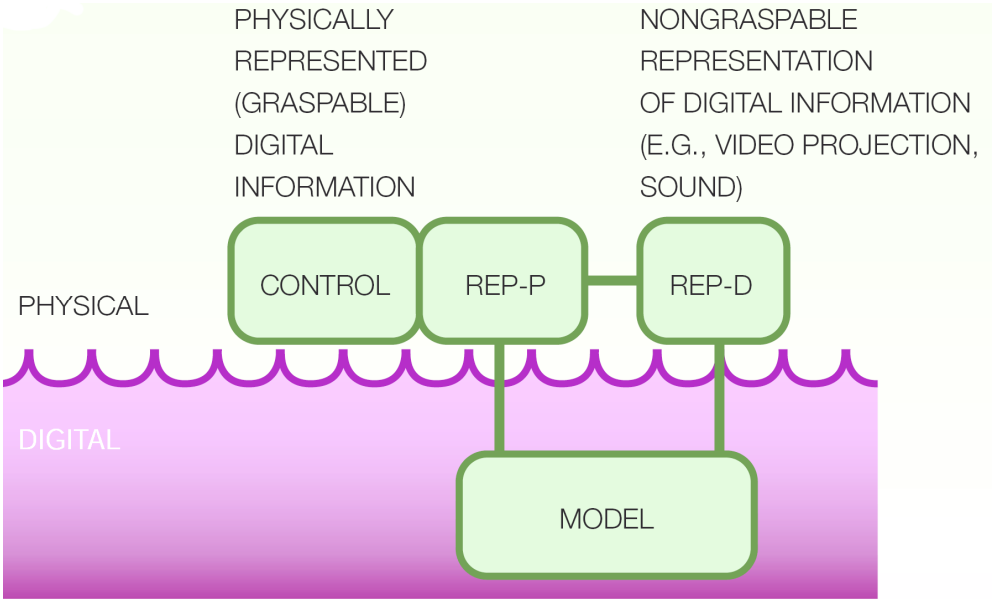


FIGURE 2.4 Ullmer and Ishii's MCRpd model for describing tangible interfaces. Figure from [247, p. 917].

electronic versions of an application). In addition, the approach is untested: no tools or applications have been built explicitly using the MCRpd approach.

2.7.2 A taxonomy for and analysis of tangible interfaces

Fishkin offers two axes, *metaphor* and *embodiment*, as being particularly useful when describing tangible interfaces [77]. Fishkin's embodiment is synonymous with our *I/O coordination* axis (see FIGURE 2.5). Both represent how closely the input focus is tied to the output focus. Fishkin points out how the degree of coordination can be used as a design tool. When the system seeks to maximize the direct manipulation experience, the level of coordination should be high. For example, most of the inspiring spatial applications provide tight coordination (§ 2.8.1). However, when the input and output are cognitively dissimilar, indirect coordination is beneficial in conveying that a mapping is more abstract. For many of the associative applications (§ 2.8.3), the coordination is looser. For example, WebStickers [109], use barcode-tagged Post-it notes as physical hyperlinks to web content; feedback (the desired web page) is displayed on an adjacent monitor. Fishkin's second axis is metaphor, describing the type and strength of analogy between the interface and similar actions in the real world. While this axis is very useful for designers, it is less important for development tools such as Papier-Mâché.

2.7.3 The TAC syntax

Shaer *et al.* present the token and constraint (TAC) syntax for describing tangible interfaces. This work "enables the description of a broad range of TUIs by providing a common set of constructs" [223, P. 359]. The idea of using tokens and constraints for describing TUIs originated in Ullmer's doctoral dissertation [246]. The primary element type in the TAC syntax is a *Pyfo*: a physical object that is part of a TUI. There are two

types of *Pyfos*: *tokens* and *constraints*. Tokens have computational significance; each Pyfo represents either digital information or a computational function. In the marble answering machine [117], the marble is a token. Constraints represent the physical affordances of the system; they limit the behavior of the tokens and do not have computational significance on their own. The physical shape of the indentation in the marble answering machine is a constraint. The other element type is a *variable*. A variable represents a computational function. The message corresponding to a marble is a variable. The tac syntax has some similarity with Papier-Mâché's software architecture (see chapter 8). There are two primary differences. First, tac's constraints are not represented in Papier-Mâché because there is no software code that corresponds to a tac constraint. Second, Papier-Mâché decouples an input object (such as a vision-recognized block) from the behavior associated with the object; this decoupling facilitates switching between input technologies. Tokens comprise both the input object and its computational behavior.

2.8 Inspiring Tangible Interfaces

To better understand the domain of tangible interfaces, we conducted a literature survey of existing systems employing paper and other everyday objects as input. The twenty-four representative applications fall into four broad groups: *spatial*, *topological*, *associative*, and *forms* (see figure 2.5).

We have categorized these interfaces according to four traits: input technology, form factor of the tangible input, form factor of the electronic output, and how tangible input and electronic output are coordinated. In the following sections, we discuss the traits of each of the four groups.

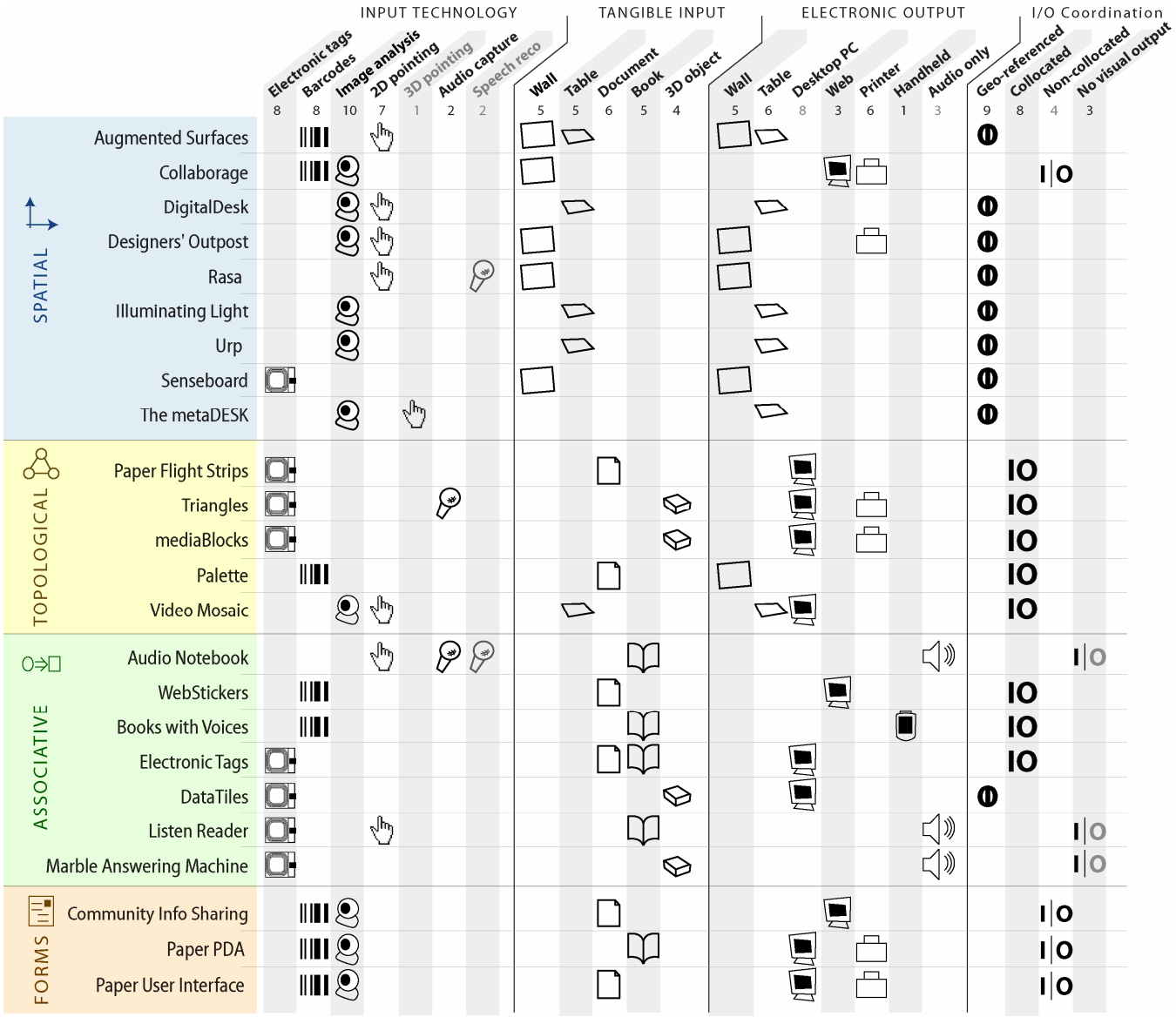


FIGURE 2.5 The rows of this diagram present the 24 applications in our literature survey, organized into four primary categories: spatial, topological, associative, and forms. Each column describes an attribute of the application: this attribute is listed textually at the top of the diagram. In the body of the diagram an icon is used to designate the presence of the column's attribute.

2.8.1 Spatial applications: interactive surfaces

In *spatial* applications, users collaboratively create and interact with information in a Cartesian plane. These applications include augmented walls, whiteboards, and tables [118, 134, 173, 180, 207, 249, 252, 253, 260] (see figure 2.6 and figure 2.7). A majority of these applications use computer vision, often in conjunction with image capture. The work described in this dissertation includes The Designers' Outpost, a spatial application.

Wellner's DigitalDesk used ceiling mounted cameras to track documents and hands on a physical desktop, with a ceiling mounted projector to electronically augment the real desk [260]. The DoubleDigitalDesk [260, 261] extends this augmented paper input paradigm to a pair of networked DigitalDesks. Content can be either physical (drawn on paper by one of the users) or virtual (information that is projected, such as remote content.) The DoubleDigitalDesk enables a user to electronically view and copy her remote colleague's physical content. The Desk does not allow her to move or delete this remote content. DoubleDigitalDesk also allows for spatial content selection, but objects have no semantic distinctive identity. Outpost continues in the direction the Digital-Desk began by augmenting physical paper with electronic information. Each object and awareness cue has a distinct internal representation in Outpost. As such, this information can be edited and displayed separately. Our mediation techniques and stronger semantic representation of content enable users to delete and move remote physical content. Lastly, while the Desk is intended as a pair-ware system, Outpost explicitly supports multiple users at each location.

Ullmer and Ishii's metaDESK [249] is a digital desk employing tangible interfaces as the controls for and views of a map of the MIT campus. Outpost employs the metaDESK

technique of tracking objects with a rear camera; it differs in that it targets an existing professional design practice.

Xerox PARC has developed several influential systems for wall-scale interaction, including the LiveBoard [72, 178] (an electronic interface) and Collaborage (a physical interface) [180]. Collaborage, a spatial application, uses computer vision to capture paper information arranged on an ordinary wall, enabling it to be electronically accessed. (see FIGURE 2.6). These pieces of paper are tagged with glyphs, a type of 2D barcode. The electronic capture of paper information enables remote viewing (*e.g.*, a web page view of a physical in-out board), but not remote interaction.

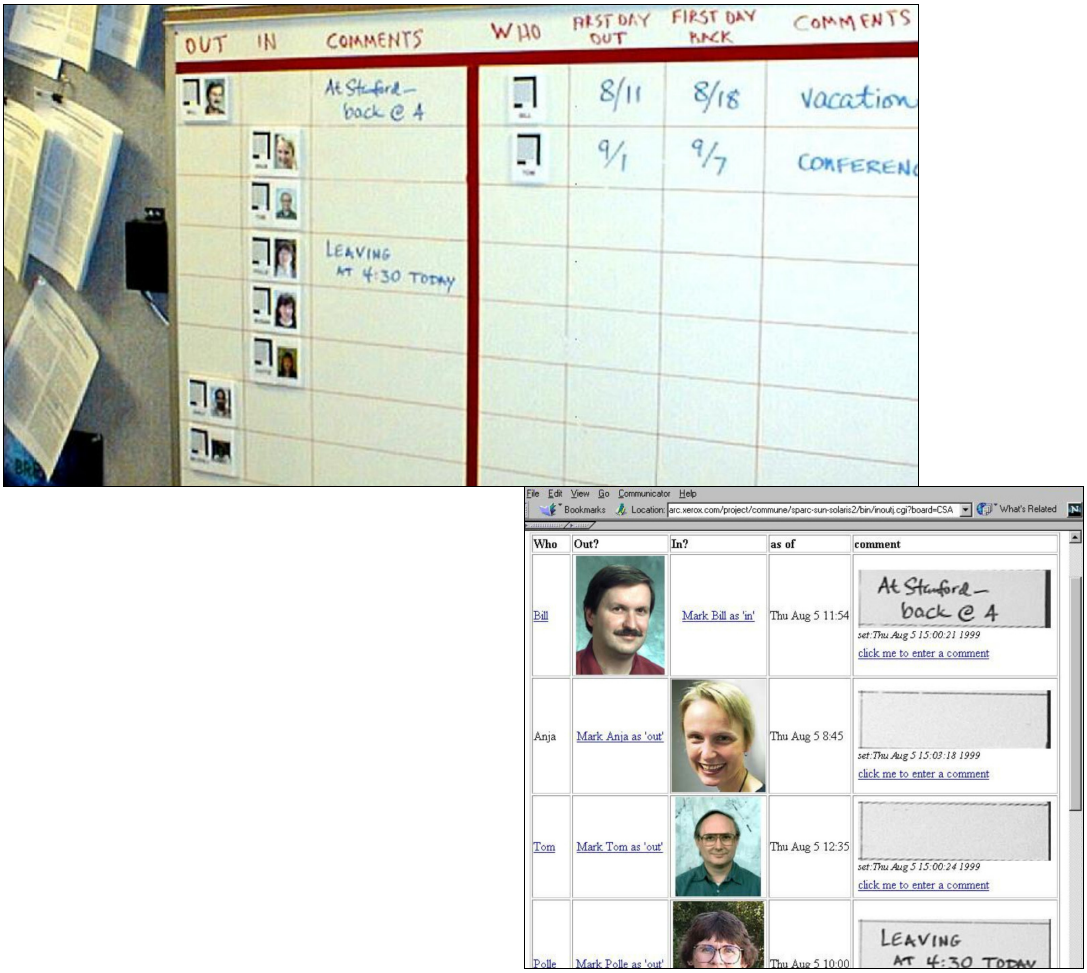
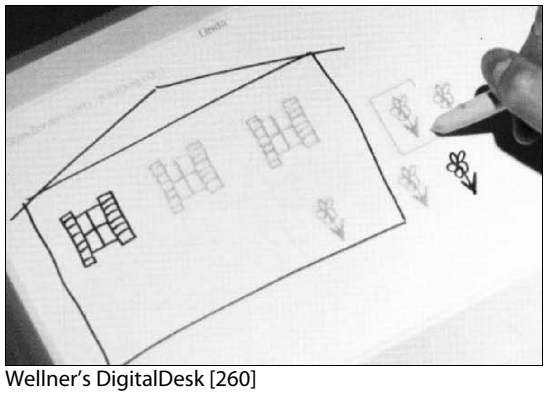


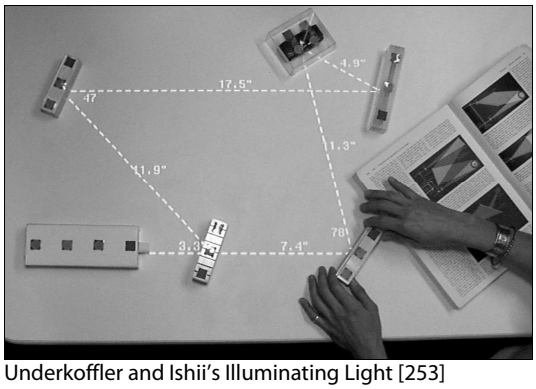
FIGURE 2.6 Collaborage [180], a *spatial* TUI where physical walls such as an in/out board (*left*) can be captured for online display (*right*).


Rekimoto and Saitoh's Augmented Surfaces [207]


Wellner's DigitalDesk [260]

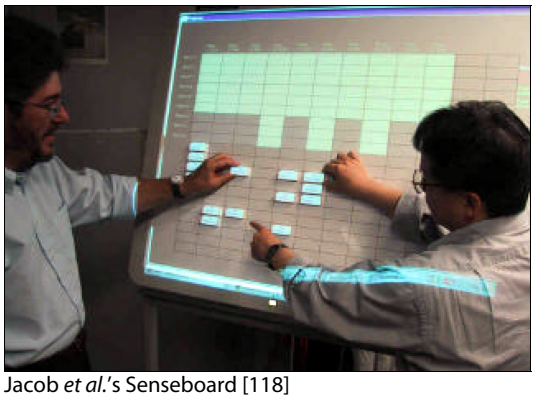

Klemmer *et al.*'s Designers' Outpost (CH. 3, 4, and 5)


McGee *et al.*'s Rasa [173]



Underkoffler and Ishii's Illuminating Light [253]


Underkoffler and Ishii's Urp [252]


Jacob *et al.*'s Senseboard [118]

Ullmer and Ishii's MetaDESK [249]

FIGURE 2.7 The other eight inspiring applications in the spatial category.

Technology Transfer of Spatial TUIs

Several of these spatial TUIs have received interest from industry. David McGee and Phil Cohen have formed a startup, Natural Interaction Systems [20, 57], based on their Rasa research. SMART technologies [168], a Canadian company, has implemented structured capture techniques inspired by the Designers' Outpost in their CamFire product. Their CamFire product is a high-resolution digital camera for whiteboard capture; it based on Eric Saund's ZombieBoard technology [217, 218]. As part of the capture process, CamFire uses computer vision techniques to perspective correct the capture and clean up the image. The addition inspired by Outpost is that the capture is segmented so that Post-it notes and other objects are semantically abstracted and become objects in their Smart Ideas electronic diagramming tool. The POLE project in Switzerland has used this system for urban planning (see FIGURE 2.8), and discussed their use of it in a research paper [42].

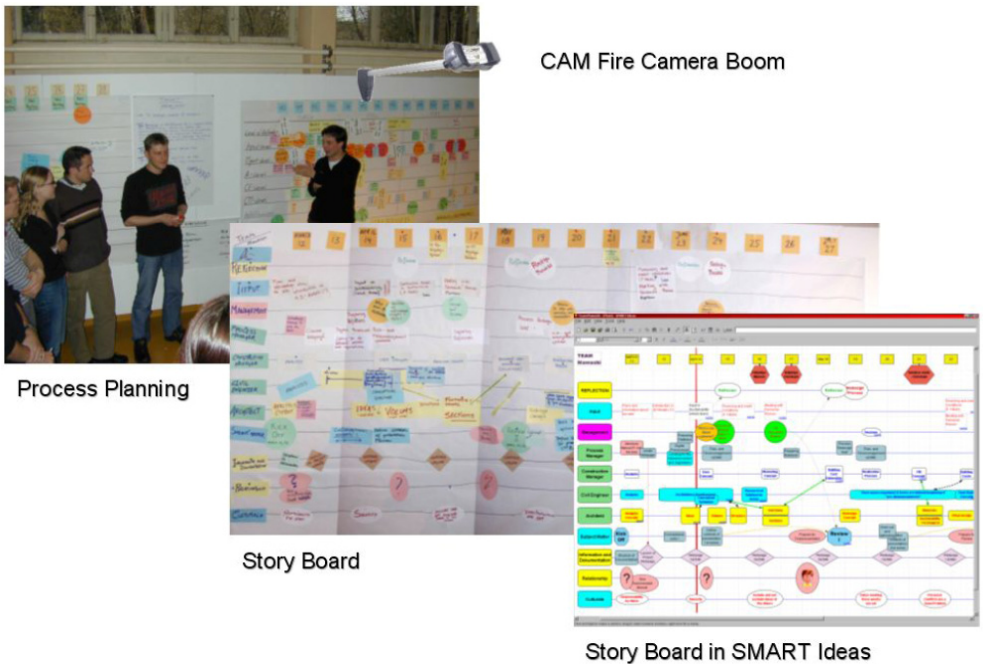
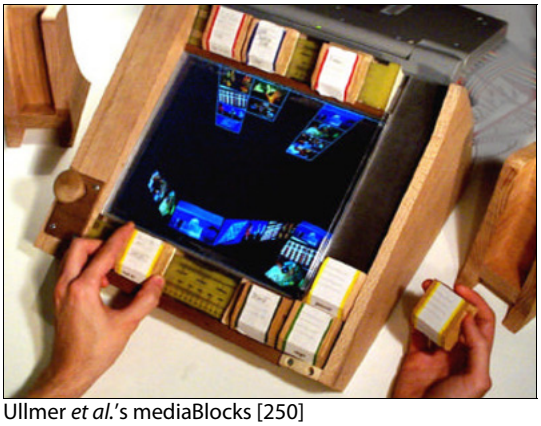


FIGURE 2.8 This SMART technology software, based on The Designers' Outpost research, extracts Post-It notes and links for import into their Smart Ideas software. Image from [42]

2.8.2 Topological applications: relationships between objects

Topological applications use the relationships between physical objects to control application objects [93, 158, 159, 189, 250] (see figure 2.9). The simplest topological relationship, and the one that most of these systems use, is ordering. Palette [189] uses paper note cards to order PowerPoint presentations. It was released as a product in Japan in 2000. VideoMosaic [159] and mediaBlocks [250] use physical objects to order segments of a video. Paper Flight Strips [158] augments flight controllers' current work practice of using paper strips by capturing and displaying information to the controllers as the strips are passed around (see section 2.3.3and figure 2.2).

Gorbet *et al.*'s Triangles [93]



Ullmer *et al.*'s mediaBlocks [250]

Nelson *et al.*'s Palette [189]

Mackay *et al.*'s Video Mosaic [159]

FIGURE 2.9 The other four inspiring applications in the topological category.

2.8.3 Associative applications: physical indices

With *associative* applications, physical objects serve as an index or "physical hyperlink" to digital media [29, 109, 117, 132, 208, 237, 255]. Durrell Bishop prototyped a marble answering machine [117] (see FIGURE 2.10) that would deposit a physical marble with an embedded electronic tag each time a message is left. To play a message, one picks up the marble and drops it into an indentation in the machine. Most associative applications employ either barcodes or electronic tags. Bishop created a partially functioning prototype using resistors embedded in marbles. Marbles could be identified by the unique resistance value. The fact that this system was never fully implemented underscores the need for tools like Papier-Mâché.

The Listen Reader [29] is an associative system combining the look and feel of a real book with an interactive soundtrack. Each RFID-tagged page has a unique soundtrack modified by the user's hand position. Hand tracking is accomplished via capacitive sensing. This coordination of reading and listening is highly compelling.

Lange *et al.'s* Insight Lab system [146] provides barcode-augmented paper on walls in a similar manner to Collaborage. The primary difference between the two systems is that Collaborage actually captures documents' contents and spatial arrangement. With

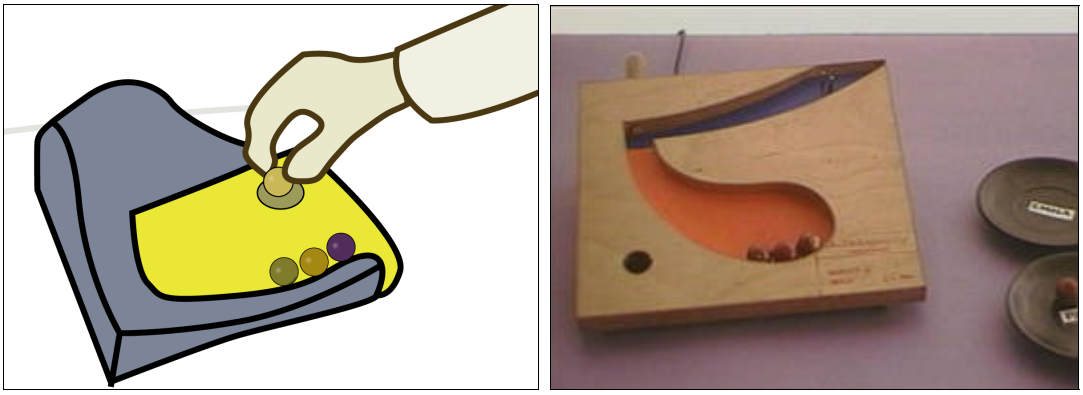
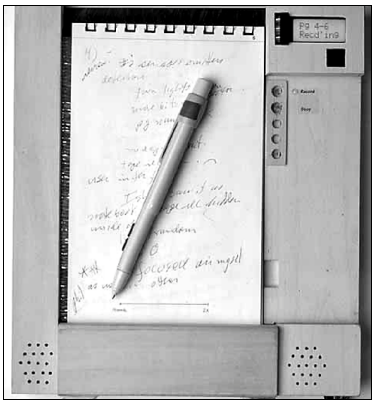


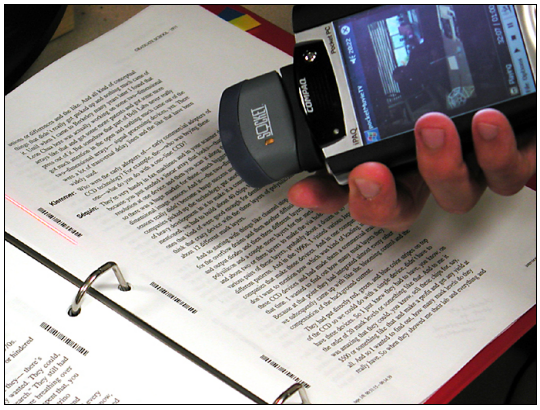
FIGURE 2.10 The marble answering machine [117], an associative TUI, uses marbles as a physical index to recorded answering machine messages. *Left:* Bishop's original sketch, redrawn by the author. *Right:* Bishop's prototype where resistors are embedded in marbles.

Insight Lab, the only computational augmentation is that the barcode can be used as an *associative* link to electronic information; the *spatial* location of documents on walls is not captured.

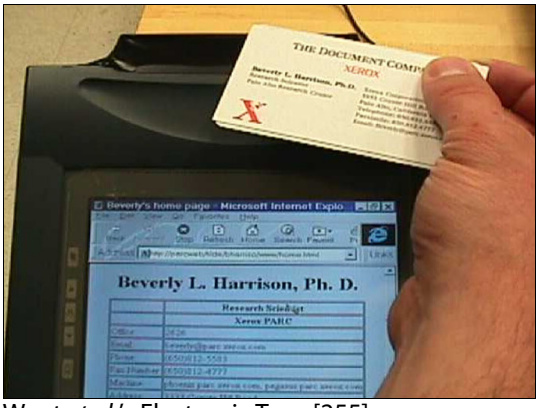


Stifelman *et al.*'s Audio Notebook [237]

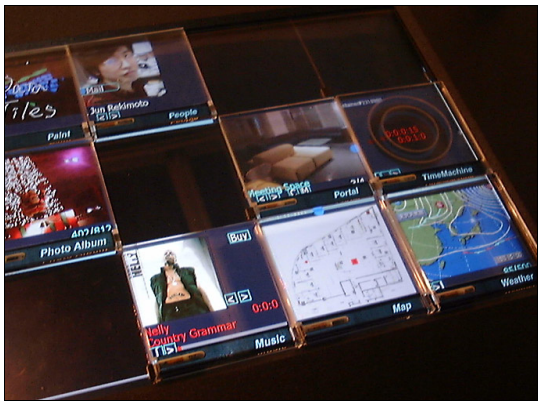
Ljungstrand and Holmquist's WebStickers [109]



Klemmer *et al.*'s Books with Voices (ch. 6)



Want *et al.*'s Electronic Tags [255]



Rekimoto *et al.*'s DataTiles [208]

Back *et al.*'s Listen Reader [29]

FIGURE 2.11 The other six inspiring applications in the associative category.

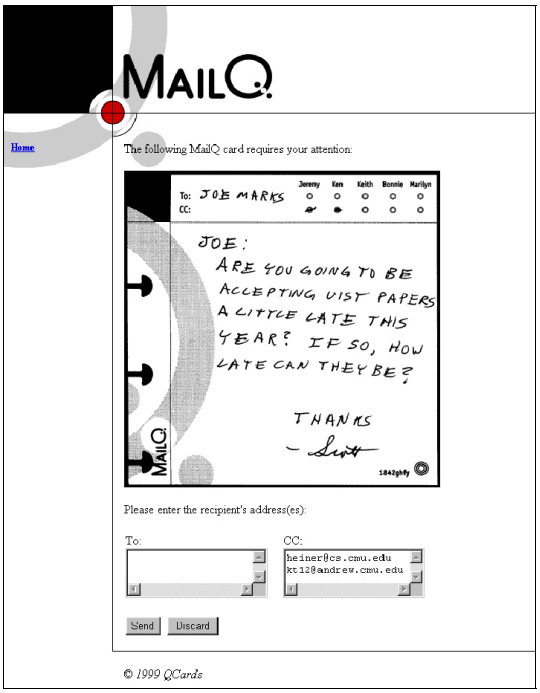
2.8.4 Forms applications: offline interaction

Forms applications provide batch processing of paper interactions [96, 107, 122]. The Paper PDA [107] is a set of paper templates for a day planner. Users work with the planner in a traditional manner, then scan or fax the pages to electronically synchronize handwritten changes with the electronic data. Synchronization also executes actions such as sending handwritten email.

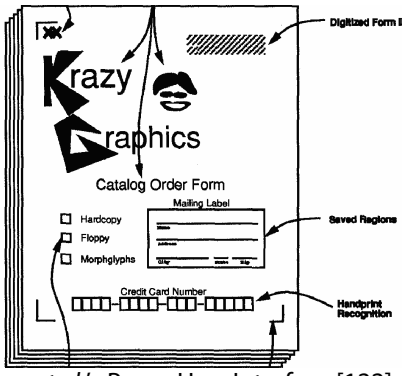
2.8.5 Commonalities

These twenty-four applications share much functionality with each other, including:

- Physical input for arranging electronic content
- Physical input for invoking actions (*e.g.*, media access)



Heiner *et al.*'s Paper PDA [107]



Johnson *et al.*'s Paper User Interface [122]

Grasso *et al.*'s Augmented Newspaper [96]

FIGURE 2.12 The three applications in the forms category.

- Electronic capture of physical structures
- Coordinating physical input and graphical output
- An add, update, remove event structure—these events should contain information about the input (such as size and color), and should be easily extensible

In all of these applications, feedback is either graphical or auditory. Graphical feedback is sometimes geo-referenced (overlaying the physical input, *e.g.*, [134, 173]), sometimes collocated but on a separate display [132, 189], and sometimes non-collocated (*e.g.*, Collaborage's In/Out web page [180]). Because output is generally electronic, we have concentrated our current research efforts on tangible input support. This taxonomy omits haptic (force-feedback) interfaces (*e.g.*, [232]) and mechatronic interfaces (*e.g.*, [78]), which do provide physical output, as these UIs are not the focus of this dissertation.

2.9 Input Technologies for Tangible Interaction

When designing a tangible interface, several different input technologies are available. At least one object needs to be "smart." In augmented paper systems, it can be the pen, the paper, or the workspace. Workspace augmentation best enables informal document use, and informal appropriation of objects in general, as all documents/objects are "compatible" with a system whenever they are in the smart space. For tools that are included with a system, it is generally most appropriate to augment the tool (pen and/or paper) directly. The Anoto system [Anoto] is an example of tool augmentation. In this section, we review the benefits and drawbacks of different instrumentation approaches.

2.9.1 Computer vision as a sensing technique

We have found computer vision to be an appropriate technology for spatial applications because it provides automatic, untethered, and untagged tracking and capture of

artifacts users place on interactive surfaces. We can characterize many sensing systems as being either of the *artificial intelligence* (*AI*) variety, "If it has some properties of a duck, it is a duck," or of the *tag* variety, "If it wears the tag we told ducks to wear, then it is a duck."

Tags can often be more robust, more accurate, and/or computationally cheaper; they are appropriate when the same object will be reused many times with the same system. The SMART Board's pen tray is an example of tagging: each of the four styli rests in a well containing an optical sensor that detects when the pen has been removed. The main drawbacks to tagging are the monetary cost and deployment time proportional to the number of objects. Because of this, there is a barrier to "suiting up" objects for use in the system.

While it can sometimes be less robust, more computationally intensive, and more laborious to develop, *AI* enables informally appropriating members in the class of objects that are being sensed. This is because it is observing some actual properties of these objects, such as their color, size, or shape. This is ideal for objects like the Post-it notes and pictures in Outpost, where free integration of paper artifacts is critical in supporting the flow state of a design session.

2.9.2 Barcodes and glyphs

Barcodes, often the cheapest and easiest method for tagging physical objects, "have been used in packaging since 1974, when the first item, a pack of chewing gum, was scanned at a supermarket in Ohio" [181]. Reappropriating that technology for HCI, Johnson and colleagues introduced the idea of a "paper interface" [122]. Their system prints a 2D barcode (in this case, a glyph) onto paper printouts, such as order forms, enabling users to "complete the document loop" and electronically trigger workflow

actions. This work illustrates the utility of barcodes as a lightweight augmentation mechanism.

Glyphs are a highly flexible 2D barcode. In their most recent incarnation, they can even be embedded into text or images [104]. However, they require a high quality image scanner. CyberCode [206] is a different type of 2D barcode; it is courser than glyphs, enabling recognition by low-cost imaging hardware such as the cameras on mobile devices. High Energy Magic has developed the SpotCode barcode recognition software for camera phones [23]; SpotCodes are a radial barcode format similar in spirit to CyberCode. Cameras and laser-scanners can read barcodes displayed on LCD and plasma screens in addition to working on paper [224].

Using imaging devices for barcode recognition would be ideal for Books with Voices because cameras are smaller, cheaper, and more common than barcode readers. The Papier-Mâché library supports camera-based recognition of CyberCodes and other barcode format (see CHAPTER 8), and could be used to build a future version of Books with Voices.

Recently, there have been several research (*e.g.*, Cooltown [131]) and commercial (*e.g.*, Cue Cat [4]) systems that use barcodes as links to web site URLs. WebStickers [154] are shareable physical handles (Post-it notes with barcodes) to electronic resources. Users can associate a note with a web URL, and later scan the note to retrieve the URL.

Palette [189] provides barcode-tagged index cards to give users control over their presentation's slide order. Users print their presentation onto index cards—one slide per card. An evaluation found the main drawback to be that "People worked on presentations 'until the last minute' and did not 'have time to print cards'" [53, P. 751]. In general, paper interfaces are most successful when the electronic content is stable (*e.g.*, in Books with Voices, see CHAPTER 6), when the electronic and paper versions can evolve independently (*e.g.*, the Listen Reader [29]), or when electronic mediation is

used to handle the discrepancies (*e.g.*, Outpost's design history interface, see CHAPTER 4).

2.10 Evaluating Programming Tools and Languages

Opening the first Empirical Studies of Programming Workshop in 1986, Ben Shneiderman wrote, "Success in programming has been traditionally measured in terms of efficient use of storage and machine resources, accuracy of the numeric results, adherence to specifications, adaptability to change, and portability. These are vital criteria, but questions about the human dimension have become equally important: Is the program readable by other programmers who must test, debug, or maintain it? Is the programming language learnable, convenient for expressing certain algorithms, or comprehensible to novice users? Are design methods, flowcharts, documentation aids, or browsers helpful?" [227].

Almost 20 years later, very little research has been published on evaluating toolkit and programming language APIs *as a user interface*. In this section, we review the research that has been conducted. In designing API evaluation methods, we can draw inspiration from both the software engineering and the empirical studies of programmers communities.

2.10.1 Early evaluation of tools and languages

Alan Kay's Smalltalk [127] language was the first work to explicitly design a language for non-expert programmers (in Smalltalk's case, middle school students) and observe how that community used the language. Kay, a founding member of Xerox PARC in 1970, was awarded the ACM Turing Award in 2003 for this work. Smalltalk was one of the first object-oriented programming languages. From the spring of 1974 through the spring of 1976, Adele Goldberg and Alan Kay taught Smalltalk to Palo Alto children

aged 9 to 15 [91, 92]. At first, they simply collected student programs as samples of what children could do. To learn more about the children's programming process, they began videotaping the programming sessions. To ascertain comprehension, they also videotaped A) the children describing their software after it was complete, and B) the students making requested changes to the software architecture. While the evaluation does not include information on the utility of particular aspects of the Smalltalk language, the student projects are highly compelling.

2.10.2 Empirical studies of programming

Common evaluation metrics in the software engineering community include *performance*, *reliability*, and *lines of code* needed to produce an application. For an excellent review of metric-based evaluation, see Clements, Kazman, and Klein's work [56]. While these metrics are important, they do not address the actual user experience of software development.

In 1985, Alan Newell argued for the need to evaluate programming languages and tools in a more user-centered fashion, writing, "Millions for compilers, but hardly a penny for understanding human programming language use. Now, programming languages are obviously symmetrical, the computer on one side, the programmer on the other. In an appropriate science of computer languages, one would expect that half the effort would be on the computer side, understanding how to translate the languages into executable form, and half on the human side, understanding how to design languages that are easy or productive to use" [190, p. 212].

The year following Newell's article, the first workshop on the empirical studies of programmers was held. John Pane's dissertation offers an excellent review of the contributions of this field [200], as does Françoise Détienne's book *Software Design—Cognitive Aspects* [62]. This community has identified several desirable properties of

programming languages that we believe are also relevant for evaluating a toolkit such as Papier-Mâché:

- *Ease of use:* Programming languages and toolkits should be evaluated on how readable programs using the toolkit are by other programmers, how learnable the toolkit is, how convenient it is for expressing certain algorithms, and how comprehensible it is to novice users [227, p. 1].
- *Facilitating reuse:* A development tool should provide solutions to common sub-problems, and frameworks that are reusable in "similar big problems" [62, ch. 4], minimizing the amount of application code.
- *Schema usage yields similar code:* In our user study, we looked for similarity of code structure—both between programmers and for the same programmer across tasks. This code similarity implies that programmers employ a common schema (design pattern) to generate the solutions. This is desirable because it minimizes design errors, facilitates collaboration, and makes maintaining the code of others easier [62, § 5.2.1]. From this perspective, the success of a toolkit is judged by the extent to which it is leveraged to generate the solution.

2.10.3 Designing usable programming systems

Pane's dissertation [200] effectively argues that usability should be a primary criterion when designing a new programming system. His research demonstrates the application of a traditional user-centered design process to the programming system hands: identifying target users, understanding their needs, and iteratively designing a system and evaluating it with the target users. The primary distinction between our work and Pane's is that Papier-Mâché is designed for software professionals; hands is designed for children with no prior programming experience. This dissertation extends the methodological contributions of Pane's work in two ways. First, we employed fieldwork

with TUI designers to learn their current methods (see CHAPTER 7). Second, in addition to conducting a laboratory evaluation, we also monitored Papier-Mâché's use in longer-term class and research projects (see CHAPTER 9).

2.10.4 Cognitive dimensions of notations

Green's "cognitive dimensions of notations" framework [39, 97] is a set of design guidelines for producing usable programs. These thirteen dimensions (with summaries included verbatim from [97]) are—

- *Abstraction Gradient:* What are the minimum and maximum levels of abstraction? Can fragments be encapsulated?
- *Closeness of mapping:* What 'programming games' need to be learned?
- *Consistency:* When some of the language has been learnt, how much of the rest can be inferred?
- *Diffuseness:* How many symbols or graphic entities are required to express a meaning?
- *Error-proneness*: Does the design of the notation induce 'careless mistakes'?
- *Hard mental operations:* Are there places where the user needs to resort to fingers or penciled annotation to keep track of what is happening?
- *Hidden dependencies:* Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic?
- *Premature commitment:* Do programmers have to make decisions before they have the information they need?
- *Progressive evaluation:* Can a partially complete program be executed to obtain feedback on "How am I doing"?
- *Role-expressiveness:* Can the reader see how each component of a program relates to the whole?

- *Secondary notation:* Can programmers use layout, color, or other cues to convey extra meaning, beyond the 'official' semantics of the language?
- *Viscosity*: How much effort is required to perform a single change?
- *Visibility*: Is every part of the code simultaneously visible (assuming a large enough display), or is it at least possible to juxtapose any two parts side-by-side at will? If the code is dispersed, is it at least possible to know in what order to read it?

Steven Clarke and the Visual Studio usability group at Microsoft have taken these CDN guidelines and adopted them as a usability inspection technique (similar in spirit to heuristic evaluation [193-195]) and as a language for describing laboratory evaluation results of the Visual Studio .NET IDE, the C# language, and the .NET APIs [54, 55, 215]. These heuristics offer a nice complement to the higher-level design goals in SECTION 2.10.2.

Clarke *et al.'s* work is the first publication by a product group taking a user-centered approach to designing programming languages and tools. Their work begins by using scenarios of small programming tasks to drive design. Then, the relevant IDE, library, or language feature is designed. After design, the feature is evaluated in a usability test. Not every element in the system undergoes this rigorous treatment; only aspects that Microsoft feels are particularly important. In June of 2004, we visited this product group at Microsoft to see their usability labs and discuss their methods. Each feature is evaluated with approximately ten developers. Each developer participates in two sessions that are roughly two hours long; the two sessions are about a week apart. The first session is used to ascertain how usable a language/IDE feature is the first time a developer sees it. The second session ascertains how easily the developer retains that knowledge, and how use changes with familiarity. This usability evaluation work is similar to our laboratory evaluation of Papier-Mâché, with the difference that

Microsoft's larger budget affords more rigorous application of the technique. The primary methodological contribution that this dissertation offers beyond Microsoft's work is our method of "triangulating" API usability by aggregating the results of multiple methods: fieldwork, controlled laboratory studies, and monitoring longer-term developer use of the API. This triangulation is especially important when evaluating tools in emerging areas such as ubiquitous computing, where best practices are unclear and the range of ideas is much wider.

2.10.5 Evaluating ubiquitous computing tools

Evaluating the usability of software tools in emerging areas such as ubiquitous computing poses a much greater challenge than evaluating tools for more mature areas such as graphical user interfaces because in emerging areas the applications and the tools are evolving simultaneously. Additionally, the amount of time required to build these novel tools dissuades many researchers from the additional step of evaluating the tools' usability. This section reviews prior work that has "gone the extra mile" to evaluate the usability of ubicomp tools.

The Context Toolkit (summarized in SECTION 2.6.5) was evaluated through the applications that the researchers and their colleagues built with the system. These applications demonstrated that the toolkit supported a wide range of applications, and that these applications could be implemented with less time, code, and expertise than was previously required.

Edwards *et al.* offer observations on evaluating infrastructure in [69]. This work draws on the authors experiences with the Placeless documents system and the Context Toolkit. The authors raise the issue that when infrastructure offers new technical capabilities, it is difficult to evaluate the usability of the infrastructure support for these new capabilities from the usability of the infrastructure-enabled applications that

actually deliver these capabilities to the end user. They advocate that infrastructure features and novel applications be designed iteratively and in conjunction with each other. Papier-Mâché differs from these systems in that it primarily *lowers the threshold* for application development, while these novel services primarily *raise the ceiling*. (For a discussion of ceiling and threshold, see [188]). Building applications as an evaluation technique is one aspect (see SECTION 9.6) of our triangulation methodology. Our methodology also includes the application of other techniques, such as laboratory evaluation (see SECTION 9.4) and use in research projects (see SECTION 9.5), to gain a fuller picture of the software's usability.

3 The Designers' Outpost

> *"Our experience strongly supports Alexander's hypothesis that good fit cannot de directly defined or designed; it is the absence of misfit, achieved by iterative design. … In a word, the computer scientist is a toolsmith—no more, but no less. It is an honorable calling."* —Frederick P. Brooks, 1977 [44].

Studies of web design practice have found that pens, paper, walls, and tables were often used for explaining, developing, and communicating ideas during the early phases of design. These wall-scale, paper-based design practices inspired The Designers' Outpost, a tangible user interface that combines the affordances of paper and large physical workspaces with the advantages of electronic media to support information design. With Outpost, users collaboratively author web site information architectures on an electronic whiteboard using physical media (Post-it notes and images), structuring and annotating that information with electronic pens. This interaction is enabled by a touch-sensitive SMART Board augmented with a robust computer vision system, employing a rear-mounted video camera for capturing movement and a front-mounted high-resolution camera for capturing ink. We conducted a participatory design study with fifteen professional web designers. Our results show that Outpost supports information architecture work practice, leading to the addition of fluid transitions to other design tools.

Portions of this chapter were originally published by the author, Mark W. Newman, Ryan Farrell, Mark Bilezikjian, and James A. Landay in [134]

3.1 Introduction

Newman and Landay's studies into web design [191, 192] found that pens, whiteboards, paper, walls, and tables were the primary tools used for explaining, developing, and communicating ideas during the early phases of design. Later phase design, where detailed page mockups are generated, occurs mostly on the computer. This finding is consistent with work practice studies across many design and engineering domains [34, 114, 262].

In one common early-phase practice, designers collect ideas about web site content onto Post-it notes and then arrange them on the wall into categories. This technique, often called affinity diagramming [37], is a form of collaborative "sketching" used to determine the site structure (see FIGURE 3.1). We have built a tool, The Designers' Outpost, that supports this practice. It combines the advantages of both paper and electronic media. A video of the Outpost system is available on the web at http://guir.berkeley.edu/outpost.

FIGURE 3.1 A designer sitting in front of a Post-it Note covered wall. Post-it notes represent chunks of information and are arranged spatially into groups of related information. These notes are linked with marker lines to show organizational relationships. Image courtesy Hugh Beyer and Karen Holtzblatt [37].

3.1.1 Current physical practice: benefits and drawbacks

The large workspace of a wall or whiteboard offers several clear benefits for collaborative design tasks. Large workspaces permit the representation of large, complex information spaces without the loss of contextual, peripheral information (see FIGURE 3.2). In contrast with the heavyweight, formal operations of the computer, it is relatively easy to fill a wall with pieces of paper and move them around to suggest different associations. Paper and walls "make information, any kind of information, tangible, easy to manipulate, and easy to organize" [209, p. 4]. Collaboration is aided both by the persistence of the design artifact, which supports asynchronous collaboration and constant awareness of the state of the project, as well as by the greater-than-human-sized space allowing multiple people to simultaneously view, discuss, and modify the artifact. Covi *et al.* refer to the work posted on walls in project rooms as "coordination documents" [58, p. 59] because of the important role these highly visible artifacts play in collaboration.

FIGURE 3.2 One of two design rooms at a Silicon Valley web site design firm visited by the author.

There are drawbacks, however, to the traditional paper-centric representation. When using paper and walls for design, it is difficult to maintain structuring marks (such as links and annotations), create multiple versions, or collaborate with designers at another location. Much of the information exists in the *relationships* between information chunks (Post-it notes). Because structure must be maintained manually, marks that the designers make about the data, such as links or annotations, often fall out of sync with the notes as they are shifted around. At some point, whether hours after a brainstorming session or months after a project, the paper is removed and the web site structure is lost. The designers in our studies also lamented that versioning is not feasible in a paper-only representation. The paper-only work practice also offers few opportunities for remote participants, whether at a desktop down the hall or in a meeting room across the world. Remote users have no way to update, or even access, the information. We also found, as others have, that the transition from the early

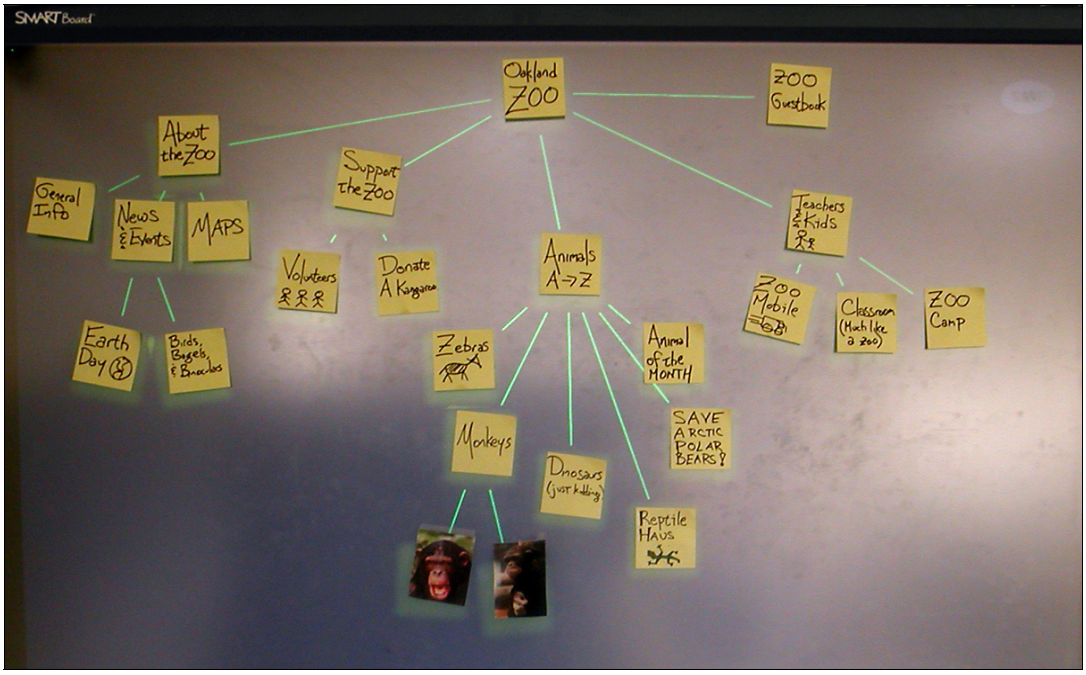


FIGURE 3.3 A web site information architecture using a combination of physical Post-it notes, physical pictures, and virtual links showing relationships between them.

paper-centric design stages to the later pixel-centric stages is highly problematic [145, 260]. As the site structure is changed during development, the early paper artifact drifts further and further out of date.

3.1.2 Supporting and extending practice with Outpost

The Designers' Outpost (figure 3.3) is a tangible user interface that combines the affordances of paper and large physical workspaces with the advantages of electronic media to support information design for the web. Users have the same fundamental capabilities in the Outpost system as in a non-computational paper-based system; one can create new pages by writing on new Post-it notes and organize a site by physically moving Post-it notes around on the board. Thus, paper in the physical world becomes an input device for the electronic world. A rear-mounted projector outputs electronic information onto surfaces in the physical world. Through its electronic capture of designs, our system supports the transition from this early representation to later electronic tools, such as denim [153, 192] (see figure 3.4 and figure 3.5).

FIGURE 3.4 DENIM, shown here in sitemap view, allows web site design by sketching. As seen here, physical information spaces created in Outpost can be electronically imported into DENIM, serving as baseline sitemaps.

Outpost is part of the UC Berkeley Group for User Interface Research's work on informal user interfaces [145]. 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. In addition to Outpost, this group has developed informal design tools for graphical [142, 144, 145], web [153, 192], speech [135, 229], multi-modal [230], and cross-device [151, 152] user interfaces.

The rest of this chapter is organized as follows. First, we discuss work in early stage web design and wall-scale user interfaces. Section 3.3 describes the formative design studies that we carried out to inform and validate our approach. Section 3.4 introduces the interaction techniques that give the core functionality of our system. Following this in section 3.5 is a description of the evaluation of the interactive system, and section 3.6 discuses implications of that study. Section 3.7 describes the prototypes of the underlying vision algorithms, and section 3.8 offers a detailed description of the final

FIGURE 3.5 In DENIM's storyboard view, designers can continue working with an Outpost sitemap by sketching out the contents of a page.

vision system in Outpost. Chapters 4 and 5 describe support for design history and remote collaboration with Outpost, respectively.

3.2 Background

Our research is inspired by fieldwork into web design practice and the writings of designers reflecting on and prescribing effective design methods. We describe each of these in turn.

3.2.1 Web site design: tools and practice

The goal of Newman and Landay's investigation into web design [191, 192] was to inform the design of systems to better support actual practice. The study comprised interviews with eleven professional web site designers from five different companies. Each interview consisted of asking the designer to choose a recent project and walk the interviewer through the entire project. The designer was asked to show examples of artifacts produced during each phase and explain their role in the process.

Three important observations were made during the course of this study. First, designers create many different intermediate representations of a web site. Examples of pervasive and significant intermediate artifacts include sitemaps, storyboards, page schematics, and mockups. These representations depict the web site at varying levels of detail, from sitemaps, which depict sites as related blocks of labeled information to mockups, which depict individual pages in high fidelity. Second, the production and use of these intermediate artifacts dominate the day-to-day work practice for most of the design process. Third, web design is comprised of several sub-specialties, including information architecture and visual design, each of which has its own tools, products, and concerns. While visual designers typically focus on interaction and graphic design, information architects are mainly concerned with the information and navigation

design of a web site. Newman and Landay found that information architecture is not well supported by current software tools: for example, sitemaps were regularly generated by placing Post-it notes on walls.

The results of these studies provided motivation for Outpost, and also provided the impetus for the development of DENIM, a sketch-based tool supporting information and navigation design of web sites [153, 192]. DENIM supports sketching input, allows design at different refinement levels, and unifies the levels through zooming. In particular, DENIM supports visualizations matching the *sitemap*, *storyboard*, and *page schematic* representations of a web site. DENIM also allows designers to interact with their site designs through a *run mode*, which displays the sketched pages in a limited functionality *browser* that allows the user to navigate the site by clicking active regions of the sketches and linking to other pages within the site. While DENIM supports authoring sitemaps, it is best suited for storyboards and page schematics. Outpost targets sitemaps. In addition, DENIM was designed as a single-user interface, whereas Outpost was designed for collaborative work.

3.2.2 Affinity diagrams

Our interest in researching computational support for information architecture was motivated by one specific design practice observed during the study discussed above. This collaborative practice consists of arranging Post-it notes on a large surface such as a wall, table, or desk in order to explore the information structure of a web site. Designers write chunks of information on Post-it notes and stick them to the wall. They then move the notes into spatially proximate groups representing categories of related information. Groups are labeled and further grouped into hierarchies of groups. This hierarchical structure serves as a baseline for the structure of the web site. Lines are drawn between notes and groups to indicate links. Early on, these links indicate

organizational relationships (i.e., it should be possible to navigate between the endpoints.) Later in the design process, these links represent hyperlinks that will appear in the finished site. This technique is called affinity diagramming; it is described in depth in Beyer and Holtzblatt's book, *Contextual Design* [37]. Usability expert Jakob Nielsen also advocates a version of this method, using index cards to design the information hierarchy [196].

3.2.3 Electronic walls

One of the first wall-scale user interfaces was the Xerox PARC LiveBoard [72]. This system demonstrated pen-based interaction techniques like those of a traditional whiteboard. Pens are an effective method for fast, fluid, informal input. Xerox PARC later developed the Collaborage system. Collaborage's computer vision capture of paper on walls [180] inspired our work on the Designers' Outpost. One drawback of the Collaborage capture system is its long latency (8-10 seconds on average). Outpost improves on this, with a location recognition latency of ~200 milliseconds, and an

FIGURE 3.6 Stanford's PostBrainstorm system offers a high-resolution, interactive wall [101].

average capture latency of ~1.5 seconds. This improvement is primarily due to our two-camera hardware approach, and software built on top of OpenCV, a highly optimized vision toolkit [40]. Outpost also incorporates other forms of input using styli and physical tools.

In addition to the wall-scale tangible interfaces described in SECTION 2.8.1, several researchers have developed electronic wall-scale interfaces. Streitz's group at GMD developed i-LAND, a system of interaction techniques for working with display surfaces embedded in rooms and furniture [238]. As part of their work, they developed the DynaWall: three adjacent electronic whiteboards that take input via hand gestures. Winograd and Guimbretiere have developed the PostBrainstorm system, which provides interaction techniques for large, tiled projector surfaces [101] (see FIGURE 3.6). Rekimoto and Saitoh [207] developed a system to integrate laptop computers, projected surfaces, and tagged physical objects. Other researchers have investigated interaction techniques for large electronic display surfaces and multimodal interaction with paper [173, 174]. This body of work on wall-scale interfaces motivates the concept that, for many tasks (especially in collaborative situations), manipulating physical objects on large surfaces is an intuitive and effective means of performing computer input.

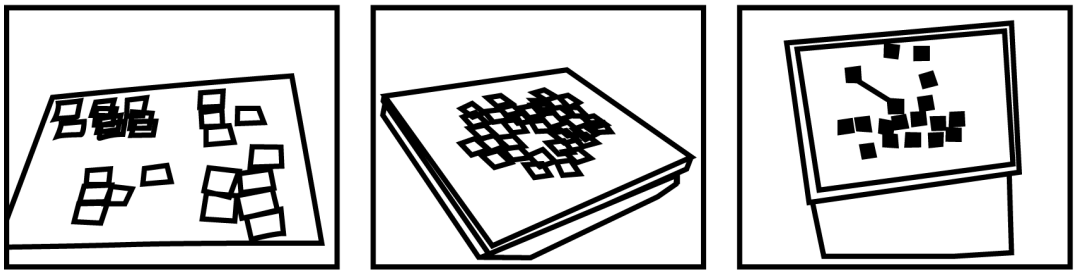
3.3 Initial Design Studies

We began the Outpost research with a series of participatory design studies that explored the combination of physical and electronic media in depth. These three studies employed both low- and high-fidelity prototypes (see FIGURE 3.7). The design teams in our studies encouraged us to support freeform-ink, electronic annotations to sitemap pages, versioning of design artifacts, fluid transitions to other tools, and opportunities for collocated and remote collaboration. In parallel, we built a set of prototypes for the underlying vision system (see FIGURE 3.8). These prototypes led us to difference image-

based recognition algorithms and a two-camera infrastructure: a rear-mounted video camera for capturing movement, and a front-mounted high-resolution camera for capturing ink.

The first design study validated the general approach. It indicated a need to minimize the extra user effort required to use the tool, and encouraged us to allow the interaction to be as freeform as possible. The second prototype fleshed out the interaction techniques and showed that a drafting desk is too small to support professional-scale web site diagrams. While our primary early interest in Outpost was to provide interactive support for design meetings, designers in the third study, working with an interactive wall-based prototype, found constant interactive feedback distracting. They encouraged us to refocus our interface on: 1) supporting free ink electronic annotations to sitemap pages, 2) fluid transitions to tools such as DENIM [153, 192], 3) versioning of design artifacts (see CHAPTER 4), and 4) supporting collocated and remote collaboration (see CHAPTER 5). We also found this system to be more appropriate for information architects than for visual interface designers.

Our research on information architecture tools began by creating prototypes that enabled user feedback and by designing a computer vision back-end for capture of paper artifacts. In the following sub-sections, we discuss the Outpost interface and design



First: Paper desk prototype *Second:* Paper and pixel desk *Third:* Interactive Wall

FIGURE 3.7 The sequence of prototypes used in the three design studies.

studies in detail. In later sections, we describe the vision back-end.

To explore the viability of combining physical and electronic representations for web site information architecture, we undertook a series of three design studies. We first evaluated the basic concept with a paper prototype study. Next, we built interface mock-ups that envisioned the combination of physical artifact state with interactive feedback. Finally, we created a wall-scale prototype for a set of participatory design sessions with fifteen professional interface designers.

3.3.1 Low-fidelity desk: design study

We created our initial low-fidelity prototype using cardboard the size of an ITI VisionMaker Digital Desk (41" diagonal), evaluating this paper prototype with two individual participants. The participants wrote on a pad of 3" × 3" yellow Post-it notes using an inking pen (see FIGURE 3.9). We asked participants to create the information architecture for a web site about off-campus housing for college students. To start, we handed them six pages of notes from mock interviews with college students seeking housing. The task included chunking interview information onto Post-its, arranging the Post-its into related groups, and merging two previously saved versions of Post-its into

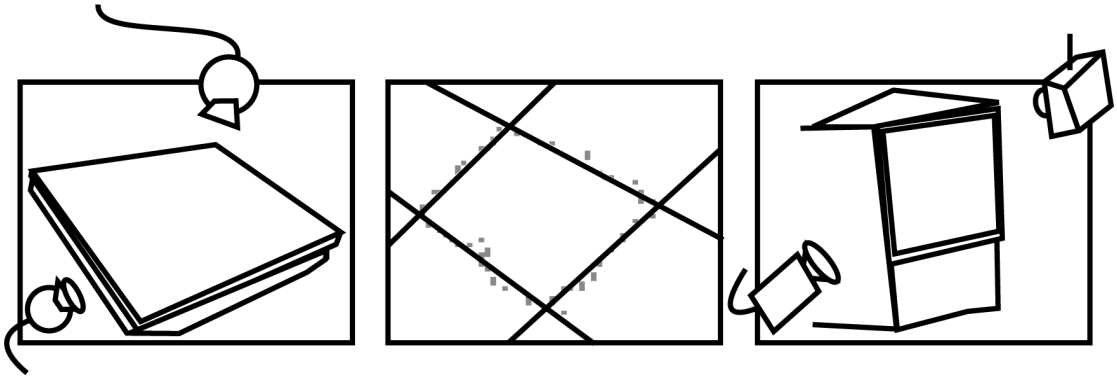


FIGURE 3.8 The sequence of computer vision studies. The first prototype (*left*) explored the difference image algorithm using the Java Media Framework and webcams. The second prototype (*center*) explored the expectation-maximization algorithm for line-fitting using Matlab. The final prototype (*right*) integrated these techniques into a functioning system with a user interface.

a unified version. A wizard acting as the computer [128, 172] gave verbal feedback about what the computer recognized as groups, which groupings were being selected, and displayed widgets and dialog boxes when appropriate.

We found that participants often forgot required system steps that had no affordance or feedback, such as underlining the note representing a group or pressing an upload button to add a note to the system. This suggested an interface where we automatically recognize as many actions as possible (*e.g.*, a new note should be automatically added when it is placed on the desk). Users were also confused by the three input devices: the inking pen for writing on notes, the virtual stylus for authoring note relationships, and the keyboard for entering version names. As a result, we removed the keyboard from our system design: this simultaneously simplifies the input model and better matches current practice.

3.3.2 Pixel and paper mock-up

Using our findings from the paper prototype, we created a mock-up of our ideas for combined physical/virtual interaction. We created static images using Adobe Photoshop to prototype Outpost's interaction techniques and visual presentation. These

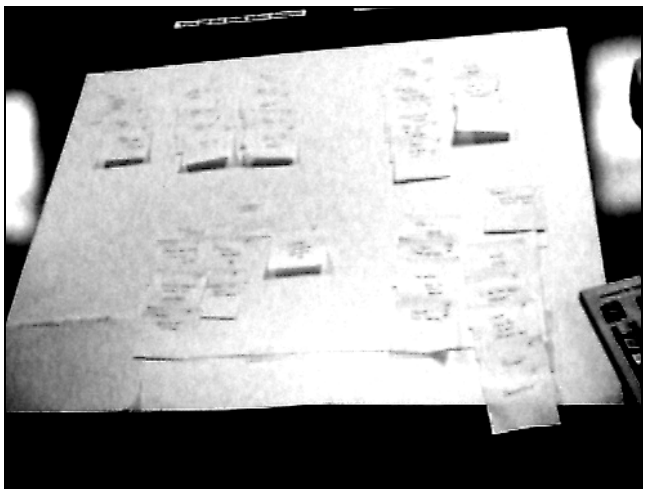


FIGURE 3.9 The low-fidelity Designers' Outpost.

images were displayed on an iti VisionMaker Digital Desk, which is a rear-projected surface with the size and slope of an architect's drafting table. We designed the initial set of four interaction techniques to aid the affinity diagramming process. 1) Designers could create *groups* by placing notes near each other. 2) *Links* could be drawn between groups using an electronic stylus. 3) Groups could be given a name with a *label*. 4) Groups could be organized into *hierarchies*. The interface mock-up showed physical Post-it notes and the corresponding electronic feedback for these initial interactions (see figure 3.10). Through our experience with this mock-up, it quickly became evident that a digital desk is too small a space for professional web site information architecture. The desk's surface area affords for a maximum of fifty Post-its and two or three users. Information architects often use upwards of 200 Post-its, and four to eight people may participate simultaneously in design sessions. To build the Designers' Outpost at a full collaborative scale, we moved our design to a smart Board, a much larger rear-projected surface with a whiteboard form factor [21].

3.4 Outpost Interaction Techniques

Our low-fidelity and mock-up prototypes informed the design of our interactive prototype. The physical, direct manipulation interaction techniques in Outpost provide

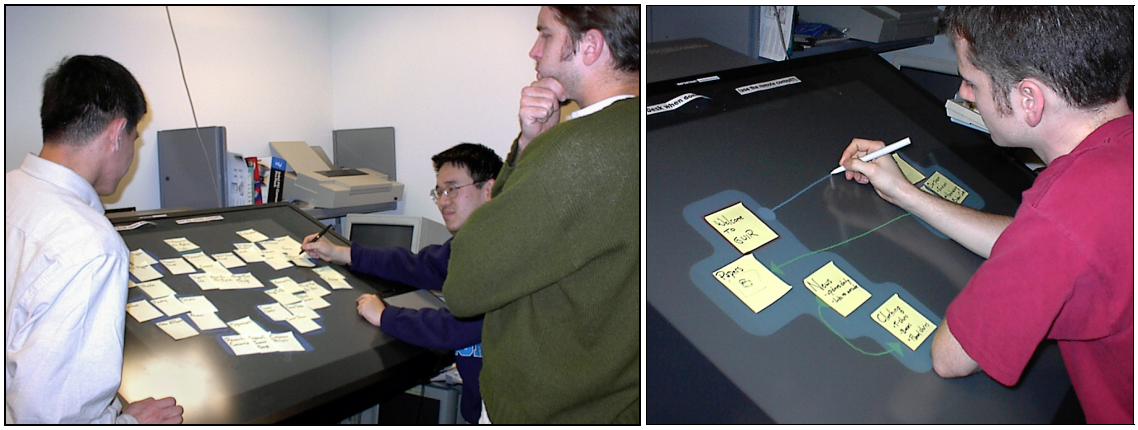
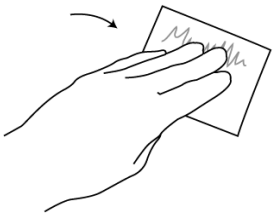


FIGURE 3.10 Mock-ups of the Designers' Outpost—Collaborating on an information hierarchy with Post-its on a digital desk.

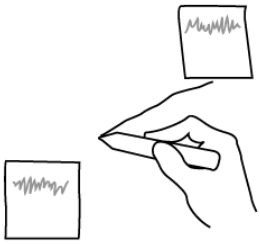
for authoring content with standard pens on Post-it notes. The system tracks notes as users physically add, remove, and move them around the board, but does not attempt to recognize the content of the notes.

Outpost supports the following interaction techniques for working with paper on the board. We have combined these physical interactions with interactions that are better suited to an electronic medium, such as digital ink annotation and specifying relationships using virtual arrows.

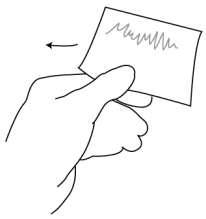
Adding Notes: Users can write on a note with a standard pen and add it to the board. Our vision system recognizes it and updates its internal understanding of the board. The system provides feedback that the note has been recognized by displaying a blue outline around the note.



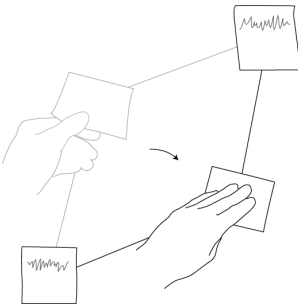
Creating Links: To link a pair of notes, the user draws a line from one note to another with the board stylus.



Removing Notes: To delete a note and its associated links, the user pulls the note off the board.

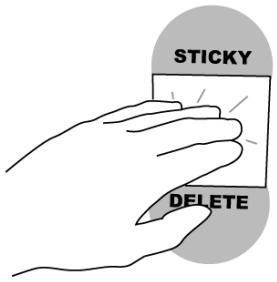


Moving a Note: To move a note, its links, and its annotations, the user picks it up and places it at a new location. This provides a lightweight means of coupling physical and electronic information. A simple timeout heuristic determines movement. If note removal and replacement occur within a specified time (Outpost



uses seven seconds), the system interprets this as a move operation. This heuristic has been sufficient for evaluating the prototype. A production implementation of Outpost should compare the image of the placed note to that of previously removed notes. This image similarity comparison is tractable using current computer vision techniques [129]. Shape Contexts offer the most promise for notes with ink content [35]; pyramid-based techniques may be superior for general images [83] (§ 7.7).

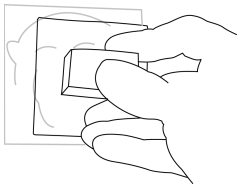
Context Menus: Tapping a note invokes an electronic context menu, enabling the manipulation of the electronic properties embodied by physical objects. *Sticky* replaces a physical note with an electronic image of the note. *Delete* removes a note (useful if the vision system misses a physical removal).



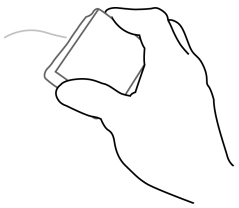
With the feedback from our participatory design study (see section 3.5), we added the following three physical tools for manipulating electronic content.

Freeform Ink: In addition to being a space for interacting with physical Post-it notes, Outpost supports freeform drawing using board styli.

Move Tool: A physical move tool provides a means of interacting with the system after the physical content has become electronic, retaining haptic direct manipulation.



Physical Eraser: Working like a normal whiteboard eraser, the Outpost eraser removes ink on the board. It operates semantically, deleting each stroke it passes over.



Two primary benefits to structured capture of informal artifacts are 1) later recall, and 2) export to other tools; saving enables both of these.

Saving the Board: Users can press save to save the board state to disk. Then they can open it later in DENIM or Outpost.

One important part of the Outpost visual design is that the board's background is black. Because the board only emits light where the user has authored content, it avoids being a giant glowing presence, thus affording a calmer interaction experience [258].

3.5 Professional Design Study

Our interactive prototype for the professional design study was implemented as a Java application running on a rear-projected 72" diagonal touch-sensitive SMART Board [21] with a 1280 × 1024 resolution LCD projector. With this prototype, we recognized the location of notes on the board using the board's touch sensor. Drawing a line from one note to another with the board stylus creates a link. The stylus is also used for creating freehand electronic ink on the board (see FIGURE 3.11). Tapping on a note invokes a context menu (see FIGURE 3.12) that in this prototype lets users either delete the note or define it as the label note for its group. In the vision-backed Outpost system described

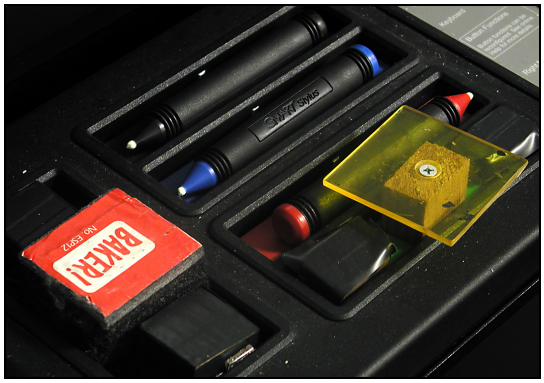


FIGURE 3.11 The board's tool tray: styli for drawing electronic ink, a clear plastic square for moving electronic content, and the eraser. *(Only the pens were available during the design study.)*

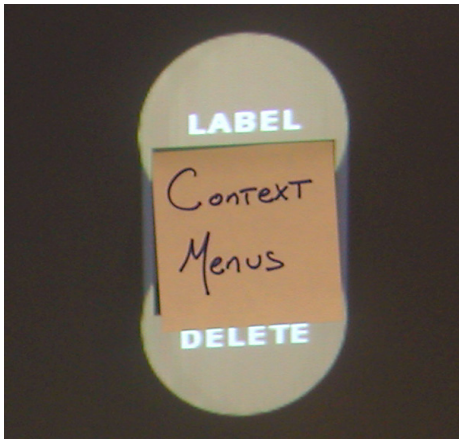


FIGURE 3.12 Tapping on a note invokes an electronic context menu for physical content.

Role	INFORMATION ARCHITECTS		BOTH	VISUAL DESIGNERS	
Group	A	B	C	D	E
Size	5	3	2	2	3

TABLE 3.1 The five study groups: their size and primary role.

later, removing a note from the board deletes it. (The delete menu item was retained for the rare instance when the system fails to detect a remove event.)

3.5.1 Study design

Fifteen professional web designers participated in the study. There were five design sessions with between two and five designers per session. In four of the design sessions, the designers were colleagues at the same company; the fifth session mixed designers from two companies. Two of the five groups were composed of information architects, two groups were visual designers, and one group had individuals performing both roles (see TABLE 3.1).

Two researchers conducted each session. One was in charge of communication, explaining the system, and facilitating discussions. The other took written notes and videotaped the session; FIGURE 3.13 – FIGURE 3.15, are stills from those recordings.

Each session lasted roughly two hours (see TABLE 3.2). We began the sessions with a high-level overview of the project and a brief demonstration of the existing prototype. We gave each team an information architecture design task to explore using the

Overview and Demo	15 minutes
Design Task	45 – 60 minutes
DENIM Demo	15 minutes
Discussion	30 – 45 minutes
Survey	10 minutes

TABLE 3.2 The time breakdown of the design sessions.

prototype; this task took 45 to 60 minutes. We conversed freely with the designers during the sessions. Throughout each session, participants explained their actions and gave feedback and suggestions for improvement. This was followed by a fifteen-minute demonstration of DENIM and then a 45-minute discussion on Outpost's utility and its relationship with DENIM and their current work practices. The study concluded with a seventeen question written survey asking participants about their background and their opinions about Outpost and its relevance to their work (see APPENDIX A).

3.5.2 Design findings

Our findings from this participatory design study offered insight into the designers' collaborative work processes and suggested an appropriate interactivity model.

Existing work processes

Every participant currently works with groups on whiteboards early in the web site

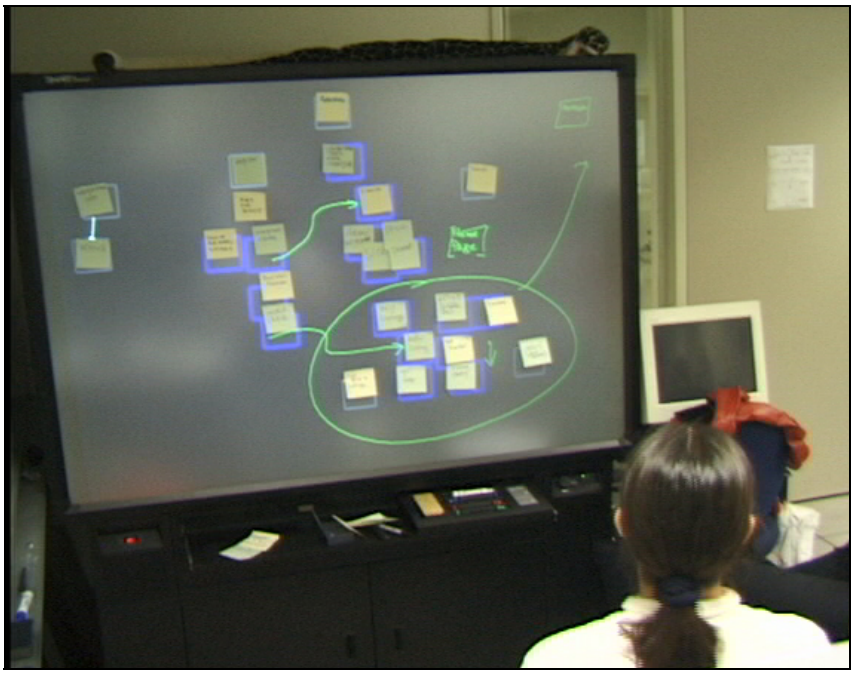


FIGURE 3.13 A design team suggested that freehand ink would be useful for both unstructured annotation of the artifact and for performing operations on groups of notes.

design process. The information architects all said that they currently create sitemaps by placing Post-it notes on the board, while the visual designers sketch page designs directly on the board. Capturing whiteboard designs was highly valued by all five teams. Three of the design teams currently use a digital camera for documenting their work, one uses a whiteboard capture device (the Virtual Ink Mimio), and one assigns a scribe to save information from design meetings. One group even uses an application called Whiteboard Photo [25] to rectify and filter out smudges, dirt, and lighting changes in whiteboard photographs. In addition, every designer said that they currently use either the Visio or Inspiration structured drawing software for creating sitemaps. Sitemaps can get quite large; designers from one firm said that two to three hundred nodes is typical.

Interactive board work process

We observed the groups going through three general phases of design when using the interactive prototype. The designers stated that these same phases were part of their

FIGURE 3.14 This is an example of the *facilitator* style; one person remains at the board guiding the group's process.

existing practice. (Two of the groups did not start the third phase during our in lab sessions, but said that it would be their next activity.)

Phase I: Brainstorming

First, the designers brainstormed, quickly putting a large number of concepts on the board. One designer said, "Get all these things on Post-its." The notes simply represent ideas. Sometimes, similar information was placed close together. Designers did not eliminate ideas or link concepts together into any formal structure at this stage.

One designer commented that Outpost would be, "good for times with the client" because after a meeting they could continue to pare down and hone the artifact without having to start from scratch with a new tool.

The designers were adamant about not wanting any system feedback during this phase. "We didn't do anything here that we couldn't do on a normal whiteboard." One team actually turned off the board.

FIGURE 3.15 This is an example of the *open board* style; all participants directly express their ideas in the artifact.

Phase II: Creating a top-level information architecture

In this phase, designers migrate from a loose federation of notes on the board to a high-level information architecture by clustering related information into groups, pruning unnecessary concepts, and linking notes together.

The tool support in the interactive prototype was well suited to this phase. This was evident in the designers fluid work style, and their enthusiastic comments while designing. This was echoed on the post-test questionnaire, where several designers expressed interest in using Outpost to create top-level information architectures.

Phase III: Drilling down—adding information with ink

After the designers created a rough cut of a sitemap, we saw work process differences begin to emerge. The visual designers began to work out basic page designs using empty board space and the board stylus. In contrast, the information architects fully fleshed out the page structure of the site, continuing to add notes.

A key design implication taken from this phase is the need for associating freeform ink with individual notes (see FIGURE 3.13). The visual designers wanted to sketch the design details, and the information architects wanted to add annotations or properties. For example, one information architect said, "I'd like to be able to attach design rationale." Design rationale is a mechanism for asynchronous communication, embedding in the artifact the motivations for making decisions [163]. The information architects also had a strong desire to use properties for project management. Two groups suggested tagging objects with properties, such as an issue (*e.g.*, "will it be possible to get copyright clearance?"), and later searching for issues across the design. Based on this feedback, we implemented annotation support as part of the design history interface (see CHAPTER 4).

Overall process

As reflects their disciplines, the visual designers often talked explicitly about what pages might look like, while the information architecture groups had active discussions about users and tasks at a more abstract level: "What does the user know here? What is the user trying to do?"

For all the teams, the site representation operated as the central shared artifact for discussion. Participants were actively working at the board only about half the time. In addition, for short periods of time (a minute or so), individuals or subgroups broke off from the main discussion to work. During these times, the board remained the anchoring reference point.

We observed two styles of interacting with the board. In the *facilitator* style, one person, usually the senior-most individual, stands at the board (see FIGURE 3.14). The entire group discusses the site, but as the discussion progresses, the facilitator creates notes that synthesize the discussion content. One group also referred to this style as "gate keeping." This was the primary work practice in three groups, and the groups affirmed this to be their normal work practice.

The second style was *open board*. As with *facilitator*, all group members actively discussed the site. In *open board*, however, there is no central person; all participants have agency to create notes and directly express their ideas in the artifact (see FIGURE 3.15). We started the sessions with a single pad of notes and a single marker next to the board. The first design team requested one pad and marker per person, which we provided for this and all subsequent groups. This paradigm affords each person their own paper "input device," a working style we had not considered but that the designers regularly employed. In adding content to the board, information moves from a personal creation space to a shared viewing space.

Several participants commented that they valued simultaneous input with a low-latency response. The SMART Board's touch sensor only supports one action at a time. (Newer SMART Boards that use SMART's DViT technology support multiple simultaneous touch points.) Concurrent use of the board has technical design implications for the note sensing technology. This result encouraged us to complete a computer vision system. Vision lends itself both to simultaneous input and to rich sensing capabilities (*e.g.*, object size, color, orientation, and capture of its contents).

The post-test questionnaire asked: "How likely is it that you would integrate Outpost as a regular part of your web site design practice?" Participants rated their response on a five point Likert scale. Four participants rated the system the top value, *very likely*. Eight gave the second value, *somewhat likely*. Three gave the fourth value, *somewhat unlikely*. One must be cautious about drawing strong conclusions from a participant's positive ratings: the positive rating may only reflect politeness toward the researchers. The primary value of a self-report Likert scale is that it provides participants an opportunity to indicate a negative opinion. With this in mind, the most valuable information from these results is that three of the participants reported that they would be "somewhat unlikely" to regularly use the Outpost prototype. We believe the primary reason for these participants' negative feelings was the distracting visual feedback in this prototype; the current system is much calmer. Additionally, in our research since the study, we introduced three substantial areas of functionality (transitions to other tools, support for design history, and remote collaboration) that provide important benefits unavailable with a whiteboard or other tools.

Only information architects need apply

Enthusiasm for the prototype correlated directly with two variables: the percentage of the designer's work that was web-based, and how much the designer saw their role as

an information architect rather than a visual designer. As one visual designer said, "We don't really do sitemaps so much. Our interfaces tend to end up with one or two screens." Information architects saw sitemap creation as the challenging process of designing the core structure of a web site. The information architects praised our faithfulness to their current wall-scale work practices, and were enthusiastic about the combined tangible/virtual interaction.

3.6 Design Implications

This study underscored several important points about how calm [258] an informal design tool must be; the system feedback should not interrupt the designers flow state.

3.6.1 Smart yet silent

We originally felt that one benefit of the prototype was the system's ability to automatically recognize groups based on note proximity and provide visual feedback. However, the designers unanimously felt that automatic grouping was not useful, as they already knew the layout of the notes.

Furthermore, the group, note outline, and menu feedback was considered distracting during brainstorming. One designer said, "I'm totally disturbed while I'm trying to concentrate on what we are doing. There are too many things flashing." In hindsight, this result is consistent with the negative user opinion about automatic interpretation and immediate feedback in SILK [142, 145], a sketch-based GUI design tool. In redesigning the system, we removed the visual feedback for proximity-based groups because the visual structure of the design is readily apparent—at best it can be redundant, and at worst it can be wrong. Additionally, we replaced the bright, crisp rectangles shown in FIGURE 3.13 through FIGURE 3.15 with the dim, penumbral shadow

as seen in figure 3.3. We designed this shadow to be at the just-noticeable of perceptual saliency.

Several participants valued the subtle visual relationships between notes. "Automatically arranging them would take away from my thinking." One designer said that she wanted "to work with this before it's turned on." A difficulty with automatic refinement of informal user input is that this refinement is distracting for users engaged in a creative brainstorming task. This implies that only explicit user actions should cause visible system actions. In general, designers felt that interactive feedback and transformation should not be forced on them: it should be *available*, but not *automatic*. As designers move from brainstorming into more explicitly creating a sitemap, their use of the interactive features will increase.

3.6.2 Sweet spot on the tangible/virtual spectrum

There are appealing aspects to both virtual wall-scale interfaces [178] as well as physical ones [180] (see figure 3.16). The Outpost project aims to leverage the advantages of both interaction paradigms.

Fluidity and physicality

This series of design studies provided insight into a sweet spot on the tangible/virtual spectrum. Working physically supports collocated collaborative processes. The direct manipulation affordances of physical notes make them easier to see, move, and share.

Electronic documents on a virtual desk	**Paper documents on a real desk**
Quick to edit, copy, transmit, share, file, and retrieve. Allows keyword searching, spell checking, instant calculations.	*Three dimensional, universally acceptable, cheap, portable, familiar, high resolution, easier to read. Tactile, can use both hands and fingers to manipulate, and can doodle on with a pencil.*

FIGURE 3.16 Pierre Wellner's comparison of advantages of electronic and paper documents [260].

We reviewed the study videotapes to quantify the pace of interaction. We found that on average, a note was added to the board approximately every 25 seconds. During active periods, a note was added every three to five seconds. Often, there was no explicit interaction for minutes at a time. A good portion of the meetings happened off the board, but referenced the board.

One facilitator began by authoring the sitemap virtually, sketching out square notes and their content. Working purely in the electronic domain has the advantage that there is no need to switch between an ink-based pen and a board stylus. However, working electronically was noticeably slower (top speed of one note every seven to ten seconds). This is because 1) the designer had to create page boundaries rather than using the pre-defined pages torn from a pad, 2) authoring with plastic pens on a plastic surface is awkward for textual input, and 3) the projector ink feedback is much lower resolution than paper. These difficulties negatively affected the artifact creation process, discouraging descriptive input. For example, in one instance, one participant wrote "B" instead of "Business" when using electronic ink. Later, he started working physically, and the working pace and artifact quality picked up substantially.

At least a whiteboard

In our designs, we were careful to preserve many of the successful aspects of working on a traditional whiteboard; the utility of these affordances became apparent in the study. Our system permits the representation of large, complex information spaces without the loss of contextual, peripheral information. One designer referred to our interface as "cross-cultural" because engineers, designers, and clients are all comfortable working informally on whiteboards.

Information appliances should be as easy to learn as physical appliances [198]. When two participants showed up half an hour late, we were pleasantly surprised to see

that the timely participant quickly brought her colleagues up to speed with Outpost's interaction techniques. After using the tool for only five minutes, she was easily able to communicate the conceptual model and the functionality of the prototype.

3.6.3 Extending the existing design process

Every group mentioned that migrating the design artifact to other tools for further refinement is an essential advantage of the Outpost system. Many of the designers currently photograph meeting whiteboards even though this only produces a static artifact. They were very interested in the prospect of returning to their desk with an interactive site representation that they could continue to work on.

An appropriate tool for Outpost designs to transition to is DENIM. DENIM offers the ability to edit the information architecture, specify page level details, and create the navigational structures for a web site. Its pen-based interface is intended for a single designer working at a PC. Outpost is most appropriate for creating sitemaps; whereas DENIM becomes more relevant when the design team starts to storyboard the specific pages and create schematics. The current Outpost system and DENIM read and write the same XML file format. This enables an individual to "save out a wall" from a collaborative design session and then flesh out the design on a personal computer or tablet. To support this, we augmented DENIM to handle images as page labels (see FIGURE 3.4 and FIGURE 3.5).

Long projects magnify the benefits of having a sitemap artifact remain in use throughout the entire design cycle. For example, one design team we spoke with was in the midst of a redesign for a large web site they had originally designed almost a year ago. Through its electronic capture functionality, we hope Outpost will help design teams with such long-term projects.

While an early interest in Outpost was to provide interactive support for information architecture design sessions, designers in the third design study found additional fruitful directions for our research. They encouraged us to refocus our efforts toward a more documentary interface, supporting free ink electronic annotations to sitemap pages, versioning of design artifacts, fluid transitions to tools such as DENIM, and supporting collocated and remote collaboration.

3.7 Computer Vision Prototypes

To illuminate the technology issues involved in building wall-scale tangible interfaces, we now briefly present the three computer vision prototypes we built. Outpost employs computer vision to precisely locate and capture Post-it notes and images that users place on the board. Computer vision is appropriate for this task as it provides automatic untethered and untagged tracking and capture of artifacts from multiple users simultaneously. The first prototype was a simple system that computed the difference image between frames, and analyzed this difference image to detect changes in the board state. In the second prototype, we used Matlab to prototype the full set of algorithms necessary to support the Outpost application. The third prototype was built on top of OpenCV, and implemented the computer vision algorithms at interactive rate, as well as a socket-based network connection for communicating with the Outpost UI.

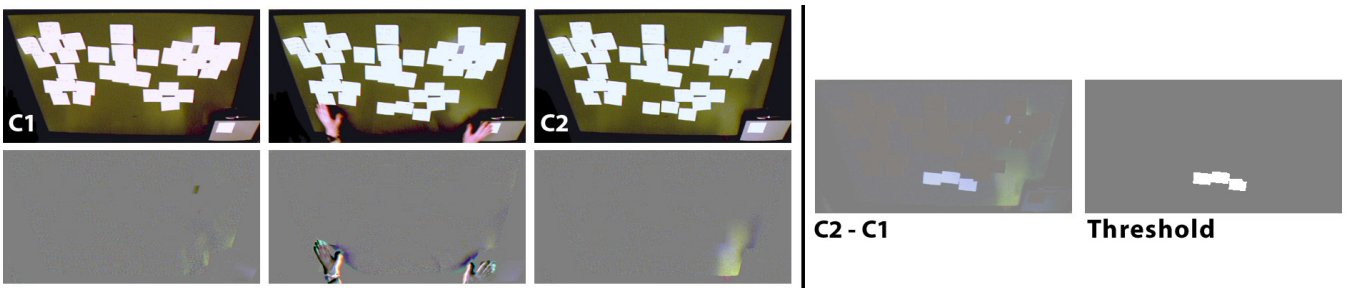


FIGURE 3.17 Excerpts from an image sequence from our prototype steady state algorithm. Raw camera frames are shown in the top row, single frame difference images are shown in the bottom row. Raw and thresholded C2 – C1 difference images are shown at right.

3.7.1 Difference image vision prototype

The foundation of Outpost's recognition system is a technique known as a difference image (see figure 3.17). Subtracting camera frame **m** from frame **n** results in the difference image $\mathbf{D}_{(n-m)}$. **D** expresses the change in board state between the two points in time.

Difference image algorithm and use

We originally employed difference images for two purposes: 1) to ascertain when users are working on the board and when it is calm, and 2) as an object detection primitive. A difference image is computed by subtracting consecutive frames (see the bottom row of figure 3.17). The content of a difference image expresses the activity of the board. An activity metric **a** can be computed by summing the absolute difference values of all pixels in the image. If **a** is larger than some threshold, the board is considered *active*. Otherwise, the board is considered *calm*. In reality, the content of a single frame difference image also results from noise in the camera sensor array and from lighting changes in the world (*e.g.*, someone walks between a light source and the board). The threshold used for this prototype was an absolute value change of 2.25 units per pixel, on a scale of 0 to 255. The goal of ascertaining activity was to find a calm frame that could be compared with the current frame to find changes in the board state. This activity metric is a fairly crude method that usually, but not always, filtered out both sensor noise and lighting changes. The method is crude because the decision of whether an image is calm, and thereby useful for a baseline, is binary. In the wall vision prototype described in section 3.7.3, we replaced the activity method with spatial filtering and temporal averaging techniques that provide a more robust baseline image for comparison. Filtering and averaging obviates the need to make a binary decision about fitness, as the new baseline image is a moving aggregate of a number of frames.

This prototype also used difference images as an object-detection primitive. Outpost does not require that notes be tracked while they are moving, but it does require that the system is aware of a note when it is initially placed down, placed at a new location, or removed. When the board becomes *active*, the system saves the last *calm* frame **c1**. When the board becomes calm again, it captures the new calm frame **c2**. Subtracting **c1** from **c2** and thresholding the result tells the system what has changed during that period of activity (the right-hand image in FIGURE 3.17). This thresholded difference image becomes the input for note recognition.

Design implications

In building this first prototype, we realized that *locating* a note is a separate task from *capturing* a note. Dividing the vision task into these two distinct parts enabled us to realize that the system should use two cameras (see FIGURE 3.18). To obtain an occlusion free view of the board for our difference image algorithm, we followed the metaDESK

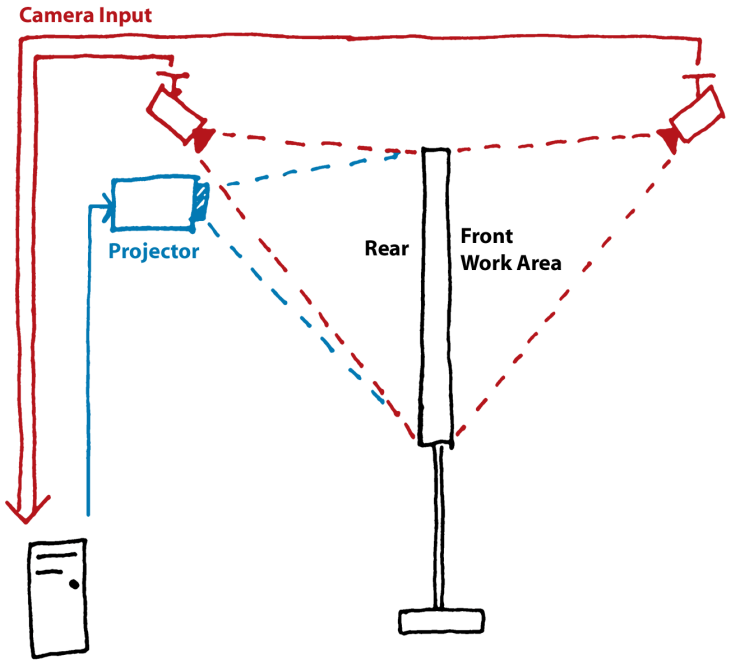


FIGURE 3.18 The physical design of the Outpost system. The computer vision system uses two cameras as input devices for the electronic world. A rear-mounted projector outputs electronic information onto surfaces in the physical world.

researchers [249], mounting a video camera behind the board. Interactive frame rates are crucial for this camera. Because the notes are fairly large (three inches square), standard video resolution (640 × 480) is sufficient for location and orientation detection.

Ink capture has the opposite set of constraints: it requires high resolution for capture but not interactive speeds because the ink capture does not control the board feedback. This suggests a high-resolution still camera. This two-camera approach obviates the need for a mechanical pan/tilt/zoom camera and image stitching algorithms.

When we began this project in 1999, we found that consumer grade web cameras compressed images in ways that make computer vision difficult [251]. The situation has since improved substantially; today, consumer grade cameras are generally sufficient for the types of vision that Outpost uses.

3.7.2 Matlab algorithms prototype

Using the difference image described above as a building block, we designed and prototyped the complete vision pipeline in Matlab before implementing it in an interactive system. This prototype introduced perspective correction, segmentation, and feature extraction. We used Matlab because writing vision code with development libraries such as OpenCV takes much longer than with Matlab, prohibiting us from experimenting with design alternatives. This need for rapidly exploring multiple alternatives helped inspire the Papier-Mâché research. This pipeline performs the following set of operations on $\mathbf{D}_{(C2\text{-}C1)}$:

1 Rectify the perspective camera view of the board plane, bringing the board into a 2D plane using a 3 × 3 homography map matrix [83, § 13.1]. A homography matrix describes an arbitrary projective transformation. More precise algorithms for camera calibration exist; we chose a homography because it is very fast.

2 Threshold the resulting image, producing a three-level image. *Positive* pixels are pixels that have gotten significantly brighter, *neutral* pixels have not changed much, and *negative* pixels have become significantly darker.

3 Segment the image using the connected components algorithm, labeling each positive and negative pixel with a cluster ID number. The system interprets note-sized clusters of positive pixels to be added notes, and note-sized clusters of negative pixels to be subtracted notes.

4 Compute the center of mass and the orientation of the note. Inspired by Freeman's work [84] (see FIGURE 3.19), we originally implemented orientation-finding using a second moment algorithm. However, we realized from our prototype that the second moment is undefined for squares, circles, and other objects that are symmetrical about both the X and Y axes: for these shapes, all orientation choices yield an identical second moment. For this reason, we moved to an expectation-maximization (EM) algorithm [83, § 16.1] that finds the best fitting square on the set of outline pixels of the note. This method is highly robust, even for a highly degraded image outline and small sample size. Theoretically, EM is a more expensive algorithm because it is iterative. In practice, we have found that a very small number of iterations (in our case six) is enough for the solution to comfortably converge.

3.7.3 Interactive wall vision prototype

We combined the system implications from our first vision prototype with the algorithms from our Matlab prototype to produce a wall-scale interactive prototype. This system employs three sensors: 1) a touch sensitive SMART Board, 2) a rear-mounted 640 × 480 industrial digital video camera, and 3) a front-mounted three megapixel USB still camera (see FIGURE 3.18) to achieve the multiple person, low-latency input and capture desires that interested the participants in our study. This prototype offers an interactive-rate solution for detecting the location of notes.

The Outpost vision system is written in C++ on top of the Intel OpenCV library [40]. The vision system and the user interface run as separate processes and pass semantic events (*e.g.*, *add*[x, y, θ], *remove*[x, y]) through a socket network connection. Currently, both processes run on the same computer. A benefit of this socket architecture is that it allows the processes to run on separate computers without modification. The only reason to avoid running the vision and the UI on different computers is network latency.

When the system starts up, it automatically detects the corners of the board. It does

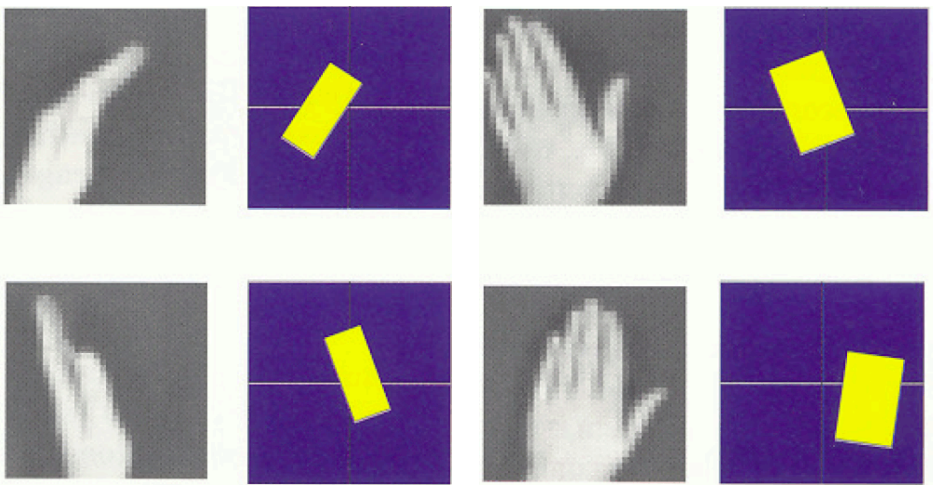


FIGURE 3.19 Hand images (from a low-resolution detector) and equivalent rectangles, having the same first-order and second-order moments [84].

this by projecting a white border on the board and capturing a frame, setting the projector completely black and capturing a frame, computing the thresholded difference image between the two frames, and finding the set of outline pixels. We use EM here as well, finding the best fitting four lines on the set of outline pixels. We use these four corners to automatically compute the homography transform. In this prototype, our camera distortion is small. We originally intended to apply a homography transform to every **C2 – C1** difference image. We substantially improved the performance of the rear camera's vision system by applying the homography transform only to the logical coordinates of the detected note corners. Thus, we transform only four points per note found, as opposed to 640 × 480 pixels per difference image; a savings of roughly four orders of magnitude. This improvement is possible because the only relevant information from the rear camera is the location and orientation of notes, and this information is uniquely determined by the four corners. This optimization is not possible with the front camera because Outpost uses the actual pixels from the front camera to display electronic notes. However, it is not necessary to transform the entire camera image, only each sub-region containing a note.

We also revised our mechanism for finding the location of a note. Because the notes have a sticky stripe across the top, the top edge is flush with the board, straight, and accurate. Sometimes the bottom and side edges curl away from the board, however. (This observation generalizes to pictures and other paper artifacts taped onto the board.) Outpost solves this by computing a four-line EM on the note outline, and selecting the top line of those four to compute orientation and location. This shortcut allowed us to move beyond the vision details so we could address Outpost's user experience issues. A production implementation of Outpost should avoid this shortcut; this can be avoided with a vision algorithm that has a stronger notion of shape (such as Shape Contexts [35]).

For the most part, this prototype was successful; the main difficulty was with process scheduling on a one-processor machine under Windows 98. Often, either the vision or the interface process was given use of the processor for extended periods of time. Because both need to run interactively to achieve interactive performance, we moved the system to a two-processor machine with Windows 2000. This resolved the scheduling issues.

3.8 Current Implementation

The current Outpost system consists of two main components. The interface component handles stylus, physical tool, and touch input on the board, and provides graphical feedback to the user. The computer vision component tracks and captures physical Post-it notes and pictures.

3.8.1 Physical tools and graphical display

The physical tools input and graphical feedback are implemented in Java using SATIN, a toolkit for informal pen-based user interfaces [110], and the SMART Board SDK. In Outpost, we make use of SATIN's extensive support for ink handling, gesture recognition, and rendering. Free ink in Outpost is captured and saved as a *stroke* primitive. We use a tap *interpreter* for invoking context menus on existing notes. We also use a gesture *interpreter* for drawing links between pairs of notes.

The SMART Board's tool tray consists of four pen tool slots and one eraser tool slot. The hardware detects the presence of the tools via a photometer in each slot. The hardware defines the active tool to be the tool most recently removed from the slot. The tools themselves are passive. The SMART Board SDK uses callbacks to inform registered applications of the current tool. We use this mechanism to know when the move tool or the eraser is active instead of the pen.

3.8.2 Computer vision infrastructure

Outpost's vision system supports simultaneous input; essential for collaborative design. Our vision system is written in C++ on top of OpenCV [40]. The vision system runs as a separate process, passing semantic events to the Outpost UI through a socket network connection.

This system offers interactive rates (~7 frames per second) for detecting the location of notes with the rear camera, combined with background high-resolution capture (~1.5 second latency) for virtual display and transitioning to DENIM. This design achieves the multiple-person, low-latency input and capture that interested our study participants. One way to think about the board capture is as a direct manipulation scanner. One operation, placing a physical document on the board, specifies both the location of the document and that the document should be captured.

Interactive vision techniques

There are several processing steps that we perform with each new image from the rear camera (see FIGURE 3.20). First, we employ spatial and temporal filtering techniques that help alleviate problems due to camera noise and lighting changes. This is a common and effective technique in many computer vision applications. Our temporal filtering computes an exponential weighted moving average (EWMA) image μ_t by recursively averaging in each new frame f_t with weight α. In Outpost's case, $\alpha = 0.04$, which is a fairly typical value.

Each frame, we rectify the perspective camera view by bringing the board into a 2D plane using a projective transform matrix. There are more precise algorithms for camera calibration; we chose a simple perspective warp because it is fast, and works well for our purposes.

Next, we construct two thresholded difference images. Objects placed on the board and people moving in front of the board cast a shadow on the board's surface. To the rear camera, areas with objects are darker than the empty board. When an object is removed, the area becomes lighter than it previously was. Added notes are found in the ($\mu_{t-1} - f$) image (darker areas are positive) and subtracted notes in the ($f - \mu_{t-1}$) image (lighter areas are positive). We segment the two binary images using the connected components algorithm, finding note-sized components from changed pixels.

After segmentation, we compute the center of mass and the orientation of the note components. We use an expectation-maximization (EM) algorithm (for a good overview, see [38]) as a robust method for finding the best fitting square on the set of outline pixels of the note. We then compute the homography transform from image coordinates into board coordinates.

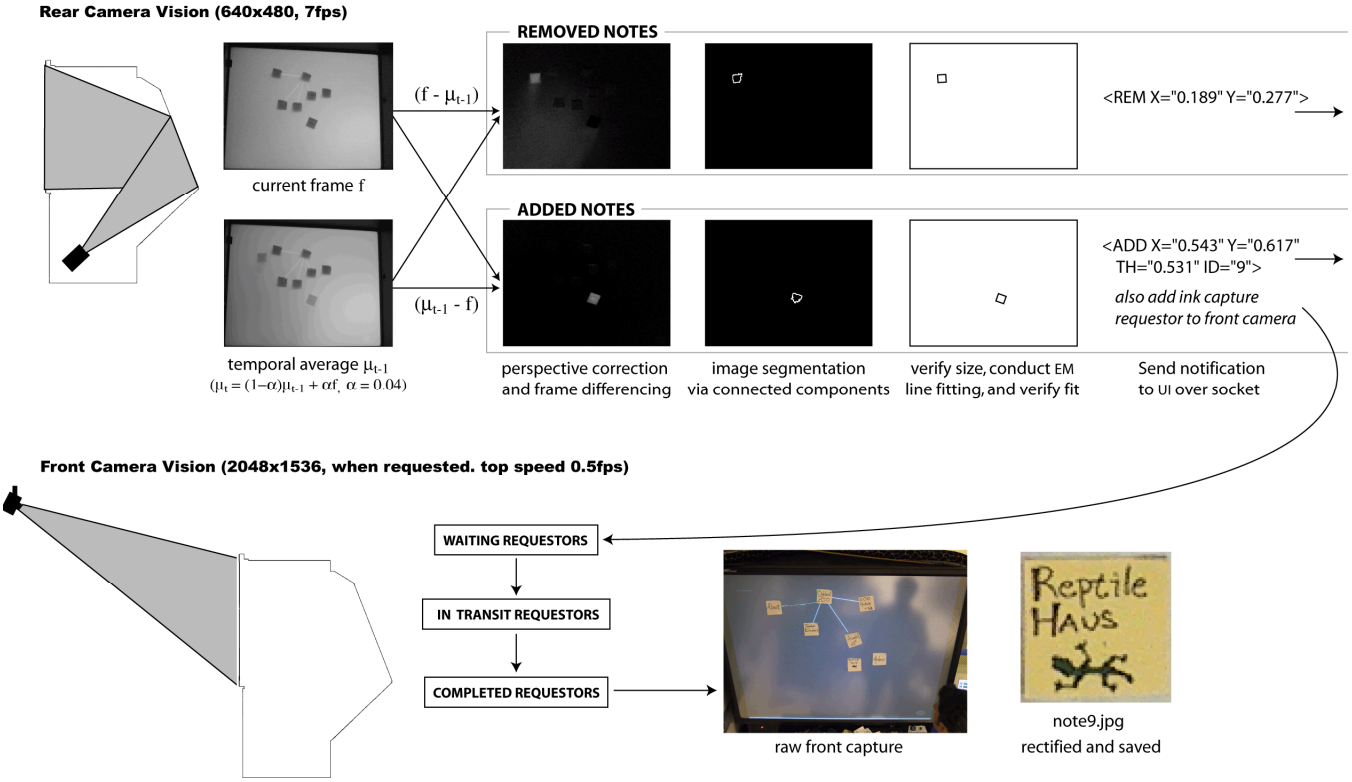


FIGURE 3.20 The Outpost vision pipeline at a frame where one note ("Reptile Haus") was added and another was removed.

As a final step, we require that added objects be found in the same place for two consecutive frames. We added this step to reduce false positives to a negligible level. At the completion of this vision pipeline, we send the semantic information about board state changes over the socket to the Java user interface.

To minimize our computational overhead, the front camera only takes a picture when the rear camera detects that a new note **n** has been placed on the board. For each **n**, we add a *requestor* to the front camera's request queue with [x, y, θ, ID] as the location to capture. As soon as the front camera is available, it takes a picture. The system corrects for perspective skew upon receiving the picture. For each *requestor*, the system saves the rectified area of the board as a JPEG file. This method insures that note capture will complete soon after the note is placed on the board (capture completion time is bounded by twice the image transfer time). It also enables multiple notes to be captured from a single image.

Outpost can run in a calibration mode, where it automatically detects the corners of the board and saves the calibration parameters to a file. We do this by capturing a frame of an entirely black board, capturing a frame of the board with a projected white outline, computing the thresholded difference image between the two frames, and finding the set of outline pixels.

Discussion

We designed the vision system to be highly robust at finding notes. The occasional recognition errors fall into three categories:

- *Missed actions:* There are a few cases where the vision system misses an add or remove action (~1% of the time). This is usually because a person is standing in front of the note, casting a shadow on the board. As visual feedback, the UI displays a faint shadow around recognized objects. When new objects are not recognized, the user must

perform the add action again. Missed deletions can be fixed using the delete option on the context menu.

- *False positives:* Rarely (~2-3% of the time), the system reports an action that did not happen; this is nearly always because the system perceives a user's closed hand to be a note. As a UI solution, we offer the delete option on the context menu.
- *Location and orientation misreporting:* In this system, there are two kinds of accuracy: resolution and calibration. Our system performs adequately in both regards. As a point of comparison, most of the time our vision system is of higher accuracy than the board's capacitance sensor; a more sophisticated camera model could improve this further.
- *Occlusion of the front camera:* Currently, the front camera takes a photograph of the entire board whenever it is requested to. The image processing system then clips out the requested portion of the image, which corresponds to a photograph of the requested note. On occasion (~2% of the time), a person is standing between the camera and the board, occluding the camera's view of the note. This happens only rarely because the camera is ceiling mounted and close to the board.

These issues can be remedied with improved vision algorithms. For example, we may be able to assume that a cluster with three straight sides and an oddly shaped fourth side is really a note with a user's hand on the fourth side. (Currently, we may discount this cluster because it is too large to be a note.) The system might also benefit from using edge detection as opposed to simple thresholding. We did not implement this in Outpost, but our interest in exploring different vision algorithms inspired the modularity in the Papier-Mâché architecture, and the inclusion of edge detection as an algorithm in the Papier-Mâché library (see SECTION 8.2).

Perhaps more importantly, these errors remind us that intelligent interfaces make mistakes. This has encouraged us to rely more on "simpler" sensing technologies,

namely the board's touch sensor. For example, until the vision was robust we required that the smart board touch sensor reported a tap. This tap—possibly a user placing a note) created a candidate note to the model at an approximate location. The model then queried the vision system, asking if any notes were added in the region of the touch. The vision system responded either *yes* or *no*. If a note was added, the vision also responded over the socket connection with the precise location and orientation information it found. The limitation of this dual sensor approach was that the SMART board touch sensor could only report one touch at a time. Once the vision was robust, we found the vision-only approach preferable, as it provides simultaneous input.

3.9 Summary and Toolkit Motivations

We have presented The Designers' Outpost, a tangible interface for collaborative web site information design. Its functions are informed by observations of real web site design practice, providing many of the affordances of current paper-based practice while offering the advantages of electronic media.

Outpost is implemented on top of a vision system that yields an interactive-rate system for robustly finding notes on a large surface. These results and those of other researchers show that computer vision is an effective technology for informal, collaborative interaction with physical media on walls.

We validated our design with fifteen professional designers, showing that electronic whiteboards should be calm and that there is substantial merit in a system that is simultaneously tangible and virtual. The designers were enthusiastic about using Outpost and achieving the fluid transition from artifacts on walls to single-user tools such as DENIM.

While the Outpost user interface is highly effective, developing this system was very difficult and time-consuming. Inspired by Myers' and Rosson's code analysis survey

[187], we have estimated the development time and code size of our Designers' Outpost application, divided into the time and code size devoted to developing the *tangible* and the *electronic* portions of the system. Development time was estimated by reviewing the author's calendar. By this estimate, building Outpost would have taken roughly four months instead of three years if it were built as a GUI. 88% of the development time for Outpost was learning about and assessing the relevant physical sensing technologies (in our case, computer vision), becoming a fluent developer with the technology-centric APIs, and implementing this complex technology enabling fluid human-computer interaction.

In Outpost, 57% of the lines of code and 88% of the development time are primarily concerned with the tangible aspect. This 57% number was computed by labeling all software files as *primarily tangible* or *primarily electronic*. All C++ files (the vision engine) were primarily tangible. Each of the Java files was categorized based on its primary function. These results indicate that similar reductions in development time, with corresponding increase in software reliability and technology portability, can be achieved by a toolkit supporting tangible interaction.

Outpost required us to learn a substantial amount about computer vision algorithms, and design software architectures for tangible interaction. At the time we developed this system, tools support for this development process was minimal. While libraries such as OpenCV provide efficient implementations of image processing techniques, it is still the developer's responsibility to 1) understand the behavior of these techniques, 2) string together low-level image processing primitives to achieve the high-level user interface goal, and 3) create a software architecture for communication between the vision input system and the UI. The Papier-Mâché architecture, described in CHAPTER 8, addresses these issues.

4 Electronic Design History of Physical Artifacts

To form a deep understanding of the present, we need to find and engage an account of the past. This chapter presents an informal history capture and retrieval mechanism for collaborative, early-stage information design. This history system is implemented in the context of the Designers' Outpost, a wall-scale, tangible interface for collaborative web site design. The interface elements in this history system are designed to be fluid and comfortable for early-phase design. As demonstrated by an informal lab study with six professional designers, this history system enhances the design process itself, and provides new opportunities for reasoning about the design of complex artifacts.

4.1 Introduction

To keep track of project milestones and variations, designers are forced to invent ad-hoc methods, usually involving saving multiple versions of files and using complex, cryptic file names to encode the properties of each version. In the physical world, they must manually photograph, photocopy, or scan an artifact to save its current state, or abandon this state and keep working.

People invested in understanding the trajectory of history from the past to the present include decision makers, students, designers, and their successors. These

---

Portions of this chapter were originally published by the author, Michael Thomsen, Ethan Phelps-Goodman, Robert Lee, and James A. Landay in [136]

stakeholders engage history through creation, revision, and reflection. In this chapter, we present an informal history capture and retrieval mechanism that supports these activities for collaborative, early-stage information design. A video of the system is available at http://guir.berkeley.edu/outpost/video/History.asx.

Our history system is implemented as an extension of the Designers' Outpost. We present three mechanisms for accessing design history: a *main timeline*, a *local timeline*, and a *synopsis view*. The *main timeline* is a visually navigable set of design thumbnails organized on a timeline (see FIGURE 4.1). This view can be filtered by activity (By Actions, By Bookmarks, or By Meeting) or by inferred properties (By Time, By Note, or By Author). We employ a branched history, presenting the current branch to the user as a linear history. This linear history is annotated with *stubs*, indicating the existence and position of other branches. It is possible for users to jump to any point on

FIGURE 4.1 Users' view of the main history timeline (*bottom*) in the Designers' Outpost, a system for collaborative web design. Outpost runs on a touch sensitive SMART Board.

the timeline, including semantic places such as when an object was created. The *local timeline* enables users to see, in the actual design, a history with just the actions relating to an individual object in the design. The *synopsis view* enables post-design review of key bookmarks. These bookmarked states can be annotated with text, and printed as hard copy for easy portability and sharing.

We designed the history system around a set of scenarios that we distilled from design fieldwork studies. We present four here: 1) reaching an unproductive point, and heading off in a new direction from an earlier point; 2) writing a summary of a design session; 3) finding the rationale behind a decision; 4) creating a set of action items from a design session.

In addition to supporting web design in a collaborative wall-scale system, our work on design history should in many aspects be transferable to other professional practices that center on the creation of an artifact by several individuals over an extended period of time. We hope that this work will inspire research on tools for other professional domains as well.

4.2 Background

Our research is inspired by work in design rationale, in history-aware systems, and in capture and access applications.

4.2.1 Design rationale

Much of our thinking about design history is motivated by *Design Rationale: Concepts, Techniques, and Use*, edited by Moran and Carroll [177]. The sixteen contributed chapters characterize the primary goal of design as giving shape to artifacts—design products—yet underscoring that "the artifact is a concrete form that does not (except in very subtle ways) manifest this process of creation."

Semi-formal design rationale: IBIS and QOC

A number of design rationale systems have been proposed in the past, such as the seminal IBIS [214] system in the 1970s, and more recently, QOC [162]. These systems employ semi-formal syntaxes to capture design rationale in the form of argumentation surrounding decisions made during a design process. These systems have not achieved widespread use with designers, possibly because they impose a rigid structure on design thinking and burden designers with creating and maintaining a separate rationale representation in parallel with the design itself.

VKB: history in hypertexts

The VKB system [226] introduces the notion of "constructive time," which is the reader's experience of accessing a history in a hypertext. Our third scenario, finding the rationale behind a design decision, draws inspiration from the VKB notion of history being created for the benefit of an external viewer. Our *By Meeting* filter is implemented in a similar fashion to the VKB meeting discretization.

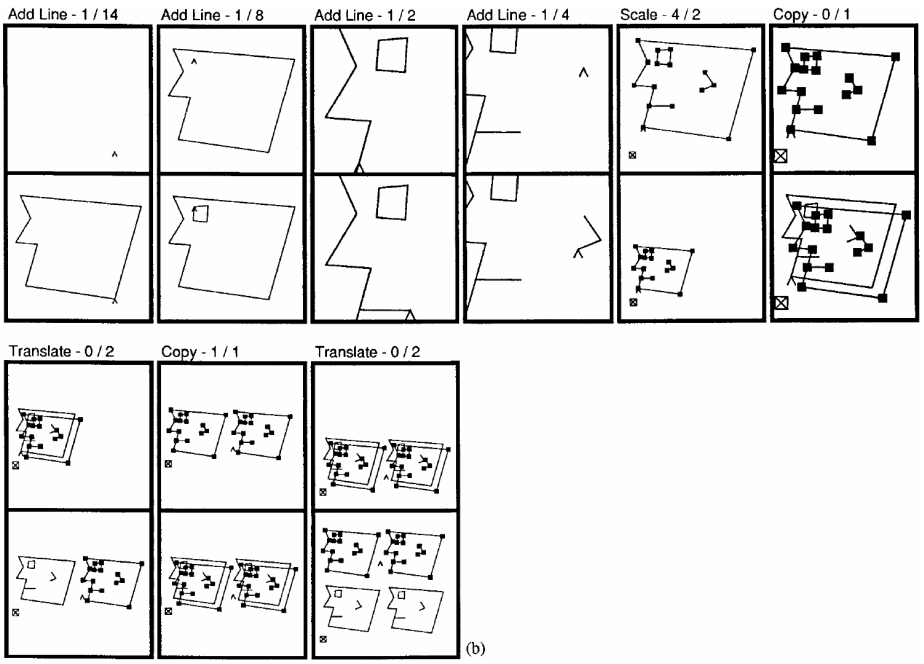


FIGURE 4.2 Kurlander's editable graphical histories [141].

4.2.2 History-enabled applications

The visual design of our global timeline is inspired by the Chimera graphical editor [141] (see figure 4.2), which introduced the comic strip metaphor for displaying a history of changes. Chimera also used highlighting to focus on the parts that changed in each frame. Our timeline extends the design in Chimera with numerous ways of filtering the displayed frames, and with display of branched history.

Rekimoto introduced the Time-Machine Computing (tmc) system (see figure 4.3) [205]. The tmc browser replaces a standard desktop browser, offering time as a unifying method for storing personal information such as documents, digital photographs, notes, and calendar information. Like the Outpost history system, it affords for time-based browsing. Additionally, like our local history system, the tmc browser allows users to select an object and return to the time it was created, a type of direct-manipulation query. An interesting personal information management feature in the tmc browser is that it enables users to place objects in a future point in time. The major difference between tmc and our system is that our system is intended as a collaborative

FIGURE 4.3 Rekimoto's Time-Machine computing system [205].

design aid, while TMC is intended for personal information management. Outpost also offers support for branched history which TMC does not. However, there are some techniques in TMC that might be beneficial to our system, such as a calendar view of information.

The WeMet system [211] for distributed, collaborative drawing also automatically captures history. According to the authors, this allows users to "reconstruct the present," and allows parties that join work in progress to review the work performed by the other participants so far. WeMet inspired us to include explicit bookmarks in the history.

Capture and access

Our focus on informal interaction [145] leads us to shy away from structured approaches and borrow from another thread of research, informal meeting capture. Informal capture systems attempt to collect information from users in natural ways, *i.e.*, information that they produce in the normal course of their activities, and attempt to structure it in useful ways for later retrieval.

The Classroom 2000 project [26, 46] captures information from multiple sources including audio and video of classroom lecturers, ink from students' notes and annotations, and lecturers' presentations slides. This information is then merged and indexed in order to support students' task of reviewing lecture notes.

The AudioNotebook [237] and the Dynomite system [263] both focus on personal information capture, and provide interaction techniques for browsing histories. Although they are concerned with audio, their methods for creating inferred bookmarks inspired us. AudioNotebook creates bookmarks based on pauses and changes in pitch, inspiring us to add inferred filters in our history system. Dynomite's ink properties inspired us to add author information to created notes. Outpost differs from these systems in that it is designed to support collaborative, rather than individual, practices.

The CORAL [179] system captures and coordinates data from multiple sources such as audio and whiteboard notes to support meeting capture. These four systems share with Classroom 2000 a task-oriented focus on visualizations to support later retrieval.

4.3 Motivations for History Support

Before building this history system, we conducted and learned from field and design studies. Newman and Landay interviewed eleven professional web site designers, providing us with two important insights. First, designers create many different intermediate representations of a web site. Second, "Designers expressed a desire to have a unified way to manage different *variations* of design ideas. Variations play a key role during the design exploration phase, and it would behoove an effective design tool to help support their creation and management" [191, p. 273].

During the creation of the Designers' Outpost, we conducted a design study with fifteen professional web site designers, which guided the design of the basic system as described in CHAPTER 3. The importance of support for design history became clear in this study: the participants stated that they often forgot the history of how some part of their design came to be, or they would alter their design and then realize that they preferred an older design. These studies both gave us insight into the working practice of web designers, and motivated our focus on better supporting design history.

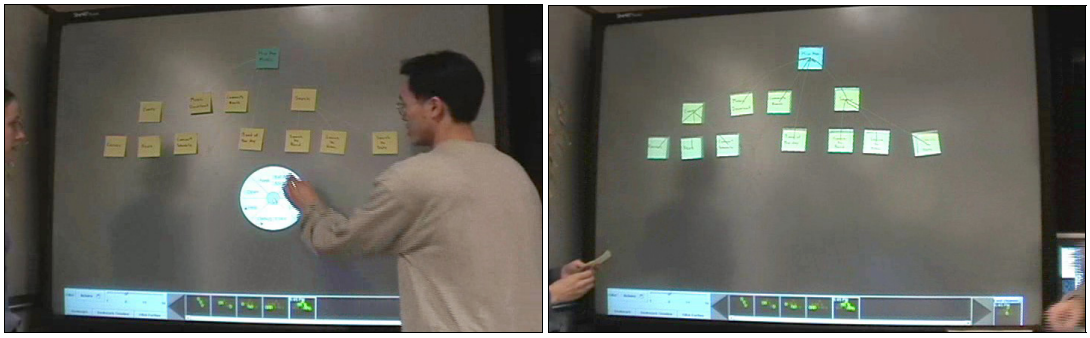


FIGURE 4.4 Outpost's electronic capture enables replacing physical documents with their electronic images. A pie menu operation (*left*) makes all notes electronic (*right*). It is easier, but not required, to work with design history when all of the information is electronic.

4.4 History Interface

To promote design history, we have made the history a first-class citizen (as in Timewarp [70]), while keeping the design history as a relatively calm entity that does not distract from the main design work at hand. Our system provides three facilities for interacting with design history: a *main timeline*, a *local timeline*, and a *synopsis view*. It is easier, but not required, to work with design history when all of the information is electronic (see FIGURE 4.4). We describe the functionality of each of these and then illustrate their utility in the context of the brief scenarios presented previously.

4.4.1 Timeline visualization

The main timeline displays a history of the design using thumbnails, as seen in FIGURE 4.5. Each thumbnail presents the contents of the board at the time of capture. To support the user in determining what has changed between adjacent frames, we highlight the elements that were altered in the most recent frame (see FIGURE 4.6).

Filtering thumbnails

Designers use the timeline display to choose the set of thumbnails to display. In the most detailed view, the timeline displays all thumbnails—one thumbnail per single action the users have performed at the board, such as adding a note or moving a

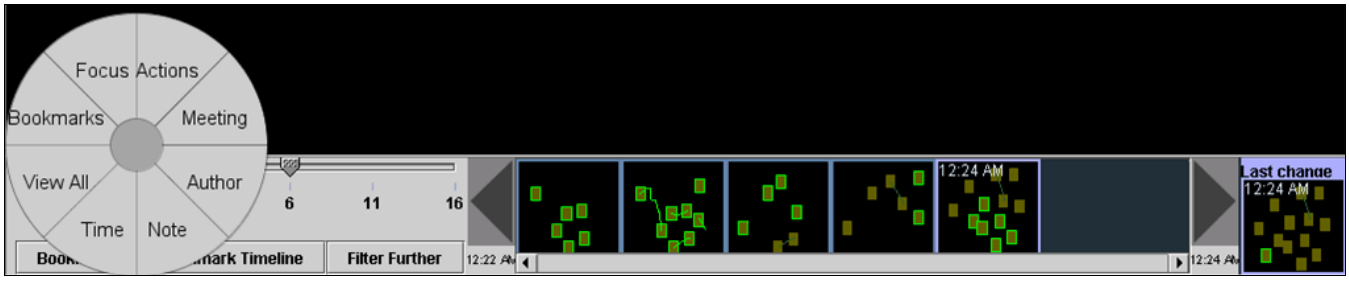


FIGURE 4.5 The main timeline at the bottom of the SMART Board. The pop-up pie menu lets users choose available filters. *Bookmark* adds the current state to the synopsis. *Bookmark Timeline* adds all states in the current view to the synopsis. *Filter Further* allows users to intersect filters.

relation. *View all* is useful for local undo, but more substantive interactions mandate using our other filters.

There are two types of filters: *activity filters* and *inferred filters*. The activity filters are based on explicit actions made by the user; they comprise *By Actions*, *By Bookmarks*, and *By Meeting*. For each of these, the user can select to have a thumbnail generated for every *n* Bookmarks, Actions, or Meetings.

Inferred filters allow a user to filter on properties that are not explicitly set by the user; they comprise *By Time*, *By Note*, and *By Author*. In the *By Time* filter, the user chooses to see only frames that correspond to actions carried out every *n* seconds. With the *By Note* filter, the user can select a note on the board to view only frames that correspond to actions performed on or in relation to that note. Finally, when using the *By Author* filter, only frames that correspond to notes altered by the chosen author are displayed.

The *By Author* filter is one example of many possible context-sensitive history queries. Here, the system needs to sense who is the author of each operation. Although this could be implemented using an RFID tag and a reader behind the board, we currently simulate this using a Wizard of Oz approach [128, 172]: during design sessions an operator indicates the person currently at the board in a simple list of names shown on a secondary monitor.

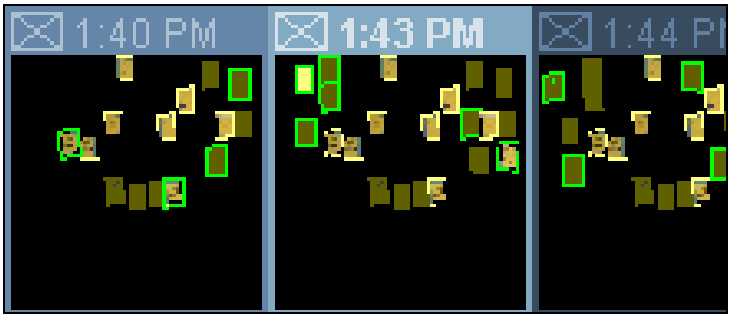


FIGURE 4.6 Close-up of the global timeline. Above each thumbnail is a time-stamp. The main thumbnail is a scaled down version of the board, with the changes highlighted in green. The frame around future thumbnails is dark blue, past medium blue, and current light blue.

Timeline navigation

Thumbnails not only display information about the state changes of the board, they also provide a direct-manipulation interface for navigating the design history. Pressing any thumbnail causes the system to undo or redo all commands that have been issued since that point in time, restoring the board to that state. This multi-level undo/redo allows designers to experiment with the information design, as returning to a previous state is always possible (see FIGURE 4.7). While the electronic touch interface works very well for selection, we found that electronic GUI widgets are clumsy and slow for scrolling and browsing. To support more fluid scrolling, we integrated a Contour Design USB jog dial [22] (see FIGURE 4.8) for direct physical interaction with our system. Snibbe *et al.* have shown that jog dials with haptic feedback are highly effective for browsing time-based media [232]. Presenting semantic information (such as branches) haptically would likely be of great benefit to users of design history systems.

While our studies have shown many benefits to a paper-centric tangible interface for freeform design, the physicality of a design artifact becomes problematic when engaging its history: it is not possible for the system alone to perform an undo operation for all possible physical actions made by the user, such as adding or removing a note from the board. In our system, a combination of user actions and history manipulations can yield one of two degenerate cases: either the current view calls for presenting an object without a physical presence, or there is a physical object that should not be present. In the former case, we display the electronic capture of the object. In the latter,

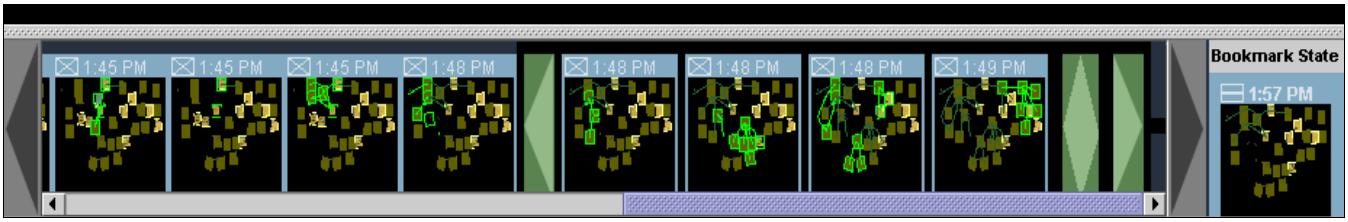


FIGURE 4.7 The main timeline, with an expanded strand containing a collapsed strand.

we give the object a red shadow to indicate it should not be present, hinting that it should be removed by the user.

Branched time visualization

As mentioned above, the system supports multi-level undo/redo. One simple example of multi-level undo/redo is the functionality present in Microsoft Office. This functionality does allow unlimited undo and redo, but one major problem is that only one strand of actions is held in the history. If the user performs actions **A**, **B**, **C**, **D**, **E** undoes three times, and then performs actions **F** and **G** (the sequence shown in FIGURE 4.9); the current action strand in the history is **A**, **B**, **F**, **G**. The fact that **C**, **D**, and **E** used to follow **B** is lost in a linear history. Some undo/redo systems, such as the one in Emacs [234], offer the user a truly branched history. However, branched histories have traditionally been difficult to navigate; the user is likely to get lost because it is difficult to build an accurate and complete mental model of the history tree.

Our goal was to preserve the entire history, with all constituent action strands, without introducing unwieldy complexity. We achieved this by merging the concept of a branched action history with the linearity of a single stranded history. One possible

FIGURE 4.8 Physical jog dial for scrolling through history.

way of presenting the branched history is as a branched tree as in figure 4.9. While this way of displaying the multiple strands presents the whole history and its two constituent strands, the visualization rapidly becomes complex as the number of branches increases. This complexity creates too large a user burden for our domain of informal, early stage design, and it requires substantial screen real estate.

As a lighter weight alternative, we present the history as a linear list of actions, where inactive branches are represented by a collapsed *stub*, as illustrated in figure 4.7. This presentation preserves the temporal order of the actions; a frame presented to the right of another frame corresponds to an action that was issued after the other. It also scales well; multiple branches can be shown inside each other by nesting the stub parenthesis markers as shown along the bottom of figure 4.10. Users can open or close (collapse) any branch, choosing a presentation of the timeline relevant to the objects of interest.

Local timeline visualization

The main timeline visualizes the history for the whole board; the local timeline provides a lighter weight history for an individual object. When selecting a note by tapping it, an object menu is displayed (see figure 4.11). The object menu supports both common operations such as deleting the note or making it persistent, as well as displaying a small note history along the bottom. This novel *in situ* timeline offers the user more detailed

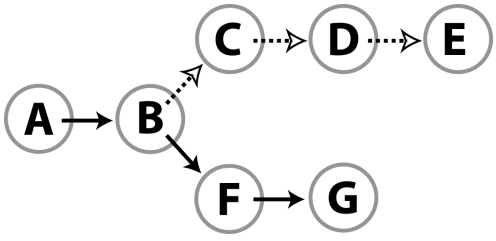


FIGURE 4.9 Branched history: Actions A, B, C, D, and E form one strand; A, B, F, and G form the other.

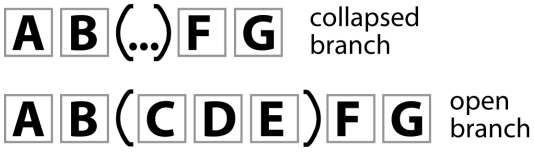


FIGURE 4.10 Stub-branching history presentation: the top history fully displays the current strand; other strands are visualized as stubs. The bottom history displays the full history; states not part of the current strand are placed between brackets.

information about a particular note without visually cluttering the entire board.

4.4.2 Synopsis visualization

One advantage of electronic capture is its ability to support radically different presentations of information. The synopsis visualization is an example of this. This visualization was designed to provide an annotated record of the set of changes that happened to a design. We provide the ability to work with this list electronically (see FIGURE 4.12) or printed on paper (see FIGURE 4.13). Because users are not always at the board, a printout serves as a take-away design record for sharing, discussion, and annotation. The synopsis visualization fills these needs.

A synopsis can be constructed in two ways. First, it can be constructed via explicit user *bookmarks*. Bookmarks can be created at design time when a team arrives at a spot worth marking or they can be created after the fact by going back to a point in the timeline and bookmarking it. Users can view their set of bookmarks when viewing *By bookmarks.* A synopsis can also be constructed from a filtered history view (*e.g.*, every twelve actions). A user can select *bookmark timeline* to add that set of states to the synopsis. These two techniques can be combined to manually augment an auto-

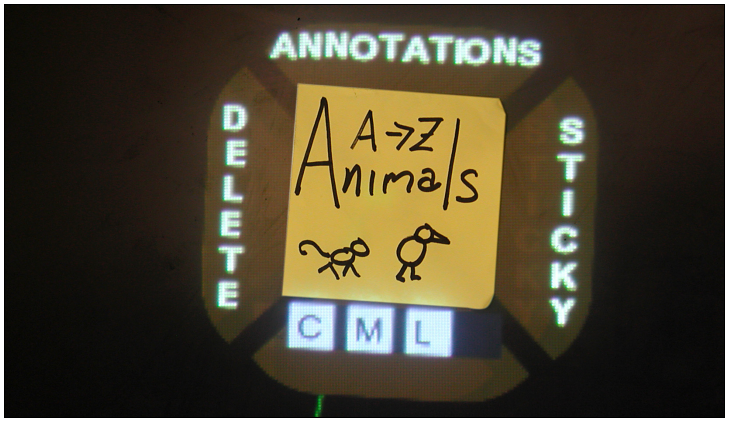


FIGURE 4.11 The electronic context menu for physical objects. The bottom element in the menu is the local timeline. In this case, the note was created ("C"), then moved ("M"), and finally a link drawn ("L"). This local timeline is display-only.

generated state set. For example, a user could begin a bookmark set with the states produced from the *By Meeting* filter, augmenting it manually with key points from the meetings. This combination of automatic and manual history echoes work by Kaasten and Greenberg [124] on managing web browsing histories.

When viewing the main timeline by bookmarks, there is a button to bring up a *synopsis view*. The synopsis view displays each of the bookmarks vertically on the left-hand side of the screen. It provides a text-box to the right of each bookmark for entering a description of that state. The synopsis view can also be printed for offline use.

4.5 History Usage Scenarios

Drawing on field studies of web design [191, 192], our previous laboratory studies (see CHAPTER 3), and on the related research literature [34, 179, 212], we have constructed four scenarios that reflect current and envisioned uses for Outpost's design history capture.

Del, Erykah, Jeru, and Rahzel are designing a portal web site for hip-hop music and culture. This portal will enable site visitors to read music reviews, interviews with

FIGURE 4.12 The on-screen synopsis view.

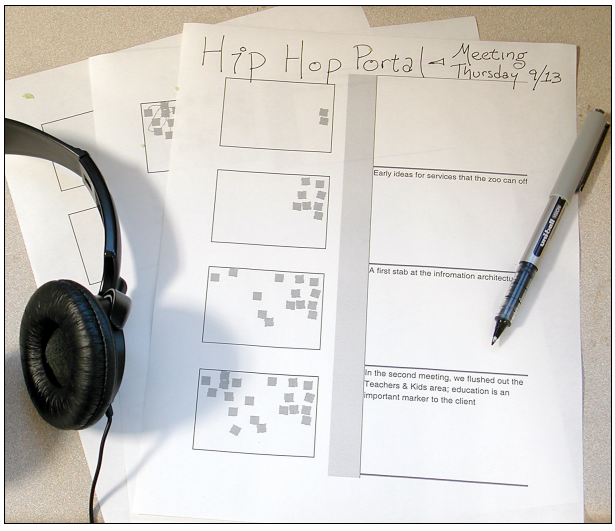


FIGURE 4.13 A print version of the same information.

artists, and relevant news stories, as well as purchase music for download, and find out about local concerts.

4.5.1 Reaching a dead-end

In the first design session, Erykah, Jeru, and Rahzel came up with a draft of an information architecture for the portal, which they are now trying to refine in a second session. After reviewing the initial design, Rahzel points out that the music reviews section is completely disconnected from the purchasing music section. If they are to make money from the site, the two should be strongly tied. After a while, Jeru concludes it isn't possible to alter the current design, so she creates an ink annotation on the main 'music review' note explaining this. She taps the 'music reviews' note to select it and then selects the by note view on the main timeline. The first thumbnail in this view displays the board state when the note was created. She taps this thumbnail to revert the board to that point in time. From there, they redesign the site so that purchasing is easily accessible from the music reviews area.

4.5.2 Writing a Session Summary

After the design session, Erykah stays in the project room. She uses the main timeline to review the team's progress. As she rolls the time forward *By Actions* she bookmarks important states in the design. Upon reaching the end of the meeting, she opens the synopsis view (see FIGURE 4.12) and annotates the key states with text. Finally, she makes a print version (see FIGURE 4.13) for herself and the other team members. This portable, sharable summary serves as an overview of what has been accomplished, and helps the team members communicate their progress to the client.

4.5.3 Find the rationale behind a decision

Del missed the design session; he was helping friends set up for a show. When he returns, viewing the electronic wall in the project room, he notices the strong linking between the music reviews and the music purchase areas. Curious why this is, he taps on the *music reviews* note, and quickly finds its local history, which he scans to understand the changes it underwent while he was away. From the note's context menu, he brings up the note's annotations. Reading the ink annotation written by one of the other designers, he quickly understands the rationale for the change.

4.5.4 Following up on a session

On Friday morning, the designers decide to perform a review of their work in the past week. Several design issues warrant further consideration; they bookmark each of these. They then annotate the bookmarks in the *Synopsis view* and print it. The print view serves as a to-do list that the designers bring back to their personal workspace. In the afternoon they reconvene and discuss the issues that each has examined, yielding a much cleaner sitemap.

4.6 Implementation

At the core of the history system is a data structure that holds command objects [86, 185]—one command object for each action carried out by the users. A command object is a software design pattern that encapsulates an atomic action as a software object called a command. These objects are stored in a command log, facilitating robust undo and redo. Some command objects in the Outpost system are *add note*, *remove note*, *move note*, *add link*, and *add ink annotation*.

These command objects are stored in a tree shaped data structure with branches. A new branch is added when the user jumps back to a previous state and then starts

modifying the board from there. The actual restoration of the board's state from a given state **x** to the user requested state **y** is handled by first calculating the *least common ancestor* **l** of **x** and **y**, then the *up-path* from **x** to **l** and finally the *down-path* from **l** to **y**. Given these paths, the state is easily restored by following the path from **x** to **l** undoing each command on the way, and then the path from **l** to **y** redoing the commands found here.

Each of the thumbnails used to visualize the command history is calculated by asking the main SATIN sheet [110] to redraw itself into a new, thumbnail-sized graphics context. The entire set of thumbnails is redrawn each time the filter changes: given a criterion, the whole tree is traversed, and the whole visualization rebuilt. It would be computationally more efficient to cache some of this information, but we have found, as others have [211], that for a research prototype rolling forward is not a substantial bottleneck. Computational efficiency has not been our focus so far and this simple approach has shown fast enough for medium-sized designs; a production implementation of this system would likely achieve faster performance by periodically caching state.

4.7 Design Study

We had six professional designers use the history system and offer their feedback (see FIGURE 4.14). When the history system was in an early state, we brought in two designers from the same firm to talk with us about their current practices and try our system. In pre-study interviews with the pair, we learned that the participants currently had a difficult time managing history; their state of the art was to save "bookmarks" and "versions" simply as files with different names. When working with our system, they primarily used the main timeline at a macro scale. Working physically and electronically occurred in cycles. They would add content for a while, work with it, then make the board electronic, and delve into the history. In addition to finding the history useful for

reflection and design rationale, the pair commented that they would find value in using the history to make accountability from the client clearer. The pair's collaborative work helped us to realize that knowing the author of content might be beneficial, leading us to implement the author wizard for the next group of participants.

With the software completed, we brought in four more designers in three groups: one pair of colleagues and two individuals. The format was similar to the study described in chapter 3. It began with an explanation of the system, continued with a design task lasting roughly an hour, and concluded with a nineteen question survey (see appendix A.2). The participants were very enthusiastic about our bookmarking features, and in the ability to generate a synopsis view.

Participants found the *View all* filter distracting, reminding us of the need for calm interaction [258]. Viewing every single command is only useful for local undo, rare during fluid brainstorming. (The one time it proved useful was when the system

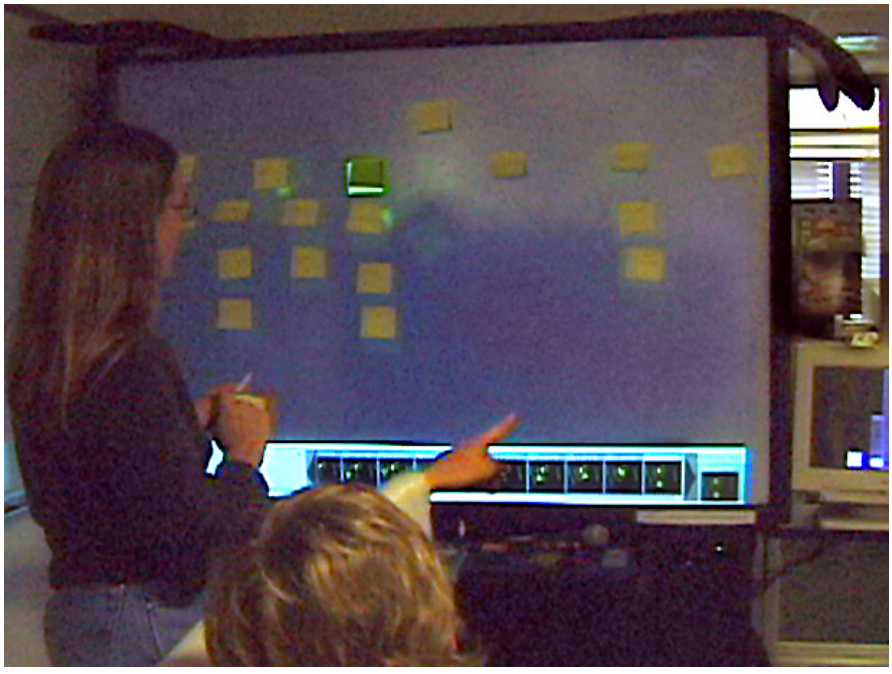


FIGURE 4.14 Two professional designers collaborate on an information architecture for the Oakland Zoo web site during the study.

misrecognized the users' input.) One finding from our previous study was that calm interaction is essential to an effective electronic whiteboard. Beyond its limited utility, *View all* is the antithesis of calm; it renders a new thumbnail for every command the user executes. This makes for a hyperactive electronic whiteboard. Based on this, we changed the default filter to be the *By Actions* filter. This provides visual locations the user can move back to at a coarser interval (the default is every six).

One participant commented that his favorite aspect of using computer-based tools was that easy saving enabled him to try new ideas and have different versions. After three of the participants had worked with the system, it became clear that save and bookmark should be integrated. We eliminated a separate save button, including the save functionality as part of the bookmarking process for the last participant. He found this integration intuitive.

4.7.1 Timeline usability

The participants were very enthusiastic about the history's ability to easily capture different states. Having a simple, touch-based visual interface with the ability to negotiate the history of the board was highly appreciated as well. The participants used the history smoothly for the most part, but sometimes, the presence of branches was confusing. As a solution, one designer suggested that sometimes it might be valuable to see the entire branch structure as a traditional graph.

4.7.2 Need for visual comparison and merging

The designers encouraged us to provide facilities for simultaneous comparison and merging of history states. One participant said, "It is very important to view multiple versions in juxtaposition, at the same time and at a scale that we can make sense out of. Much of the impact is visual." Terry *et al.*'s work on supporting simultaneous develop-

ment of alternative solutions [243, 244] provides an excellent set of UI techniques for viewing multiple alternatives simultaneously (see FIGURE 4.15). Outpost's history system would likely benefit from such techniques.

One designer commented that in his current practice, "When I'm working, I'll do the information architecture on Post-its, and draw links on the whiteboard. I'll take snapshots at different points in time. And then I'll project earlier states onto a wall, and go from there." This was a current practice uncannily similar to Outpost and its history facilities. (The other participants did not have this advanced a practice for dealing with history, possibly because such a practice is difficult with current tools.)

4.8 Summary

This chapter presented an informal history mechanism for collaborative design of information architectures that extends the Designers' Outpost. It comprises three novel history visualizations for collaborative early-phase design: a stub-branching main timeline, an *in situ* object timeline, and an annotated synopsis view.

Six professional web site designers evaluated the system. They were excited about the functionality with the exception of garish interactions like constantly updating history thumbnails, encouraging us to make calmer interactions the default, such as

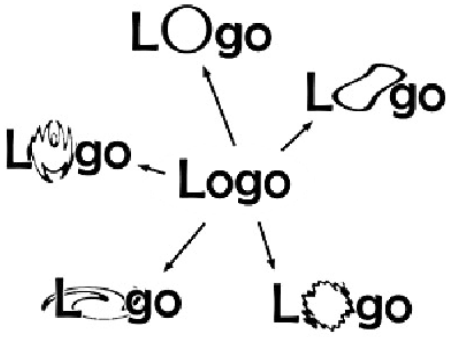


FIGURE 4.15 Creative pursuits require experimentation and exploration of possibilities, but interfaces typically stifle the ability to easily explore alternatives in parallel. Terry *et al.*'s work supports this exploration [243, 244].

manual bookmarks and infrequent auto-bookmarks.

Our design study showed that calm, lightweight history provides substantial value to designers. History-enabled design tools could benefit from heuristics that help build a concise history without distracting the user. Examples might include automatically bookmarking periods of intense work and bookmarking changes between creating content and editing. This research area could also benefit from integrating informal audio capture and access into the history system. We have found that in brainstorming sessions, the discussion among designers often captures information not expressed in the resulting visual artifact.

Many professional practices center around the creation of an artifact by several individuals over an extended period of time (writing papers is an example that comes to mind at the moment). We hope that this work on exploring interfaces for history in the context of collaborative wall-scale design will inspire work in other professional domains as well.

Through this evolution and extension of the Outpost code base, we gained an appreciation for user interface tools, and became aware of their absence for developing tangible user interface input. Where appropriate, the design history support has been added to the SATIN toolkit rather than the Outpost application. This also helped motivate our interest in toolkits in general and Papier-Mâché in particular.

5 Tangible Remote Collaboration

A tension exists between designers' comfort with physical artifacts and the need for effective remote collaboration: *physical objects live in one place*. Previous research and technologies to support remote collaboration have focused on shared electronic media. Current technologies force distributed teams to choose between the physical tools they prefer and the electronic communication mechanisms available. This chapter presents Distributed Designers' Outpost, a remote collaboration system based on The Designers' Outpost, a collaborative web site design tool that employs physical Post-it notes as interaction primitives. We extended the system for synchronous remote collaboration and introduced two awareness mechanisms: transient ink input for gestures and an outlined shadow of the remote collaborator for presence. We informally evaluated this system with six professional designers. Designers were excited by the prospect of physical remote collaboration but found some coordination challenges in the interaction with shared artifacts.

5.1 Introduction

For three decades, we have heard pundits tout the imminent arrival of the paperless office. However, paper remains a central artifact in professional work practices and use of paper is consistently increasing [222]. It is tangible, portable, readily manipulable,

---

Portions of this chapter were originally published by Katherine M. Everitt, the author, Robert Lee, and James A. Landay in [73], and by Katherine M. Everitt in [74]

and easily editable [157]. Many designers we have spoken with work in collaborative teams at multiple locations. When working with their remote colleagues, they are forced to choose between the physical tools they prefer and the electronic communication mechanisms available. The designers felt that consensus building was vital to their work process, as it establishes deep relationships, especially when participants have different backgrounds [27]. However, it is more difficult to build relationships without a sense of physical presence.

The remote collaboration system presented in this chapter (see FIGURE 5.1) extends the Designers' Outpost. This chapter discusses how structured capture enables fluid remote collaboration. To better support remote collaboration, we introduce an interaction paradigm where objects that are physical in one space are electronic in the other space, and vice versa. This paradigm has the potential to enable more fluid design among distributed teams, but must also overcome the problems of maintaining awareness between distributed groups.

We present and evaluate two mechanisms for awareness: transient ink input for gestures and a blue shadow of the remote collaborator for presence. The transient ink is a pen-based interaction technique for conveying deictic (pointing) gestures. Users mark

FIGURE 5.1 Our remote system running on two SMART Boards. Notes that are physical in one place (see left) are electronic in the other (at right). The Outpost history bar at the bottom shows previous states of the board.

up the board to suggest changes or relationships without permanently cluttering the workspace. Transient ink is displayed on both boards for a few seconds, and then fades away. The mechanism for presence awareness is a blue shadow that represents the location of the remote participants with respect to the shared workspace. Users of the system can get a sense of the locations and intentions of remote collaborators without needing their physical presence.

The Designers' Outpost was originally a single location interface. We extended Outpost to communicate between two remote hosts. The shared communication consists of user actions (*e.g.*, adding and moving notes) augmented with remote awareness information (a vision-tracked shadow of the remote users and transient ink).

5.2 Background

Our remote collaboration research draws on earlier work in media spaces for remote interaction. We now discuss this work, and also look forward to the possibilities for remote actuation.

FIGURE 5.2 Krueger's VIDEOPLACE art installation introduced vision-tracking of users' shadows. A video projector in the gallery showed the user's shadow, augmented with computational behaviors such as the creature shown on the left-hand side of the above image [138].

5.2.1 Distributed media spaces

In the mid-1970s, Krueger's VIDEOPLACE art installation introduced the use of computer vision to track a user's shadow [138, 139] (see FIGURE 5.2). Computer feedback was generated based on the user's movement in an art space. The vision system was based on chroma key technology developed for broadcast news. While the gallery was constrained to have a solid background that enabled the chroma key technique to work, this system was decades ahead of its time.

Over the last decade, there has been compelling research in distributed media spaces for visual collaboration tasks, such as shared drawing through electronic whiteboards. These researchers found, as we have, that users are interested in collaborating on design artifacts from different places. Clearboard [116] (see FIGURE 5.3) and VideoWhiteboard [242] are pair-ware systems that integrate visual drawings with video presence on a single display. Clearboard users draw on a glass board. The board is augmented with a live video projection, giving the appearance of "looking through the glass" at the remote participant's drawing, face, and upper body. The glass board and video camera setup is duplicated at each end. VideoWhiteboard works in a similar fashion, except that the video image is the shadow of a standing remote user's body.

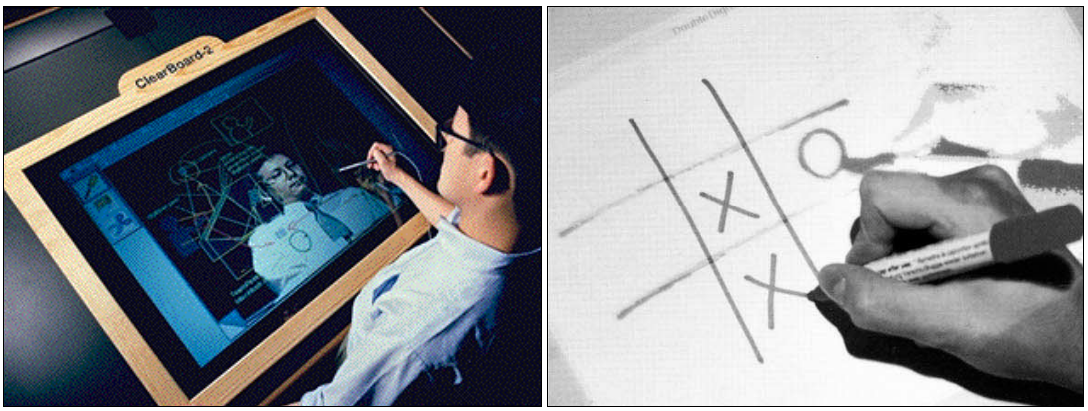


FIGURE 5.3 The Clearboard [116] *(left)* and DoubleDigitalDesk [261] *(right)* systems introduced the idea of synchronous remote collaboration through large displays.

Both of these systems use a direct video feed, cleverly aligned, to transmit both the drawing and presence information.

While the data transmission in these systems is a raw video feed, Distributed Outpost has a structured representation of the content. Its computer vision algorithms locate physical objects and users' shadows, building an internal representation of this content and awareness feedback. This semantic understanding of the information allows for more flexibility in presentation. For example, in Distributed Outpost the awareness display can be removed, modified in color, or shown as an outline only. Distributed Outpost provides more control over content changes, allowing objects to be erased or moved without affecting the rest of the display. In addition, all of the advantages carry over to Outpost's design history and ability to transition to other tools.

Our research goal is to bring together tangible user interfaces and distributed media spaces to create and evaluate an application that supports an existing design practice.

5.2.2 Remote actuation

Our work with Distributed Outpost offers tangible input locally and electronic output remotely. An alternate approach is to use actuation for physically controlling a remote space. As these technologies mature, they may become feasible for design tools like Outpost.

InTouch [41] provides an identical set of cylindrical rollers to participants at two different locations. The networked rollers behave as though they are physically connected. This system provided a shared mechanism for synchronous awareness of touch. InTouch's compelling aesthetic experience encouraged us to explore richer awareness mechanisms for our design tool.

Reznik and Canny's Universal Planar Manipulator (UPM) [210] provides a view of the future where physical objects can be controlled remotely. The UPM is a rigid,

horizontal plate, which vibrates in its own plane and moves generic objects placed on it because of friction. However, the technology is not yet mature enough to support large numbers of objects, and our system is based on vertical as opposed to horizontal surfaces.

5.3 Interviews and Fieldwork Informing Design

Previous fieldwork and design studies [34, 191] have found that designers often need to collaborate with colleagues and clients who are not in the same office or even the same city. We brought six professional designers into our laboratory to provide feedback on Distributed Outpost. We first asked them to discuss their current remote collaboration practices. The designers described several important collaboration tasks including: consensus building, concept mapping, user focused design solutions, and defining project features, function, and interaction.

5.3.1 Current experiences with remote collaboration

Working with remote participants is a "nightmare," stated one designer. The designers expressed three primary frustrations with their current collaboration tools. First, they felt that their interactions with remote colleagues were impoverished. Second, they felt the tools well suited to collaboration (*e.g.*, email, telephone), were ill suited to design. Finally, all of the designers in our study had developed ad hoc methods when designing with remote colleagues.

The study participants reported four different methods for working with remote collaborators.

1 *Whiteboard, video, and email:* One group maintained their physical practice of using a whiteboard with sticky notes at a central office. Remote participants can view the screen though a video link, however, their participation is severely limited. Distributed workers

send email to the facilitator when they have input. Thus, they are very reliant on the facilitator for their participation in the design session, and there is a time lag between their contribution and its visibility to the rest of the group. As a group member stated, "This makes it almost impossible to have active participation of remote participants."

2 *Two whiteboards and videoconference:* Occasionally both offices will have sophisticated videoconferencing technology. Designers work on two separate, manually synchronized whiteboards with Post-it notes. Each side has a remote controlled pan/tilt/zoom camera. The technology is adequate for viewing the distributed boards, and the resolution is high enough to view written text. However, there are significant pauses in the interaction while one side zooms the camera in to see a change, and there is trouble keeping the separate representations consistent.

3 *Collocated meetings (and occasional conference calls):* Another group was limited to only generating ideas when they were collocated. Once the ideas were generated, the potential design was typed into a computer for sharing with the remote clients. When meeting with clients in a conference call, each person had their own paper printouts on which they recorded potential changes to the design. Later, these designs were synchronized by the designers in a discussion meeting to come up with the final design.

4 *Visio and email:* Another participant developed designs alone with Microsoft Visio, a graphical diagramming tool. When it came time to collaborate, he would email the document to another user, who would change it and email it back. Some of his colleagues did not have this tool and thus worked on paper printouts and had him enter the changes into his document. This setup made real time collaboration impossible and added significant lag to the design process.

5.3.2 User needs for remote collaboration

One of the largest problems we identified was a lack of a shared workspace. For large, remote teams, it can be hard to maintain focus without a shared artifact to discuss. It is also difficult for remote users to gesture or convey spatial relationships when they do not have access to the items under discussion. The formality and constraints of current technologies also interrupt the flow, making design more difficult [145].

Many designers stressed the importance of establishing common ground with the people they worked with. "It's not the end, it's the means," one designer explained. Consensus is vital for moving forward in the project. When the participants have different backgrounds, it becomes especially important to establish deeper relationships.

The designers we interviewed found it difficult to establish a rapport with distributed participants. They said they felt disjointed from their peers working remotely. This remains a problem even with a sophisticated video conferencing setup. Latency, a lack of presence information, and out of sync artifacts remain barriers to effective collaboration.

5.4 Interaction Techniques

Our system addresses designers' needs in two ways. We provide a unified workspace with support for spatial gestures between remote colleagues. We also provide presence and awareness mechanisms to help remote participants establish common ground. In supporting these requirements, we felt it was important to keep the physical interaction and maintain a calm interface, such as in AROMA [202, 203].

5.4.1 Shared workspaces and transactional consistency

Our system consists of a shared workspace though which groups of designers can interact. Several designers can participate at once when working with the board. The computer vision system supports simultaneous input of several Post-it notes.

A note is created by writing on a physical Post-it note and placing it on the board. When a local user *physically adds* a note to the whiteboard, the remote system *electronically displays* a photograph of that object (see figure 5.4 and figure 5.5, top row). The vision system's rear camera locates the note and the front camera takes the photograph (see figure 3.20). The front camera very rarely has problems with users occluding note pictures (see section 3.8.2). When any user performs an action, both the local and the remote system are updated. Both teams can interact with any note, regardless of whether it exists as a physical object or remote analogue.

To delete a note, the user simply removes it from the board (see figure 5.4, middle row). To move a note, the user picks it up and places it in the new position (see figure 5.4, bottom row). Currently, the system does not recognize specific notes based on content and so it assumes that the note is the same if it is replaced within seven seconds (see section 3.4).

We would like both teams to be able to edit and move all objects. When the objects are electronic (such as with links), this is easily facilitated. When the objects are physical (such as with Post-it notes), editing them from multiple sites introduces some difficulty. One option is to only allow the creator editing ability [180]; that is not very appealing.

We have taken an alternate approach. Post-it notes in Outpost cast electronic shadows as feedback to the user that the system is aware of their presence. When a note's physical state becomes transactionally inconsistent, the system casts a strong red shadow indicating to the user to remove the artifact (see figure 5.5d). The red shadow

identifies that the physical note is no longer an information handle to the virtual remote analogue. This feedback is lightweight; it provides awareness that the note is out of date, but does not require the user take any action.

We originally introduced the red shadow feedback in Outpost's design history system (see CHAPTER 4). There, it identifies notes that are out of date with respect to time. Here, it identifies notes that are inconsistent with the remote users' board.

There are two user actions that make a physical note transactionally inconsistent: deleting and moving. If the note is deleted by the remote user, the faint recognition shadow is replaced with a red shadow (see FIGURE 5.5D). The local user could remove the note to dismiss the shadow or re-post the note if they disagreed with its removal.

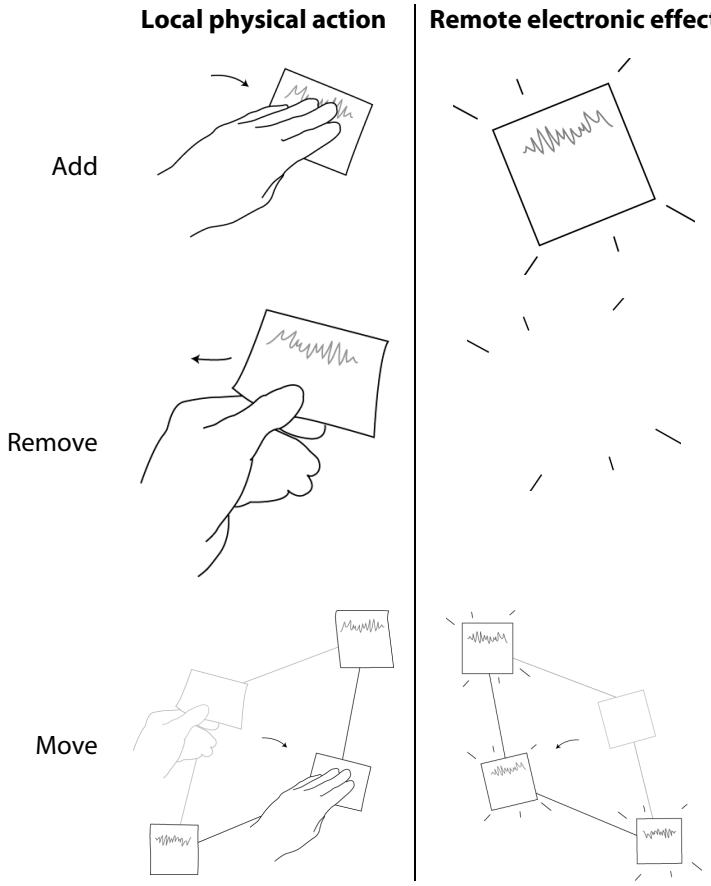
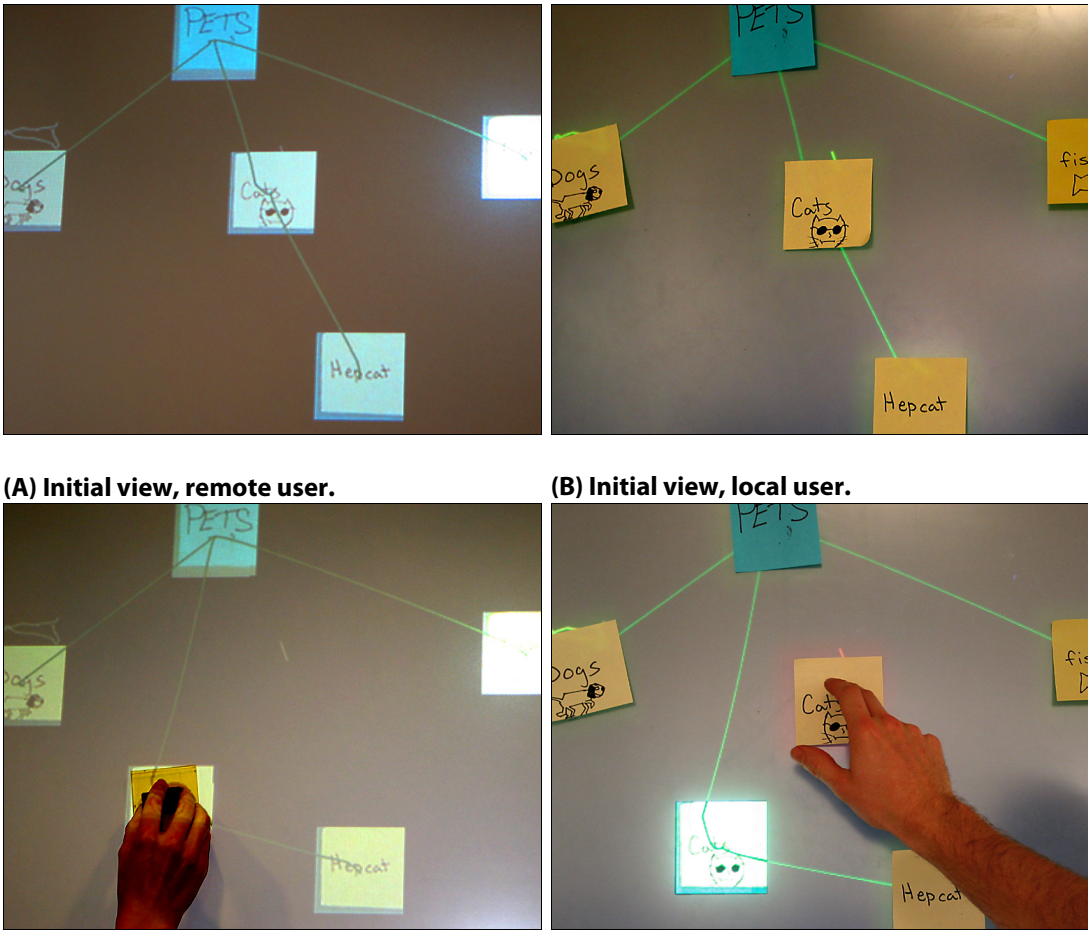


FIGURE 5.4 Interaction techniques for creating, deleting, and moving physical notes in Remote Outpost. The left column is the user's action with the physical note; the right column shows the electronic display on the remote board.

When a note is moved, a red shadow displays behind the out-of-date physical note and a virtual note appears in the new position (see figure 5.5). The local user could then remove any physical note with a red shadow.

When a note is virtual, the physical handles are missing, and must be replaced with electronic controls (see figure 5.6). In this case, a note context menu is available for deleting notes, and the physical move tool is available for moving the notes as described in section 3.4.



(A) Initial view, remote user. **(B) Initial view, local user.**

(C) Remote user moves note. **(D) Local user acknowledges move.**

FIGURE 5.5 Moving a note: A and B show the remote and local views before the move. In C, a remote user moves the electronic version of the 'Cats' note with the move tool. D shows the virtual 'Cats' note at the new location and the local user removing the out of date physical 'Cats' note (marked with a red shadow).

5.4.2 Desktop Outpost

Distributed Outpost works on any hardware with a 2D input device, *e.g.,* a SMART Board, a digital desk, a tablet PC, or an ordinary desktop PC. While users benefit from the computer vision tracking and capture of physical objects, it is possible to create notes with a tap and draw on them with the stylus tool. Links can be added by drawing a line between two notes. Erasing and moving ink and links is supported with the stylus button.

Although the setup is not ideal and the tangible advantages of Outpost are not available, users can still work with the notes using pen-based design interaction. This setup is more cost effective and flexible for remote participants with limited resources.

5.4.3 Transient ink for deictic gestures

An important affordance of remote collaboration systems is the ability to convey deictic gestures. Without this, it is difficult for users to understand what their remote colleagues are communicating or to express their opinions on relationships.

When users want to draw their collaborators' attention to a particular spatial position or artifact, they need some way to convey this deictic gesture. We found that a

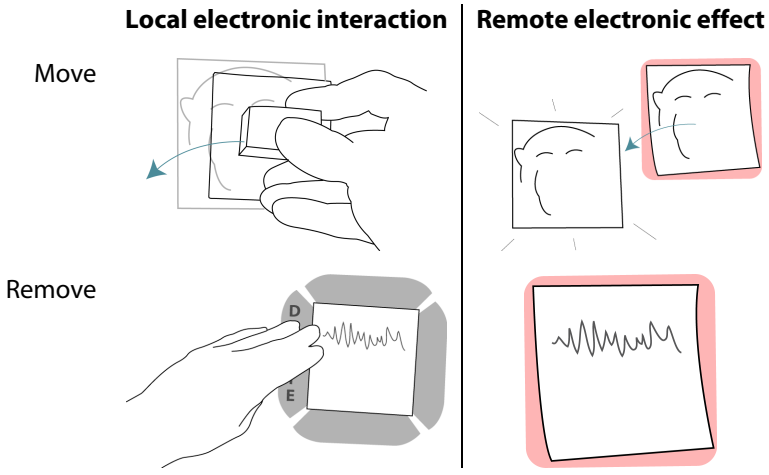


FIGURE 5.6 Interaction techniques for moving (*top*) and deleting (*bottom*) electronic content.

simple remote pointer did not convey enough information. For this reason, we developed transient ink as a richer interaction technique for enabling distributed users to convey deictic information to each other. This is similar to the technique described in [245], which conveyed a transient pointer to a remote site. However, our technique has richer interaction. It conveys an ink stroke rather than a pointer, and it allows multiple simultaneous strokes.

Users draw electronic transient ink on the board with the red stylus tool (see figure 5.7). The ink is rendered on both displays for a few seconds, and then it fades away. This allows users to convey relationships, suggest links, and point to notes without committing their changes and permanently cluttering the board.

When using the board interface, a specific stylus is used to create transient ink instead of regular ink and links. From the desktop setup, it must be selected from a pie menu. In addition to transient ink as a mechanism for conveying gesture, we use computer vision techniques to provide stylized shadows of people to help provide a rough idea of remote collaborators' locations around the board.

5.4.4 Distributed presence

A sense of presence is important to developing a working relationship with remote colleagues. However, the designers we interviewed did not feel that the currently

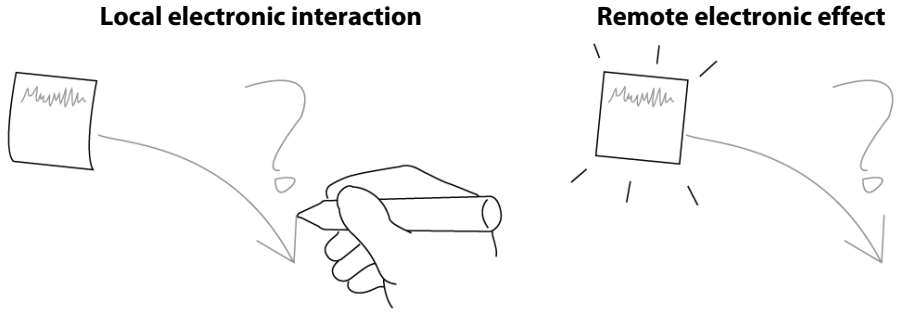


FIGURE 5.7 "Should *this* note be moved down *here*?" Transient ink is used to convey pointing information and temporary graphical material by a remote user. The written ink fades away after several seconds. The writer's view is on the left; the receiver's view is on the right.

available videoconferencing and audioconferencing technologies provide a sufficient sense of presence to establish a rapport.

Our presence shadow is inspired by Clearboard [116] and VideoWhiteboard [242]. The seamless interaction paradigm put forth in these systems is particularly appropriate to support awareness for our system. It is important that the presence mechanism be calm and non-distracting, allowing designers to focus on the task.

We extended the rear camera's vision processing, used for detecting notes, to detect peoples' shadows on the board (see figure 5.8). As a person casts a shadow on the board, we determine if it is the appropriate size and darkness for a person. If so, the vision system calculates the shadow boundary. If more than one person is working at the board, the awareness will show multiple shadows.

An early version of the remote awareness, used for the feedback session (see section 5.5), displayed a translucent blue oval based on the center point, width, and height of the detected shadow. We implemented the feedback in this manner because it was the simplest from an implementation standpoint. However, not surprisingly, our study

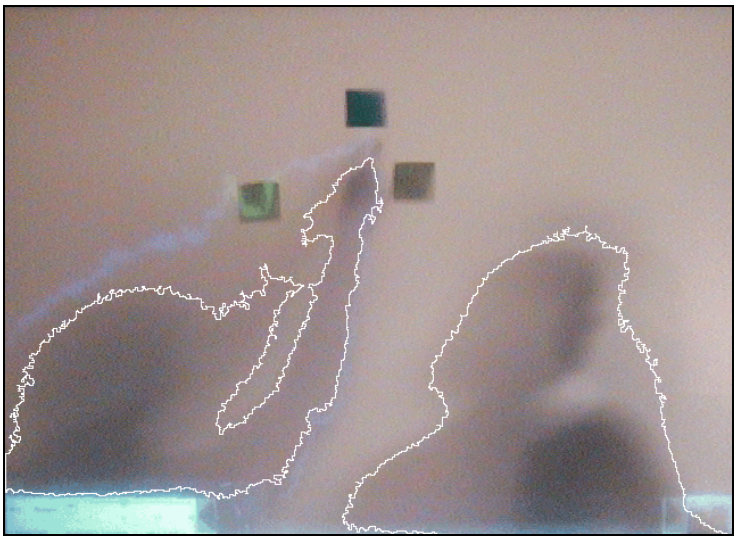


FIGURE 5.8 The view from the rear camera of two users, one of whom is pointing to a note on the board. The calculated borders of the shadows are drawn in white, on top of the raw pixel input.

participants found that this did not provide enough detail.

The current presence visualization is a stylized shadow outline of the remote users, displayed on the background of the design surface (see figure 5.9). This shadow conveys the remote users' presence, gesture, and location in a lightweight fashion.

All content and presence information is sent using sockets over ip, unlike prior work [116, 242], which required a dedicated video link. We present a shadow instead of a live video image of the user because it is calmer; live video is too distracting when interacting with whiteboard content. Our goal was to subtly display and communicate information that is not part of the user's primary foreground task.

5.5 Software Infrastructure

Our remote collaboration system extends the Designers' Outpost. Two sets of additions were required. First, we extended the command object system to replicate state across a

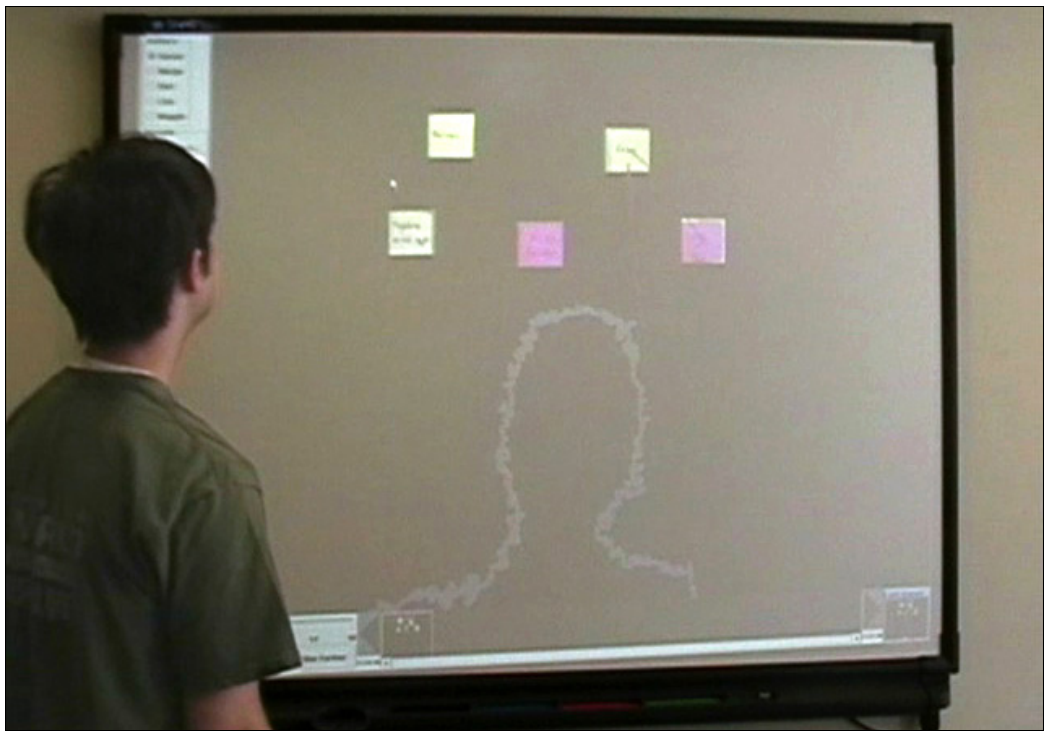


FIGURE 5.9 The distributed awareness mechanism. A blue shadow outline in the background represents a remote collaborator.

network for synchronizing shared applications in a manner similar to Berlage and Genau [36]. Second, we added functionality to the vision system to support distributed collaboration.

5.5.1 Data transfer

The system works as a peer-to-peer system; both endpoints replicate their commands, sending the corresponding command objects to the opposite endpoint. Each object has a unique global identification tuple, composed of its creator's hostname and an integer corresponding to its position in the local command queue. This identifier is used to refer to objects between hosts. We modified the SATIN command queue so that when a command is executed, it is also marshaled for serialization over the wire, and sent to the remote host. Because most of the changes were made at the toolkit level, other SATIN-based applications can benefit from this infrastructure with minimal application-level change.

Outpost designs are serialized to files as XML documents. We use the same XML serialization scheme for network communication. The connection between the machines is socket based. Users have the option of connecting to different remote hosts, or not connecting at all if they wish to work alone.

At present, photographs of notes are stored as JPEG files on a networked file server. They are accessed as needed over the file system by the hosts. This could easily be modified to use a web server to support collaboration between distinct organizations.

5.5.2 Vision and tracking

Our vision system is written in C++ on top of OpenCV [40], a highly optimized library of computer vision and image processing primitives. The original Outpost vision

tracking system used the rear camera to track notes (see SECTION 3.8). We extended this to find shadows of people using the system.

The system processes the image using spatial and temporal filtering, corrects for perspective distortion, and computes a running average of the expected background image.

We construct three thresholded difference images. Possible pixels from added notes are found by subtracting the current frame from the expected average image. Potential shadows are found the same way. They have a lower threshold than for notes, because a person's shadow cast from standing in front of the board is not as dark as the shadow cast by a note stuck directly onto the board. Potential removed notes are found by subtracting the expected average image from the current frame.

At this point, we segment the binary images using a connected-components algorithm. The found elements expected to be notes are subjected to size and shape restrictions using an expectation maximization algorithm before being classified as notes. The person objects are also subjected to size restrictions of 0.5% to 40% of the board.

The vision system runs as a separate process, passing events (*e.g.*, *add*[x, y, θ, ID], *remove*[x, y], and *addPerson*[x, y, w, h]) to the local Outpost UI through a socket network connection.

5.6 User Feedback

We had six professional designers visit our lab. We asked them to come prepared to design a web site of their own choosing. We had one group of three, one group of two, and one single user come in and use the system. Each session lasted 1.5 to 2.5 hours. First, we orally interviewed them about their current remote collaboration practices. Then, we introduced the remote system and had them use the system to design their

site. Designers were generally enthusiastic about the system's potential to improve their work.

Due to the technical constraint of having only one full Outpost board setup, the groups worked with one Outpost Board connected to a VisionMaker digital desk. The input for the digital desk was an input-only Wacom Graphire pen tablet. One of the participants used a mouse instead of a Wacom Tablet. Although we recognize the importance of audio to distributed collaboration, the current implementation has no audio support. This is not a major drawback, as a conference call can easily provide multi-user audio support. For our feedback sessions, the board and the desk were located in the same room. The participants were allowed to speak to each other but unable to see each other because of a curtain.

During the sessions, users input ideas using physical Post-it notes for concepts and styluses for linking and annotation. They were able to access transient ink by using a specific stylus tool. The rear camera tracked their shadow location. The one difference between our study implementation and the final implementation is that, in the study, the shadow awareness was implemented as a simple oval, rather than a true shadow. While clearly not optimal, we chose to conduct the study with this partial prototype to rapidly garner feedback. The shadow transmission was also unidirectional: it was only transmitted from the board (with cameras) to the desk (without cameras).

The Digital Desk side had no cameras, and thus was set up to run Desktop Outpost. The digital desk users were seated in front of the slanted desk. Users could input notes and write on them using the Wacom stylus. Transient ink was available to them as an option through a pie menu accessed with a right click.

5.6.1 Qualitative feedback

The users were very enthusiastic about the shared workspace. They felt that it would increase the value of working sessions with team members and clients. They appreciated seeing their colleagues' input in real time. They felt it improved their collaboration, as spatial relationships were visible in real time to everyone. One designer mentioned that she preferred Distributed Outpost to the whiteboard and videoconferencing setup because Outpost digitizes the information for later use, and there is no pause in the work for zooming and panning of videoconferencing cameras. They liked the flexibility of the notes, and the ability to collaborate and throw out ideas quickly.

Users also liked the concept of transient ink. One designer especially liked this concept because he could show relationships between elements without committing to the interaction. Designers found that Outpost's functionality made it easy to make changes and communicate their intent to others. About half of the participants found the transient ink useful; the others did not use it during the test. As one user commented, one may as well make marks with ordinary ink and then erase them. However, one of the participants rated the transient ink as being more important in remote collaboration than voice.

Half of the users found the presence awareness shadow compelling. They felt it was vital to provide a frame of reference for the remote participant. They could thus refer to data objects with an understanding of how the remote person viewed them. They felt this gave a better understanding of participation from the remote site.

5.6.2 Areas for improvement

Although the users seemed generally enthusiastic about the potential of the system, there were some coordination problems. With a physical Post-it note, it is clear when two co-located people wish to move or edit the artifact at the same time. With

Distributed Outpost, there are no restrictions on who can change or edit notes. In our study, there were times when users at both ends edited the same element because they were working in the same area of the board. However, these conflicts were infrequent and easily corrected.

Even though the remote shadow was designed to be unobtrusive, some designers found it a bit jumpy and distracting. They requested feedback with smoother motion that provided more human characteristics, such as showing hesitation and acceleration: "things that one can translate into feelings." They also wanted more detail than the oval shadow provided; we have since implemented an outline shadow that provides this detail and smoothness.

Overall, the designers we interviewed were enthused by our system and felt the concepts would be helpful in increasing the interactivity of their remote design collaboration. This study also highlighted the need for integrated audio communication; this is an important area for future research in design tools for distributed collaboration.

5.7 Summary

We have presented Distributed Designers' Outpost, a remote collaboration system supporting designers' need for both physical artifacts and distributed collaboration. Our remote system provides a shared workspace where the participants can edit any object, regardless of where it was created. We presented two novel awareness mechanisms: transient ink input for gestures and a vision-tracked stylized shadow for presence. Six professional designers provided feedback about the system, and were enthusiastic about its potential to support their current practices and increase their ability to work in distributed teams.

Computers have been instrumental in allowing us to communicate quickly with people all over the world. However, we lose some of the advantages of meeting face to face. Hopefully, this work will help to bridge the gap between the virtual and physical worlds and help remote teams to work more comfortably and effectively.

The difficulties involved in extending the vision tracking system in Outpost encouraged us towards the more flexible architecture that Papier-Mâché provides. With Papier-Mâché, extending a vision system for capturing shadows is only a few lines of code. Additionally, all of the command object extensions for remote collaboration were implemented in the SATIN toolkit. Any application that uses SATIN for managing commands can automatically take advantage of the remote collaboration features described in this chapter.

6 Books with Voices

> *"If the book had been invented after the laptop it would be hailed as a great breakthrough. It's not technophobic to prefer to read a book; it's entirely sensible. The future of computing is back in a book."*
>
> —Neil Gershenfeld, 1999 [88, p. 14].

The second project in this dissertation research on tangible user interface input is Books with Voices. Our contextual inquiry into the practices of oral historians unearthed a curious incongruity. While oral historians consider interview recordings a central historical artifact, these recordings sit unused after a written transcript is produced. We hypothesized that this is largely because books are more usable than recordings. Therefore, we created Books with Voices, which provides barcode-augmented paper transcripts for fast, random access to digital video interviews on a PDA. We present quantitative results of an evaluation of this tangible interface with 13 participants. They found this lightweight, structured access to original recordings to offer substantial benefits with minimal overhead. Oral historians found a level of emotion in the video not available in the printed transcript. The video also helped readers clarify the text and observe nonverbal cues.

Portions of this chapter were originally published by the author, Jamey Graham, Gregory J. Wolff, and James A. Landay in [132]

6.1 Introduction

"Oral history is primary source material obtained by recording the spoken words—generally by means of planned, tape-recorded interviews—of persons deemed to harbor hitherto unavailable information worth preserving" [235, p. 40]. The discipline began in 1948 when Allan Nevins founded the Columbia University Oral History Research Office.

Our experiences reading the work of oral historians reflecting on practice, doing contextual inquiry with oral historians, and conducting oral histories ourselves led us to develop Books with Voices: barcode-augmented paper transcripts enabling fast, random access to digital video interviews on a PDA. Members of the Regional Oral History Office (ROHO) at UC Berkeley [2] participated in our design process. We showed nine different mock-ups of barcode-augmented oral histories to twelve members of ROHO. They were excited to hear as they read and encouraged us to make our paper book design more usable.

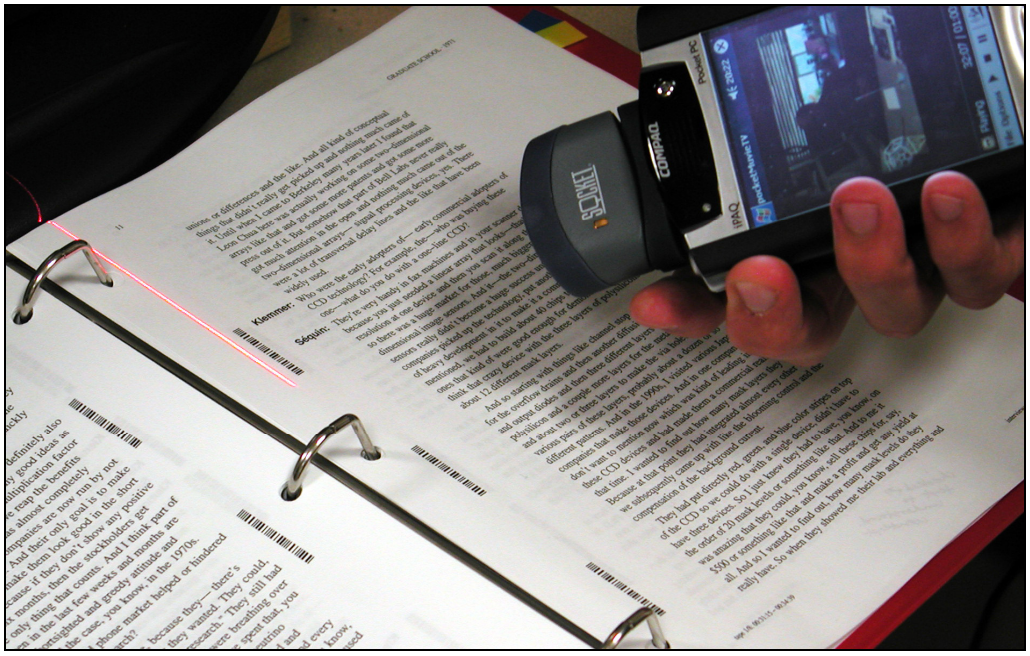


FIGURE 6.1 Accessing digital video by scanning transcripts.

We evaluated Books with Voices (see figure 6.1 and figure 6.2) with 13 users: eight oral historians from roho, one oral historian from the University of San Francisco, and four members of the UC Berkeley Computer Science Graduate Student Book Club. The response from participants was overwhelmingly positive. The calm interface allows users to concentrate on the task and stay in the flow. Participants repeatedly accessed recordings while reading. They did so to: a) get a sense of the personality of the interviewee, b) hear the tone of a particularly compelling passage, and c) verify the accuracy of the transcript. We found several fixable usability issues, most notably that the pda must be rotated into an awkward vertical position to scan. Motivated partially by our system, roho has begun transcribing their oral histories digitally.

6.2 Fieldwork into Oral Histories

We undertook a three-pronged approach to better understanding the discipline of oral history. First, we read oral historians' reflections on practice (*e.g.*, [11, 68, 89, 235]). Second, we conducted a contextual inquiry at roho. Third, we experienced the process by conducting oral histories with two well-known computer science professors.

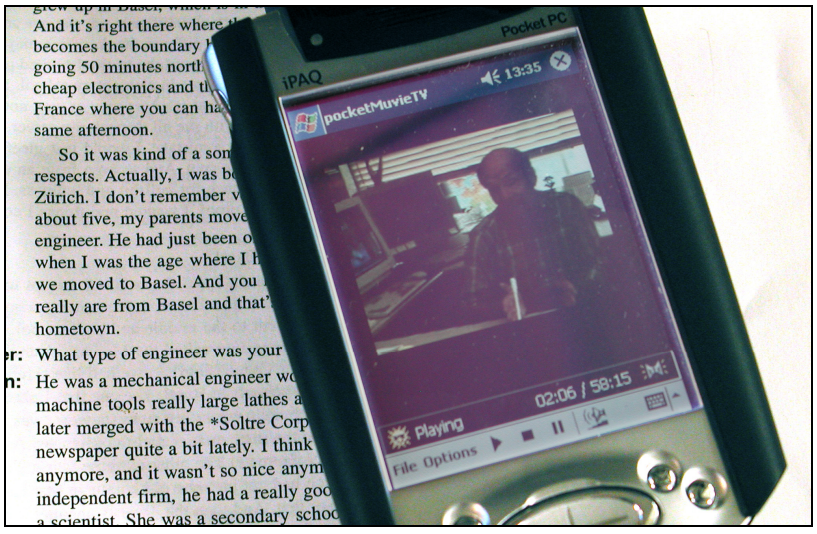


FIGURE 6.2 PDA playing a video of a recorded oral history.

Through our fieldwork and literature review, we discovered a curious incongruity. While oral historians consider the audio or video recording a central historical artifact, these recordings sit unused after a written transcript is produced. Louis Starr wrote about why transcripts dominate use:

> "This is not so much because those who favor the transcript have the better of the argument of theoretical grounds as because of practical convenience: to most researchers, a written document that carries page numbers, and an index to them, is vastly preferable. Tapes, no matter how carefully indexed, are awkward to use, particularly if the memoir is a massive one. … A consensus emerges: tapes are more suitable for some purposes, transcripts for others; but so far as possible both should be preserved, allowing researchers to choose for themselves" [235, p. 43].

Mackay, in three research projects spanning several years, also found people's preference for paper. Summarizing this work, she writes:

> "Contrary to what many believe, users are not Luddites, clinging to paper as a way of resisting change. On the contrary: most are excited by the benefits offered by computers… Their resistance is, in fact, extremely practical. New computer systems are either less efficient or simply cannot perform many required tasks" [157, p. 81].

Suchman writes about how participants understand conversation, stating "contextualization cues by which people produce the mutual intelligibility of their interaction consist in the systematic organization of speech prosody, body position and gesture, gaze, and the precision of collaboratively accomplished timing" [239, p. 72]. These contextualization cues are more available in a video recording than a written transcript.

The written transcripts of these interviews are a wonderful artifact. However, they lack a humanity that is available in the videos through the interviewee's body language and prosody.

6.2.1 Contextual inquiry at ROHO

In our research, we spent time at ROHO, the third oldest oral history office, and one of the largest. Interviewers are generally knowledgeable about the area in which they interview. For example, ROHO is documenting the "Disability Rights and Independent Living Movement" (DRILM) [6]. One member told us, "All of the project interviewers have personal experience with disability." In other cases, the interviewer is well versed in the history of the area but is not a direct participant. Interviewers generally prepare with two to three weeks of background reading on the subject area and on the writings by or about the interviewee.

A typical oral history consists of between four and twenty hours of oral interviews. A rule of thumb is that each interview session should last roughly an hour and a half. Some historians bring a still camera to photograph the interviewee or objects important to them. Afterwards, interviewees are generally sent a preliminary transcript for review.

Isolated interviews are rare. Usually, a small group works together to create a set of 12 to 20 interviews about a subject. With DRILM, seven interviewers and two managers conducted oral histories with more than forty interviewees.

Oral historians in general are highly enthusiastic about video and other digital technologies. Traditionally, interviews have been recorded on analog audiocassette. At ROHO, digital MiniDisc is rapidly replacing cassettes, and we anticipate that digital audio recorders (*e.g.*, [13]) will soon replace MiniDisc as the *de facto* standard for audio recording of interviews. Digital video is gaining popularity in the oral history community at large (as judged by traffic on the main oral history mailing list), and ROHO has begun using it in their work.

The main barrier to adoption is that few historians feel they have the time to learn a new technology. One particularly compelling aspect of our system is that it augments, rather than replaces paper. Books with Voices paper transcripts offer the same familiar

affordances as current transcripts because both *are* ordinary paper. The user population is not required to change how they read. As users recognize the benefits of our system, they *can* adopt it, but they are not *forced* to give up their current practices.

6.2.2 Conducting oral histories

To better understand oral history practice, the author attended an oral history training workshop, and then conducted oral histories with UC Berkeley computer science professors David Patterson and Carlo Séquin. This comprised three interviews with Prof. Patterson (four hours and 77 pages worth) and four interviews with Prof. Séquin (six hours and 114 pages worth). These interviews were recorded on digital video and converted to MPEG-2. The digital files were then professionally transcribed with time-stamps corresponding to each utterance.

The next section offers an overview of research on oral histories and audio retrieval. We then discuss the paper prototype (SECTION 6.4) and the interactive system (SECTION 6.5). SECTION 6.6 describes the evaluation of the interactive system with oral historians. The chapter closes with a summary in SECTION 6.7.

6.3 Background

We now discuss related research on oral histories, and work on reading and video retrieval.

6.3.1 Technology support for oral histories

The topic of the May 2002 issue of *Forum: Qualitative Social Research* [89], an online journal, is "Using Technology in the Qualitative Research Process." It offers an excellent overview of current work in the field and shows social scientists' strong interest in using technology in their work practices.

Palaver Tree [71] is web-based educational software enabling middle school students to conduct email oral histories with elders (such as civil rights leaders). This work inspired us to research oral histories. While Palaver Tree's contribution is primarily pedagogical, our contribution is primarily technological.

Steven Spielberg's *Survivors of the Shoah Visual History Foundation* has "collected more than 50,000 eyewitness testimonies in 57 countries and 32 languages" from Holocaust survivors [233]—a huge collection. Researchers at the University of Maryland are addressing the information retrieval issues in this archive (particularly multilingual access), and have conducted ethnographic studies detailing the needs of researchers using this archive [102]. Many other oral history web sites have some of their oral histories online; [43] has an excellent list in the appendix.

6.3.2 Reading, listening, and video browsing

One of the most compelling electronic book systems is XLibris [219]. XLibris, and e-books in general [220], enable more rapid searching of text and more fluid sharing of annotations. However, "there is a tension between the advantages provided by computation and the advantages provided by paper: the choice depends on the reader's goals" [219, p. 249]. We chose paper over e-books for our domain because, "People clearly prefer reading on paper to reading on their PCs" [220, p. 65].

Audio Notebook [237] and Dynomite [263] inspired our interest in using text as an interface to streamed media. Audio Notebook is a paper notebook that sits on top of an ink and audio capture device. As note-takers write, ink is time-associated with audio recorded at that time. Dynomite offers similar functionality, with an electronic notebook as opposed to a paper notebook. These projects compellingly show the use of one modality (written notes) as a query interface to another modality (recorded audio).

One aspect of our work is providing an interface to pre-recorded video. In an evaluation of a digital video browsing system, the authors found that "the most frequently used features were time compression, pause removal, and navigation using shot boundaries" [149, p. 169]. While these interaction techniques become less important in the presence of a full transcript, it would certainly benefit our system to incorporate them.

6.4 Paper Prototype of a Paper Interface

In our design, barcodes augment the paper transcript, providing a physical affordance for fast, random access to digital video recordings. As our first design step, we produced nine different paper mock-ups, brainstorming the visual design of these augmented-paper reading artifacts. We based these mock-ups on the print format that ROHO currently uses, adding barcodes and video stills to the sides of the text and metadata to the header and footer (oral history title, interview date, and time-code information).

We showed these mock-ups to twelve members of ROHO: the director, the head transcriber, several interviewers, the technology director, and two historians that use but do not produce oral histories. People spoke passionately about the importance of hearing the original voice. (With one exception: One woman felt that she was a reader only, and not a listener.) This echoes the feelings of other oral historians such as this H-OralHist list posting, "Each speaker's voice is so distinctive! I've found that sometimes even when two speakers seem to be saying something very similar, their intonations can indicate subtle differences in meaning that can complement each other. … The unedited tapes are much more lively and interesting although of course it takes much longer to listen to them than to skim transcripts" [213]. The participants appreciated the metadata as well.

The participants encouraged us to address two issues in the next iteration:

1 For this domain, people did not want many video stills in the printed book. These "talking head" stills are nearly identical; repeated printing would offer little value. One person suggested that one video still at the beginning of each section would be about right, and others agreed that this made sense. People also encouraged us to incorporate photographs and other media.

2 People really liked the barcode access, and about three barcodes per page seemed to make sense. In the interest of "clean" visual design, most of our mock-ups had barcodes that were evenly spaced on the page (*e.g.*, top, middle, and bottom). The users did not like this; they specifically asked that barcodes be visually aligned at speaker turn and paragraph boundaries (see FIGURE 6.3 and FIGURE 6.4).

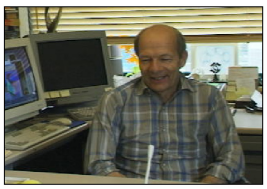
6.5 Interactive Prototype

We used the feedback we received on the mock-ups to build an interactive version of Books with Voices, based on the Video Paper software [95]. Video Paper is a paper-based interface for browsing, retrieving, and viewing pre-recorded video. The Books with Voices document production pipeline (see FIGURE 6.5) is based on Video Paper, and comprises three phases:

1 Creating an MPEG-2 video (at 20 FPS, 208 × 160 pixels) and making JPEG thumbnails from a video source.

2 Creating a paper layout (see FIGURE 6.3) from a time-stamped transcript. We modified the Video Paper rendering engine to produce a document more suitable for our domain.

3 Pocket PC software that reads the barcode and plays the corresponding video.



Séquin: I'm Carlo Heinrich Séquin. I'm Swiss. I have an Italian first name. I have a German middle name, and I have a French last name. And now I'm in a country where they have trouble pronouncing either one of them correctly. You could almost joke that my parents were politically correct because at the time when I was born, those three languages were indeed the three official languages in Switzerland so I represent each one of them. I grew up in Basel, which is in the German part of Switzerland. And it's right there where the Rhine leaves Switzerland. It becomes the boundary between Germany and France. So going 50 minutes north gets us to Germany, where we can buy cheap electronics and then within 50 minutes west gets us to France where you can have a nice dinner and you can be back the same afternoon.

So it was kind of a somewhat international haven in many respects. Actually, I was born in Winterthur in the canton of Zürich. I don't remember very much of that because when I was about five, my parents moved to Basel. My father was an engineer. He had just been offered a good job in Basel so just when I was the age where I had to go to kindergarten, that's when we moved to Basel. So most of my memories really are from Basel and that's what I consider my real hometown.

Klemmer: What type of engineer was your father?

Séquin: He was a mechanical engineer working with a firm that made machine tools, really large lathes and milling machines, and it later merged with the Soltre Corporation which has been, in the newspaper quite a bit lately. I think things didn't go so well anymore, and it wasn't so nice anymore. When it was an independent firm, he had a really good time. My mother was also a scientist. She was a secondary school teacher. And I have one brother. He's two years younger. And I guess we both grew up in a very much science– oriented environment. So reason was always accepted as the highest cause and we're never too much into deep philosophy and religion.

Klemmer: Were your parents religious at all?

Séquin: Not really, no. And I think I went through the motion in school. I was a Protestant on the most liberal side. I went through confirmation, but I voiced serious doubts at that time. And the priest gave us the instruction to try as hard as he could, to kind of convince me there's about 10 different proofs of the existence of God, except every proof had a tiny little fault. So it's like, in his words, "Only like 95 percent correct, but since we have 10 of them, that's way more than 100 percent" And somehow that argument didn't convince me then, neither does it convince me today. I guess at this point I have to admit that I probably always was somewhat of a smart ass and a loud mouth.

It probably was only the second or third class in elementary school when our teacher was telling us a story of Robinson

FIGURE 6.3 The first page of Carlo Séquin's Books with Voices transcript. The barcodes are aligned with speaker changes and paragraph boundaries.

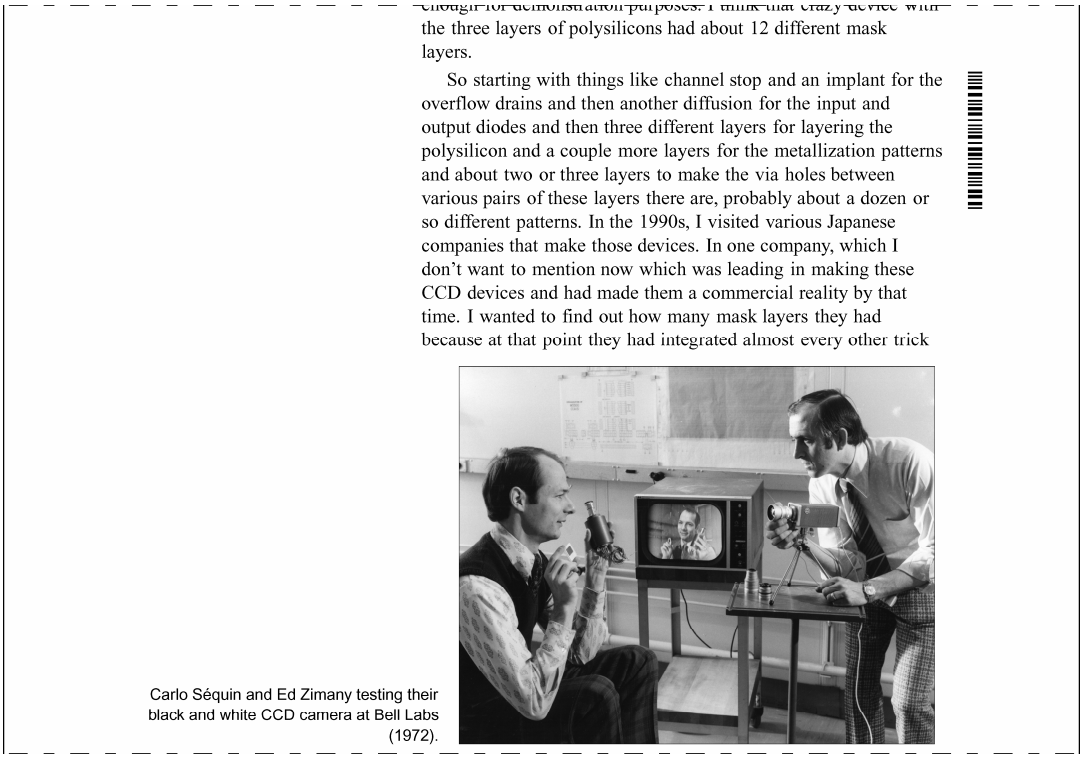


FIGURE 6.4 Detail of an internal page including a photograph.

Books with Voices requires a PDA with: 1) a display screen, 2) audio output, 3) a scanning device (such as a barcode reader or digital camera), and 4) access to a video store (we use a 2 GB PC card hard drive on the iPAQ; this could also be achieved by wirelessly streaming video to the device.)

Section 2.9 outlined the benefits and drawbacks of different tagging technologies; the four technologies potentially appropriate for an augmented book are:

1 Passive electronic tags (*e.g.*, RFID tags)
2 Active electronic tags (*e.g.*, motes [59])
3 Barcodes (including 2D variants such as DataGlyphs [103])
4 Vision-based content analysis (*e.g.*, optical character recognition)

Our inquiry found photocopying to be an important historical research practice, making tagging inappropriate. Tagging would also make book production more expensive and time-consuming. OCR and barcodes work with photocopies and require

no additional materials or time. Barcodes are preferable for our domain because they are more reliable and the interaction is simpler.

6.6 Interactive Prototype Evaluation

To understand both professional and amateur use, we evaluated the utility of Books with Voices as an augmented reading tool with thirteen users: eight oral historians from ROHO (two history professors, three editors, one interviewer/editor, and two transcribers), one oral historian from the University of San Francisco (a history professor), and four members of the UC Berkeley Computer Science Graduate Student Book Club.

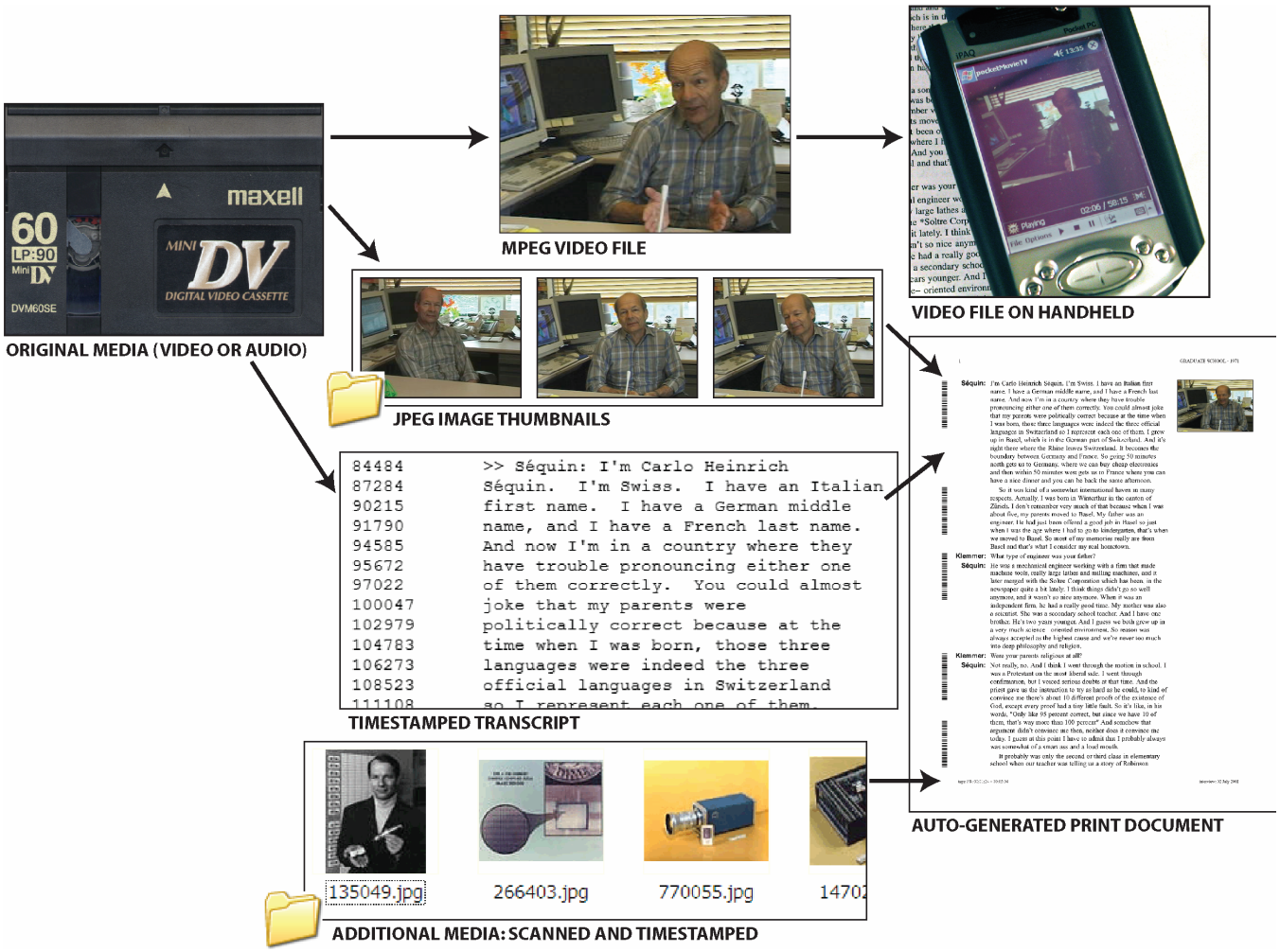


FIGURE 6.5 The Books with Voices pipeline.

We conducted this study to learn A) if and when oral historians find video access valuable, and B) how suitable our interaction techniques are for oral histories.

6.6.1 Study design

Each session lasted between 90 minutes and two hours. The studies were conducted in the participants workplaces. We videotaped the sessions and took time-coded handwritten notes.

First, we showed participants the printed transcripts, and demonstrated using the trigger button (see FIGURE 6.6) to scan a barcode, invoking the corresponding video. We then gave the users a few minutes to practice using the system.

In their oral histories, Professors Patterson and Séquin both talk about graduate school. Patterson's discussion of graduate school is 15 pages long (42 minutes of video) and Séquin's is 10 pages (28 minutes of video). For our main task, we asked users to spend 30 to 45 minutes reading these sections and then write a short summary of what they read. To watch all of the video would take 70 minutes; we intentionally designed the task to have more video than could be watched in the allotted time.

FIGURE 6.6 The trigger button initiates barcode scanning.

After completing the main task, we asked the professional oral historians to complete two short editing tasks (these were not relevant for the book club members). Oral history transcripts are nearly always edited for clarity (*e.g.*, removing "um," "like," and false starts), and sometimes edited for flow. Hypothesizing that strongly correlating transcript and recording benefits editing tasks, we asked participants to edit one page of the paper transcript and compare an already edited page of the paper transcript with the recorded video. At the end of the study, we asked the participants to fill out a 35-question survey, addressing their background, current practices, and opinions on the system. Twenty questions were multiple choice; the remaining fifteen were free response (see APPENDIX B).

6.6.2 Results

The study participants (see TABLE 6.1) took between 26 and 58 minutes to complete the main task (mean 43.1, median 45), successfully accessing between 2 and 21 media clips (mean 9.5, median 10). Eight users experienced a total of 31 failed scans (mean 2.4). One user accounted for 12 of the failed scans; she also had 11 successful scans. We believe six of the failed scans were because users did not hold the device fully orthogonal to the barcode, and 25 were because of hardware errors (*e.g.*, the barcode expansion pack was not properly seated) or software errors (*e.g.*, we printed a faulty barcode).

User #	Time to complete main task *(minutes)*	Number of successful accesses	Number of failed accesses
1	58	14	7
2	39	18	4
3	48	6	3
4	45	10	1
5	45	2	2
6	44	4	0
7	56	11	12
8	53	21	0
9	44	4	1
10	46	10	0
11	26	10	0
12	30	5	0
13	26	9	1
mean	*43*	*9.5*	*2.4*
median	*45*	*10*	*1*

User #	Listening position: Headphones	Listening position: Telephone	Listening position: In-hand	Environment: Quiet	Environment: Noisy	Reading position: Lap	Reading position: Desk w/o PC	Reading position: Desk w/PC
1			●	●		●		
2	●				●	●		
3		●		●				●
4	●			●			●	
5			●	●			●	
6		●		●				●
7			●	●			●	
8			●	●		●		
9	●			●				●
10		●		●			●	
11			●	●		●		
12		●		●		●		
13			●	●		●		
total	3	4	6	12	1	6	4	3

TABLE 6.1 Task time, access statistics, and usage style for the thirteen users in our study.

6.6.3 Benefits of paper for fast, direct video access

Our study shows that users frequently and fluidly access recorded interviews when paper books are the interface. When asked, "Do you feel that Books with Voices would change your usage practices?" participants reported responses such as, "Would be easier to select and listen to portions of an interview—easy to find excerpts—this is what is way too hard and time-consuming with analog tapes" (#10); "I think the video/audio would frequently be useful for confusing and interesting passages" (#3); and "Yes—accuracy, deeper meaning" (#5).

Participants had several motivations for accessing recordings: to get a sense of the personality of the interviewee, to hear the tone of a particularly compelling passage, and to verify the accuracy of the transcript. Participant #9 watched the video extensively at the beginning of the main task. At the end of the study, we asked her about this. She responded that watching the video helped her understand Professor Séquin's character.

All 13 users indicated interest in using Books with Voices for historical research. On a five-point Likert scale, half the users reported they would be "Very Likely" to use the system and half reported they would be "Somewhat Likely." None responded "Neither Likely nor Unlikely," "Somewhat Unlikely," or "Very Unlikely."

When asked, "What aspects of the system do you particularly like?" several participants complimented the system for its ease of use and for, "Being able to sense tone of voice, context." More than half specifically appreciated the direct access, *e.g.*, "I like how easy it is to access specific points in the audio" (#2). Aspects of the system they particularly disliked included the lack of fast forward and rewind, that the audio was "hard to hear" (#4), the "image hard to see" (#6), and that the "scanner is bulky" (#12).

Nine of the thirteen users reported that, "the number of barcodes" was about right. Three reported, "I'd like barcodes a bit more often," and one reported, "I'd like barcode

a bit less often." This indicates to us that barcodes offered enough value that their visual addition did not bother the participants.

Books with Voices proved effective for the editing task. With analog tapes, the editing process is often compromised by the difficulty of accessing the appropriate media segment. While transcribers sometimes verify transcripts against the tapes, it is rare that the interviewer or the interviewee has time. Users fluidly integrated video watching into their editing process. The participants responded enthusiastically to the question, "What role, if any, do you think Books with Voices could play in editing transcripts?" writing "Very helpful to determine context, expression, figuring out certain words" (#1), "I think it could make the process more efficient—in time for searching for tape sections" (#2), and "Fantastic, accuracy, nuanced" (#5).

Our transcripts contained the word "*inaudible" when the transcriber could not clearly hear what was said, and he prefaced transcribed words with a "*" when he was not confident the transcript was correct. Most of the participants referred to the video to clarify these inaudibles, and/or a sentence that appeared incorrectly transcribed to the participant.

6.6.4 Richer practice, minimal overhead

We often look to new technologies, like Books with Voices, to save labor. Our participants accomplished with just a few button presses what would have otherwise consumed hours. More importantly, our system makes a *richer* practice *tractable*. Books with Voices augments reading with an audiovisual experience previously unavailable. However, this experience takes slightly more time than just reading. In the user study, some participants (*e.g.*, #4) stopped using the software when they felt rushed late in the task.

6.6.5 Listening and watching

We found that people held the device in especially different ways, depending on their age, body size, personal style, and working environment (see TABLE 6.1).

Usability of audio

Participants were free to listen using the iPAQ's built-in speaker or to use headphones. Hearing the audio clearly was of great importance to the users. Three used headphones; #2 and #4 worked in noisy offices; #9 worked in an open, but quiet, office cubicle space. Of the ten speaker users, four held the device to their ear like a telephone, and six held it in their hand. Five of the six subjects under 30 in a quiet work environment held the device in their hand. Only one of the six participants over 30 held the device in their hand; two used headphones and three held it like a telephone. FIGURE 6.7 shows the diverse ways that participants read and interacted with the device.

The speaker volume on the iPAQ (like most PDAs) is soft. In a quiet environment, it may be acceptable, but generally speaking, we believe that once accustomed to the device, users will use headphones or plug the PDA into external speakers. In the questionnaire, one user lamented that because of the quiet audio, "I had to hold it up to my ear and not see the video."

FIGURE 6.7 Video stills from our evaluation: participants watching and listening to oral histories on the Books with Voices PDA. Note the many configurations of the PDA and the paper transcript.

Usability of video

When listening in a telephone style, as four users did, it is not possible to watch the video. The nine non-telephone users sometimes watched the video, but often did not. Several participants explicitly complained that the video was either too small or not bright enough. Increasing the resolution and brightening the videos (either with software or a physically brighter device) would help. Our prototype's 2 GB hard drive holds 20 hours of video in its current form. If file size becomes an issue, it is worth lowering the frame rate or using higher compression in exchange for increasing the resolution. Although sound is much more important than image for this domain, we believe a well-presented video is still valuable.

Reading styles

All four book club readers and two oral historians read with the transcript on their lap. Four historians read on a desk without a computer, and three read on a desk with a computer. Our transcripts were bound in three-ring binders. We never would have expected binder style to become a usability issue. The physical world brings both physical benefits and physical problems. Séquin's binder was like a hardcover book; it had a cardboard spine and covers, and just plastic in between, allowing it to remain closed or open. Patterson's binder was made from one piece of shaped plastic; it would stay closed, but not open. Participant #4 promptly got rid of the binder, and several other users responded by weighting the binder to keep in open. After six users, we fixed the problem, replacing the "paperback" binder with a "hardcover" binder.

Multitasking: reading while listening

Because people read three times faster than they listen, it can be difficult to do both simultaneously. However, we found that several users comfortably listened to one

section while reading another. A few users (*e.g.*, #5) also listened while writing their summary. Generally, it seemed that younger users were more comfortable with this multi-sensory approach.

6.6.6 The barcode scanning process

Perhaps the largest usability issue is that the user has to rotate the PDA somewhat awkwardly to scan barcodes. Currently, the device must be held *perpendicular* to the paper and *oriented* so the scanner scans vertically. We noticed that most participants had difficulty with this; three asked for horizontally oriented barcodes in the free-response section of the questionnaire (#5, #7, and #9), and a few more asked about this during the study.

The *perpendicular* requirement is an intrinsic property of today's infrared barcode scanners. (A barcode reader mounted on the back face of the device would work, though it might be bulky.) The vertical *orientation* is a design flaw of our system that could be remedied either by horizontal barcodes or by presenting video horizontally on the PDA. Moving from an infrared barcode scanner to a PDA with a rotatable integrated camera, such as some Sony CLIÉs offer, would address both rotation issues. This way, the device could be oriented for viewing, and the sensor could still be oriented for capture.

Making and recovering from errors

When there were scan errors, the device notified users through an audio chirp and a visual error dialog. However, about half of the participants scanned with their hand covering the device (because of the required orientation and the device's bulk). Therefore, they did not see the error dialog, and were unaware an error occurred.

Switching to an orthogonal sensing technology would likely remedy this; speech output of errors might provide additional clarity.

We found that the buttons need to be easier to access. The hardware trigger button should be larger. A few users had to be told where the GUI stop playback button was, and many had difficulty pressing it. It would be much better if stop was mapped to one of the four physical buttons on the lower section of the iPAQ.

Slow start latency

It takes about 3 seconds for users to orient the device and execute a scan, and 5 seconds to wait for the video to begin playing. Several users effectively adapted to this latency by pre-scanning the media, and putting the device aside while it loaded. One unexpected problem with the start latency is that some users did not realize that it had successfully scanned, and would re-scan the same barcode. This can be fixed in three ways: by better visual or auditory feedback, by ignoring duplicate scans, and by lowering the latency.

6.6.7 Visual design

The user study encouraged us to improve the visual design of the system in two ways. First, page numbers should be on the outside margins as opposed to the inside. Second, we should produce an index (as is currently done) and augment it with barcodes. Participants were very excited about a barcode-augmented index. Three asked for it in the questionnaire's free-response section and a few more mentioned it during the study.

6.6.8 Requested Features

Nearly everyone asked for fast forward, rewind, and/or backspace. Backspace is a technique available on transcription systems, letting transcribers auto-rewind a few seconds. A physical jog dial would be a good way to offer fast forward, rewind, and backspace.

There is great potential for Books with Voices as a cscw technology. In our questionnaire, we asked, "What are your thoughts on using Books with Voices to help you keep track of important parts of an oral history?" and would this "functionality be useful for sharing? (*e.g.*, email.) How might it work?" Participants responded enthusiastically, saying, "It could be pretty useful if you could keep a list of scanned sections and then choose to keep or delete chosen sections from the list" (#11). With regard to email, #1 wrote, "It would be helpful to email portions to others. Esp. for editing, since many people work on any single transcript," and #8 wrote, "There would have to be some way to add your annotations, but [email] seems useful."

6.6.9 General remarks

Participants used the device differently depending on their job. One transcriber mostly used the device for editing and fact checking, and could not help but make edit marks on the transcript as she read. A senior history professor accessed the video the least (twice). She took care to finish the study in a timely manner as she had another meeting immediately afterward.

While the current system makes text a fluid interface from text to video, there is no facility to go from video to text. Occasionally, participants got lost. Subtitles on the video indicating page and paragraph number might remedy this.

One success of the system in our study was that, while there were usability problems, none of the users found the system conceptually difficult. The concept of using paper transcripts as an interface to original recordings seemed perfectly "natural."

6.7 Summary

Reading is a highly evolved practice. Our evaluation showed that Books with Voices effectively enables active reading by scaffolding new technologies on paper, which is

highly familiar, cheap, and usable. Our technology support for oral histories differs from much current research in that it employs paper as an *archival,* rather than *ephemeral,* artifact. After using the Books with Voices device, the oral historians at ROHO asked us how they could switch to digital transcription tools. They saw many benefits to this, including the ability to use Books with Voices. They now store all of their audio and video digitally, and time stamp their transcriptions using the Start Stop transcription system [24].

We have worked with the Oakland Museum to explore a deployment of Books with Voices in a museum setting. The deployment was thwarted by the fact that Pocket PC devices lose all of their memory (and thereby any installed software) when they run out of batteries. However, the museum curators were very excited about the technology, and it is likely that they will adopt a technology like Books with Voices once handheld hardware is more reliable.

The core technology element in Books with Voices is associating a physical tag with an electronic media file and a time index into that file. As with most of our inspiring applications, this functionality was built from scratch, and experimenting with alternate versions (such as vision-recognized barcodes or RFID tags) would be a substantial engineering effort. Additionally, even the small changes we have made to Books with Voices (e.g., the optional inclusion of a time to stop playback in the barcode) have been a time-consuming re-engineering effort. Papier-Mâché's software architecture would have allowed Books with Voices to be built much more quickly. In addition, technology, user interaction, and structural changes could have been made more easily and without the danger of breaking legacy code.

7 Fieldwork Inspiring Papier-Mâché

"Every decision … bases itself on something not mastered, something concealed, confusing: else it would never be a decision."

—Martin Heidegger, 1971 [105].

Our experiences with the Designers' Outpost and Books with Voices provided a wealth of experiential knowledge about the development of tangible interaction. In keeping with our design philosophy of "triangulation"—employing multiple methods to gain different types of understanding—we also endeavored to learn from the experiences of others. Toward this end, we conducted structured interviews [37] with nine researchers who have built tangible interfaces. Four of the interviewees worked in academia; the other five in industrial research. There were 28 interview questions addressing general system questions, planning and organizational structure, software design, user and system evaluation, and difficulties in design and implementation (see APPENDIX C). We conducted these interviews in person at the workplaces of researchers in the San Francisco Bay Area (three), and over the phone (one) or via an email survey (five) otherwise. These researchers employed a variety of sensing techniques including vision, radio frequency and capacitance sensors, and barcodes. Here, we present the findings of this research; we concentrate on the difficulties encountered in designing and developing these systems. To maintain the anonymity of interviewees, we use examples

from our own research to illustrate our findings, rather than the actual systems built by our interviewees.

7.1 Difficulties of Acquiring and Abstracting Input

Employing physical objects as input devices implies the use of novel hardware: some physical instrumentation is required for the computer to sense the presence, absence, and manipulation of objects. A general theme among interviewees was that acquiring and abstracting input was the most time consuming and challenging piece of application development. This is not, as the cliché goes, a "small matter of programming." Acquisition and abstraction of physical input, especially with computer vision, requires a high level of technical expertise in a field very different from user interface development.

These novel input technologies, especially vision, do not always functioning perfectly. We found that consequently, it is important to design a system where occasional errors do not prevent the system as a whole from function, and to provide feedback so that users can diagnose and help recover from system errors. An interviewee explained that, "The sensing hardware is not perfect, so sometimes we had to change interactions a bit to make them work in the face of tracking errors." This error-aware design of TUIs is similar to the techniques used for voice user interface design [170], where limiting grammar size and providing confirmation feedback help systems minimize errors, and help users diagnose and recover from errors when they do occur.

7.1.1 Group size and composition

The size of the nine hardware and software development groups ranged from one to four (both the mean and median size were three). Between one and five additional people were involved with interaction design and/or industrial design but not with

software development (the mean was 2.4 and the median 2.7). Sometimes these conceptual contributors were the project manager or an advisor. Other times the conceptual contributors were colleagues with a different project as their primary focus.

In each of the three projects that employed computer vision, the team included a vision expert. Even with an expert, writing vision code proved challenging. In the words of one vision researcher, "getting down and dirty with the pixels" was difficult and time consuming. Writing code without the help of a toolkit yielded applications that were unreliable, brittle, or both. Additionally, in two of the non-vision projects, the person who developed the tangible input was different from the person who developed the electronic interactions.

In the remaining cases, the developers all had a substantial technical background, and worked on both the physical and electronic portions. We speculate that if tools with a lower threshold [188], such as Papier-Mâché, were available to these individuals, then a larger portion of the team may have contributed to functioning prototypes, rather than just conceptual ideas.

7.2 Iterative Implementation, Using the Familiar

Iterative implementation was alive and well among our interviewees. All of the interviewees' systems evolved from or were inspired by existing projects in their research groups. For two of these researchers, the evolution was from a virtual interface to a tangible interface, as in our research group, where the Designers' Outpost was partially inspired by the DENIM pen-based sketching tool [153, 192]. For two others, the tangible interface was an application or evolution of a physical input technology that the group had previously developed or had experience with. Four researchers had experience building TUIs prior to the project discussed in the interview. For these groups, the project we discussed was a continuation of work in this area. This next step was either

exploring an alternate point in a design space, exploring richer interactions, delivering greater use value, or exploring lower complexity. We see two primary reasons for this evolutionary iteration:

1 Good researchers often work in a similar area for a number of years, exploring different points in a design space.

2 Reusing existing code and/or technical knowledge lowers the threshold for application development.

7.2.1 Paper and existing code for low-threshold prototyping

Two of the interviewees began with paper prototypes, often trying out different scenarios to understand the interactions required before writing code. "The paper prototypes helped us understand the space considerations/constraints—Helped us work through the scenarios." One of these researchers also used physical objects (without computation) to think through the interaction scenarios, "to get an idea of what it would feel like to use our system."

The remaining seven interviewees began prototyping with technologies and tools that they were familiar with or had a low threshold, later exploring less familiar or higher threshold tools. Working with similar technologies and tools over a number of years affords fluency with a medium, in much the same manner as artists tend to work in the same media for long periods of time. An interviewee explained to us that he "was able to leverage the technology that we had earlier developed to build a prototype (with minimal functionality) in roughly eight weeks. Extended that work into the application that was fielded to end-users approximately two years later."

An alternate method of achieving fluency, or readiness-at-hand [106], is the use of tools with a low threshold. One researcher appreciated Max [19], the music and

multimedia authoring environment because, "It's about I can make five versions of this by next Tuesday." This researcher chose to use Max for prototyping even though it was not appropriate for the final implementation.

7.2.2 Length of projects

We uncovered a somewhat surprising regularity in project duration. When responding to an unstructured question about the time their research project spanned, six of the nine interviewees reported their project as lasting two and a half to three years. Often this timeframe was qualified that the main research effort within this spanned about a year and a half, and that there was a fast and furious period of prototyping at the beginning. The above quote describing an eight week prototype is an example of the fast and furious phase. The three remaining projects were shorter, lasting six months to a year. This is consistent with the time-scale of all three projects described in this dissertation. We fast and furiously built the basic interactive Outpost over a period of two months in the Summer of 2000. This was preceded by almost a year of design, and followed by two years of refinement, feature additions, and evaluation. Books with Voices and Papier-Mâché followed a similar pattern.

7.2.3 Refactoring as architecture needs became clear

Often, the interviewees re-designed aspects of their system architectures to support a wider range of behavior. In Outpost, for example, we introduced a command subsystem to support design history and remote collaboration. This refactoring reflects positively on the developers; advocates of agile programming methods argue that software architectures should be simple at first, becoming more complex only as needed [32]. This heavy refactoring also reflects negatively on the current state of software tools in this area. Much of the modularity that our interviewees introduced could be more

effectively provided at the toolkit level, and indeed is supported in Papier-Mâché. Examples include changing the type of camera, switching between input technologies, or altering the mapping between input and application behavior.

The challenge of refactoring with limited development resources is that it often is "hacked"— the code is altered just enough to support the additional functionality, but not in a manner that is robust, flexible, maintainable or understandable. One developer described their situation as "The code was way too complex at the end of the day" because there were "a lot of stupid problems" such as temporary files and global variables that inhibited reliability and malleability. These systems often rigidly relied upon all components being available and functioning properly, and did not fail gracefully when most, but not all, of the components were working. In one case, the interviewee's group created an improved version of their core technology, but it was too late for use in the research prototype, and happened only "after one of the engineers finished what I had begun." Toolkits can alleviate many of these development headaches.

7.3 User Experience Goals Motivating Tangibility

The primary motivation our interviewees had for building a tangible interface was the desire for a conceptual model of interaction [197, ch. 1] that more closely matched user's behavior in the real world, often as one interviewee described it, "trying to avoid a computer." This motivation is also seen in Fishkin's discussion of embodiment [77] in tangible user interfaces (see the summary of this work in section 2.7.2). In addition to the development issues, we spoke with the interviewees about their user experience goals. Here are three quotes from the interviewees that illustrate some of these goals:

- "Technology should make things more calm, not more daunting."

- "People don't want to learn or deal with formidable technology."
- "They're torn between their physical and electronic lives, and constantly trying work-arounds."

The central finding of *The Myth of the Paperless Office* [222] (summarized in section 2.1) is that users work practices are much more successful, and much more subtle, than a naïve techno-utopianistic perspective might suggest. Additionally, the book illustrates that while digital technologies change practices, they do not supplant our interaction with paper and physical objects. The interviewees, through design insights and their own observations of practice, came to a similar conclusion. This reverence for the nuanced success of everyday objects inspired much of the interviewees work. These interviewees had many of the same goals that our literature survey found (see section 2.1), such as avoiding projector-based solutions in order to increase robustness to failure. An excellent example of this appreciation for the success of our interaction with the physical world can be seen through one researcher's goal that his group's system, "should mimic [users] current tools as closely as possible to begin with. Functionality can be added that is unavailable in the paper tool, but only after [our TUI] captures their current work practice."

7.4 Development and Reuse: Architecture, Library, Functionality

A user interface software toolkit comprises two pieces: "a library of interactive components, and an architectural framework to manage the operation of interfaces made up of those components. Employing an established framework and a library of reusable components makes user interface construction much easier than programming interfaces from scratch" [188]. A particular application uses library components, assembling them and communicating with them using the software architecture. This is

then extended with the particular functionality required by the application. Ideally, there should be a small number of architectures, and each architecture should span a reasonable space of applications. A new architecture should be introduced only when a new class of applications is introduced. There should be a moderate number of library components. Some components (such as the basic GUI widgets and file I/O protocols) can be used in a huge space of applications. Others (such as a domain-specific visualization or algorithm) are used in a much smaller number of applications. Additionally, each application will have its own functionality that developers create, or at the very least, they compose and customize library components.

In our interviews, we found that for this emerging area of tangible interfaces, each development team was creating an architecture, a set of library components, and an application (though the developers did not generally describe their work with such an explicit taxonomy). For all of our interviewees, Papier-Mâché would have eliminated the need to create a software architecture, with the exception of the distributed portion of the applications, for those that had one. Papier-Mâché would have also drastically minimized the amount of library code that needed to be written. Examples of library code that would have remained include particular vision algorithms, support for particular brands of RFID readers, and support for particular flavors of barcodes. Papier-Mâché would have also substantially reduced the amount of application functionality code, shifting the balance from creation from scratch toward composition of components.

7.5 Events and Constraints are More Appropriate Than Widgets

Widgets are a software abstraction that encapsulates both a *view* (output) and a *controller* (input); see SECTION 2.5 for a summary of how this is accomplished with graphical UIs. While some post-WIMP toolkits have hoped to provide an analogue to

widgets (*e.g.*, the Context Toolkit [65]), in practice *toolkit support for the view* (output) is distinct from *toolkit support for the controller* (input), and with good reason: a particular piece of input can be used for many different types of output and vice versa.

In designing an architecture for tangible interaction, we can learn from the abstractions introduced in the model-based UI research (see [241] for a comprehensive literature survey). With model-based tools, developers specify an interface declaratively, and at a high level. In a traditional GUI tool, a developer might imperatively specify that a group of radio buttons should be displayed in a pop-up menu. With model-based tools, a developer might specify that an application should present a widget for a user to designate an item from among a set. The model-based tools would then decide upon the most appropriate widget (radio button, drop down menu, *etc.*). This high-level, declarative specification offers a flexibility of implementation. In the era of desktop interfaces, the primary goals of model-based research were "increasing the quality and reducing the cost of developing interfaces" [241, p. 1]. With the emergence of ubiquitous computing, and the increased number and heterogeneity of devices, model-based techniques have seen increased interest. Two research directions that attempt to ameliorate this increase for developers and end users are tools for creating a single application code base that can be deployed on multiple distinct devices (*e.g.*, [85, 140]) and for automatic UI personalization (*e.g.*, [259]). Papier-Mâché's event structure and behaviors draw from this literature to provide a similarly high level of abstraction, allowing developers to declaratively specify objects, events, and behavior at a semantic level, *e.g.*, "for each Post-it note the camera sees, the application should create a web page."

The basic event-based software design patterns were uncannily similar across many of these systems. While each of these systems was built independently, without access to the source code of others, all interviewees settled on a small set of similar events for

defining behaviors in applications. The basic events hinged on notifying application objects about the presence, absence, and modification of physical objects. These events were used to bind manipulations in the physical world to electronic behavior.

In the tangible world, as in the graphical world, events tend to fall into one of a few groups. The first group is events that carry as their payload one or more *binary* values. Examples of this flavor include events that are triggered when a button is pressed or released, when a pointer enters or exits an area, and when an RFID tag or barcode moved into or out of a sensor's range. A related set is event properties that take on one of a small number of *discrete* states. The second group is events whose properties are one or more *continuous scalar* values. One example of this type is the planar position of the mouse; or for a vision-tracked object, its position, orientation, size, and other scalar values that the vision system reports. A third type is the rich *capture* of information from the world; for example, an audio recording, video recording, or photograph. These sources may have particular aspects of the recording semantically extracted (Outpost extracts the images of Post-it notes from images of a board).

Using these basic primitives, some of the interviewees created higher-level events, and *constrained* the system behavior to be a function of these events. For example, one interviewee created a distance operator that measured the distance between objects on a surface, and constrained the behavior to be a function of that distance.

7.6 Declaratively Authoring Behavior

In general, UI programming languages and software architectures employ either an *imperative* or a *declarative* programming style, and sometimes a combination of the two. Imperative code provides a list of instructions to execute in a particular order, *e.g.*, a Java program that counts the number of words in a text file via iteration. Declarative code describes *what* should be done but not *how*. SQL [87] is a well-known example of a

declarative language: with SQL, database queries specify the information to be returned but not the technique for doing so. Spreadsheets are another highly successful example of declarative programming. For example, the value of a cell might be defined as the summation of the eight cells above it. This value updates automatically as the other cells change, and no "program" or control loop needs to be written to implement this functionality; it is handled by the system internally. Declarative programming has a model-based [241] flavor, and is beginning to rise in popularity for programming web services (*e.g.*, Sun's Enterprise Java Beans [9]) and graphical UIs (*e.g.*, Microsoft's Avalon [1]).

Tangible interfaces couple physical input with electronic behavior; for example, a marble represents an answering machine message [204]. This coupling, where the relationship is either discrete (such as the marbles), or continuous (such as the links in Outpost, which have their endpoints constrained to physical Post-It notes) can be more concisely and flexibly expressed declaratively than imperatively. All nine of our interviewees described the behavior of their system as providing tangible input coupled with electronic behavior through event-based bindings or queries. In associative applications, this binding is quite straightforward: barcodes or RFID tags serve as a physical hyperlink to a particular piece of electronic behavior or media. These elements are bound together, and the behavior is executed through an event. With spatial applications the binding is parameterized through the location, size, shape, and other identifying characteristics of the object, and with topological applications the behavior is influenced by the relationships between the physical objects.

While our interviewees described their systems clearly using declarative terms, not everyone felt this declarative model was effectively implemented in their software. Several interviewees wished they had a more flexible method of defining bindings,

making it easier to change the input technology and to explore alternative interactions for a given input technology.

7.7 Choice of programming language

At the time of TUI development, our interviewees used several different programming languages: C++ (three), Java (two), Prolog (one), Director (one), Visual Basic (one), and Python (one). Two of the non-Java teams have since switched to Java. Eight of the nine interviewees used a Windows PC as their development platform.

Most of the interviewees chose a programming language based upon one particular requirement. They requirements cited were:

1 *Technology integration:* Sometimes, the decision was made to ease integration with a particular piece of input technology, *e.g.*, the Outpost vision system was built in C++ because OpenCV was in C.

2 *Library support:* The majority of our interviewees chose a language based on the library use it facilitated. Two developers chose the Windows platform and a Visual Studio language specifically for their easy interoperability with Microsoft Office. A third interviewee was constrained to the Windows platform for integration reasons, and chose Director for its rapid development capabilities. Two of our interviewees decided on Java because of its rich 2D graphics libraries.

3 *Developer fluency:* For one interviewee, C++ was chosen "because [the lead software developer] knew it well." The same fluency sentiment was expressed differently by another interviewee that C++ "was our language of the time. Now we're Java."

4 *Rapid development*: One interviewee told us that "I used Python. This language make prototyping fast and easy. It can be too slow at times, but not too often thanks to

Moore's law." Another interviewee, mentioned above, settled on Director [5] for rapid development reasons.

These four reasons—technology integration, library support, developer fluency, and rapid development—informed our choice of Java as a programming language. Java offers excellent library support, many developers are fluent in it, development time is very fast for a full-fledged programming language, and it was tractable for us to provide input technology integration. The double-edged sword of Java is its platform independence. Development and execution on multiple platforms provides a wide audience of developers and a flexibility of deployment, and this was a feature that one interviewee specifically requested. However, this independence comes at a performance cost and makes integration with novel input technologies more difficult. In 2004, C# would probably be a slightly better choice for a toolkit like Papier-Mâché from a technical perspective. There is a very real technical and political cost in losing platform independence, but the performance and integration gains may be more valuable in many cases.

7.8 Choice of input technology

Section 2.9 addressed the benefits and drawbacks of different input technologies and tagging approaches. The technology that can offer the greatest benefit is vision, but it brings along with it reliability concerns and development headaches. Our interviews uncovered how these tradeoffs manifested themselves in practice. A great appeal of vision was that, "it gives you information at a distance without a lot of hassle, wires, and instrumentation all over the place. It puts all the smarts in one device and instrumentation is limited. It also is possible to retrofit existing spaces."

7.8.1 Input technology portability

Most of the interviewees either experimented with different input technologies, or were interested in trying different input technologies for their application. As with tools and languages, the choice of sensing technologies sometimes also shifted between prototyping and implementation. One interviewee prototyped his spatial interface with a SMART Board as the sensor. Later, he replaced the SMART Board's touch sensor with vision for two reasons: First, SMART Boards are expensive and bulky, while cameras are inexpensive and small. Second, the resistance-based SMART Boards provide single-input of [x, y]. (The newer DVIT SMART Boards support multi-touch interaction by using four cameras mounted in each corner, orthogonal to the input surface.) Vision offers a much richer input space. This vision task is exactly the kind of task that Papier-Mâché can support. While vision offers many benefits, all of the interviewees shared the sentiment that, as one researcher explained, "The real-time aspects of camera interfacing were probably the hardest." Another researcher, after having completed his project lamented that, "it's not always worth it to live at the bleeding edge of technology. … Make sure you have a very good reason if you choose to work on a problem whose solution requires pushing more than one envelope at most."

One interviewee explored several input options before settling on RFID because, "we didn't know what to do" for input. As an experiment, "I had an intern that did a version with optical sensors (vision)." Another RFID user first, "spent a lot of time looking into barcodes and glyphs, but they didn't seem right."

Myers, Hudson, and Pausch [188] point to rapid prototyping as a central advantage of tool support. Vision is an excellent technology for rapid prototyping of interactive systems. It is a highly flexible, software configurable sensor. There are many applications where the final system may be built using custom hardware, but the prototypes

could be built with vision. The one challenge, prior to Papier-Mâché, is that it is difficult to rapidly prototype applications with computer vision.

7.8.2 "More than two serial ports"

When we asked for specific toolkit functionality requests, one researcher quipped, "more than two serial ports." In nearly all contemporary software architectures, a single mouse and keyboard are presumed to be the only input devices. Employing alternative input mandates a large software engineering effort. In some cases, the hardware- or software-imposed ceiling on the number of input devices prevented the researchers from realizing their ideal designs. As most hardware has migrated to USB or FireWire, it is becoming less true that the number of "serial ports" *per se* are a limiting factor, but the software's ability to handle and program these simultaneous inputs is still a limitation.

7.9 Distributed Applications

Three of the nine applications provided the ability for multi-device, networked interaction. These systems were designed roughly around a distributed MVC architecture, where a database served as the server and central connection point. In these systems, the clients supported sensing input, information presentation, or both. Clients would report events to a server hosting the model, and then the server notified all of the other clients. In the most sophisticated system, the interaction clients were heterogeneous. Board clients reported data to a board server, and this server then sent events to applications, which were often web apps, but could also be devices like a printer. The Papier-Mâché toolkit targets single-machine applications. By virtue of the fact that Papier-Mâché integrates easily with the SATIN ink UI toolkit [110], there is a remote command system available for replicating events between hosts (see SECTION 5.5.1). A toolkit that more explicitly supports distributed interaction and data storage could draw

upon much of the literature in ubiquitous computing toolkits (see section 2.6), and is an area for future research.

7.10 Importance of System Feedback for Users and Developers

Good feedback is a central tenet of user interface design [197 ch. 4, 228 §11.2]. One researcher found that, "One key issue was that sensing errors were pretty mysterious from the users' perspective." Providing visual feedback about the system's perception of tracked objects helps users compensate for tracking errors. Feedback is particularly important to developers, because the complexity of their task is so high.

Debugging is one of the most difficult parts of application development, largely because of the limited visibility of dynamic application behavior [62]. The novel hardware used in tangible uis, and the algorithmic complexity of computer vision, only exacerbate this problem. One interviewee had "the lingering impression that the system must be broken, when in fact the system was just being slow because we were pushing the limits of computation speed." The current state of debugging tools in this area is quite poor; another interviewee used Hyperterm, the Microsoft Windows command line tool designed for modem communication, to debug the serial communication of their input hardware.

7.10.1 Understanding the flow of control

Imperative software languages make it very difficult to see the flow of control of an application, especially one that is highly reliant on events. Control flow moves rapidly between class files, making it so that understanding a particular behavior can be quite difficult. One interviewee explained that, "The debugging of incorrect or missing elements within the fusion operation was the most cumbersome and time-consuming parts of the development process." This should not be taken as a critique of object-

oriented design, but rather as a critique of relying solely on textual information for software understanding. Well-designed visual tools can be more effective for creating and debugging dataflow. Successful examples of visual dataflow graphs include the Max MIDI authoring system [19] and spreadsheets [47, 123].

7.10.2 Feedback over longer time scales

Visualizations explaining the current state of an application are useful at short time scales, and Papier-Mâché provides this. An area for future research would be to provide logging and feedback of application behavior at longer time scales. One interviewee told us that he "Put it up, and ran it for about six months in two or three locations in the building." To evaluate the robustness of the system, he then watched for failure mode. "These failure modes helped drive further development. This failure mode analysis is key." Another told us that, "We were worried about robustness. So I made a prototype and left it in the hall for months." There are three broad areas where long-term monitoring could help: software errors (such as crashes), recognition errors, and usability errors (where the software behaved as expected, but the user interface was poor). The first could be addressed by self-evaluating software techniques similar to those of Liblit *et al.* [150]. The second could be addressed through logging user mediations. The last could be addressed by logging access to help systems (when available) and undo actions, or by introducing a user feedback system (a simple example would be an "I don't understand" button that could log application state).

Several participants specifically asked us for better error diagnostic tools. One researcher gave us some example questions that she hoped tools would solve: "This crashed, what happened? Why won't it boot? How far does it get?" Interviewees also asked us for the ability to remotely administer and diagnose deployed systems. They

wanted to be able to find out answers to questions such as, "Which sensors did they use? In the way you think or something else completely?"

7.11 Summary

The threshold for developing robust tangible interfaces in the absence of tools is tremendously high. This high threshold discouraged experimentation, change, and improvement, limiting researchers' ability to conduct user evaluation, especially longitudinal studies. One interviewee avoided these studies because his team lacked the resources to "add all the bells and whistles" that make a system usable.

The results that had the greatest effect on the Papier-Mâché architecture were: 1) all of the interviewees used an event-based programming model; 2) interviewees experimented with different input technologies, and that each of these prototypes was built from scratch; and 3) understanding the flow of an application and debugging failures is quite difficult.

8 The Papier-Mâché Architecture

The preceding six chapters of this dissertation provide a design space of existing applications and related toolkit work (CHAPTER 2), an in depth description of two applications our group has designed and evaluated (CHAPTERS 3–6), and an understanding of the architectural decisions and development challenges that represent current practice (CHAPTER 7). This literature survey, experiential knowledge, and fieldwork data show that a toolkit for tangible input should support:

- Techniques for rapidly prototyping multiple variants of applications as a catalyst for iterative design
- Many simultaneous input objects
- Input at the *object* level, not the *pixel* or *bits* level
- Heterogeneous classes of input
- Uniform events across the multiple input technologies, facilitating rapid application retargeting
- Classifying input and associating it with application behavior
- Visual feedback to aid developers in understanding and debugging input creation, dispatch, and the relationship with application behavior

---

Portions of this chapter were originally published by the author, Jack Li, James Lin, and James A. Landay in [133].

8.1 Introduction

These goals provided the basic framing for the architectural decisions in Papier-Mâché. A toolkit is software where the user interface is an API, and the users are software developers (see FIGURE 8.1). Papier-Mâché is an open-source Java toolkit written using the Java Media Framework (JMF) [17] and Advanced Imaging (JAI) [15] APIs. It abstracts the acquisition of input about everyday objects tracked with a camera, or tagged with barcodes or RFID tags. These three technologies span the vast majority of input needs of our 24 inspiring applications. (The exceptions are systems that employ tethered 3D tracking [81], speech input [173], or capacitive sensing [29].) This library supporting input technologies also illustrates Papier-Mâché's capability for a developer to originally implement a system with one technology and later retarget it to a different technology. The need for rapidly retargeting input encouraged our use of event-based bindings—rather than widgets—as an architecture for tangible interaction.

We explain the Papier-Mâché architecture using two examples: 1) an RFID implementation of Bishop's marble answering machine [117] (see SECTION 2.8.3), and 2) a simplified version of PARC's Collaborage In/Out Board [180] (see SECTION 2.8.1) using computer vision and barcodes. For each of these applications, a developer has two primary tasks: declaring the input that she is interested in and mapping that input to application behavior.

8.2 Input Abstraction and Event Generation

Papier-Mâché represents physical objects as *Phob*s. (In this dissertation, Java class names are designated in italics.) The input layer acquires sensor input, interprets it, and generates the *Phob*s. A developer is responsible for selecting input types, such as RFID or vision. She is not responsible for discovering the input devices attached to the computer, establishing a connection to them, or generating events from the input.

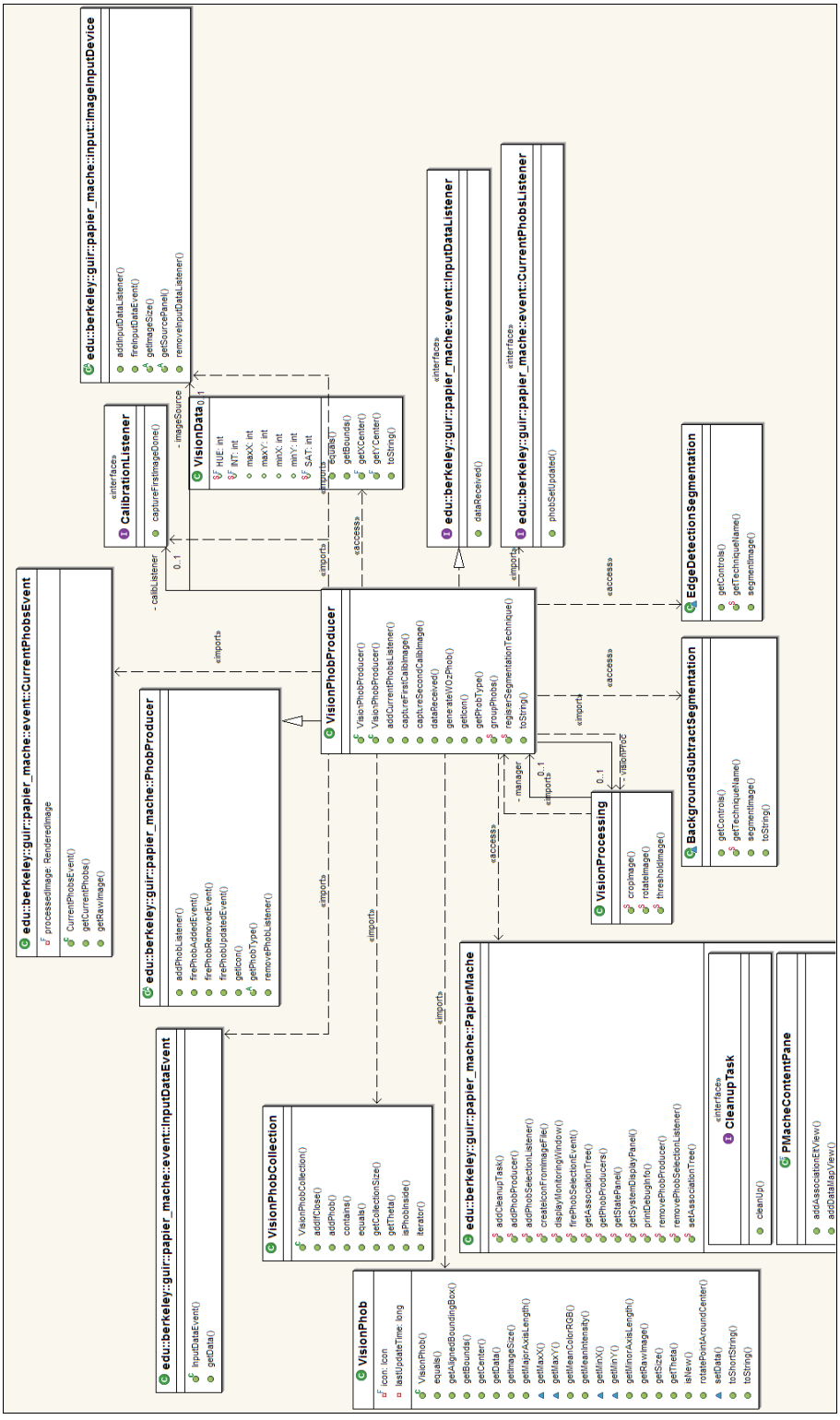


FIGURE 8.1 A toolkit is software where the user interface is an API: the architecture of the system, along with a set of classes and their methods. This is a UML diagram of a piece of the Papier-Mâché API.

These "accidental steps" [45] are not only time-consuming, but require substantial hardware and computer vision expertise, a field very different from user interface development. For example, the marble answering machine developer adds her application logic as a listener to an RFID reader but does not need to manage a connection to the hardware. Similarly, the Collaborage developer tells Papier-Mâché that he is interested in receiving computer vision events with a video camera as the source.

A piece of physical hardware that is attached to the computer needs to implement the *InputDevice* marker interface (see FIGURE 8.2, top row). The Java programming language makes a distinction between *classes*, which contain an implementation, and *interfaces*, which define a set of methods that implementing classes must support. A marker interface is a design pattern where a programmer creates an interface without any methods; implementing a marker interface is a technique for tagging the implementing class as providing a certain type of support. Using the inheritance mechanism as a tag provides for both compile-time and run-time verification that class instances provide the necessary support. In this case, each input device has a unique API for dealing with the underlying hardware. The *InputDevice* marker interface tags the implementing classes as being responsible for input acquisition.

For each category of input hardware, there is a class implementing the marker

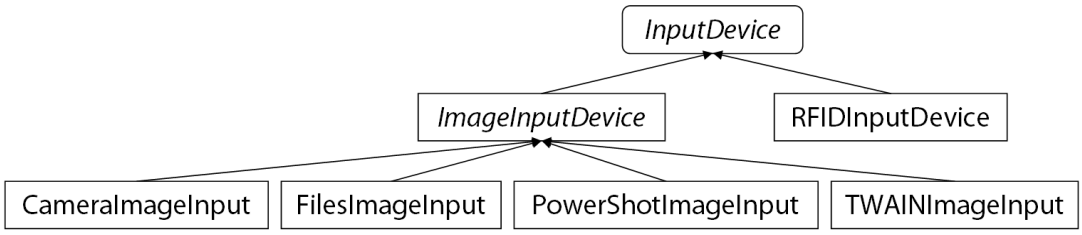


FIGURE 8.2 The inheritance hierarchy for physical input devices. Each device class encapsulates a physical input. The InputDevice is a marker interface: it is an interface class that contains no methods. All classes that represent a physical device implement the marker interface to denote that they represent a physical device.

interface that provides a general API to Papier-Mâché for handling that type of input (see FIGURE 8.2, middle row). The *ImageInputDevice* and *RFIDInputDevice* are the two library examples that Papier-Mâché includes for general device types.

For each device class, there are several different APIs currently available. Papier-Mâché abstracts the varying aspects of particular members of an equivalent device class. For example, different types of cameras have different types of APIs for image acquisition. These specific device styles use the public interface and event dispatch from their superclass, adding acquisition functionality that is particular to their type (see FIGURE 8.2, bottom row). Device drivers obviate the need to create implementations for all camera models; only one library element needs to be written for each imaging standard. The two main imaging standards are webcams and TWAIN. The Papier-Mâché library supports webcams through the Java Media Framework. JMF supports any camera with a standard driver, from inexpensive webcams to high-quality 1394 (FireWire) video cameras. Papier-Mâché supports TWAIN capture, an alternate protocol designed for scanners but increasingly used for digital cameras, through Java TWAIN [18]. Additionally, Papier-Mâché supports the use of simulated input through the *FilesImageInput* class. Simulated input is useful for creating and testing code when a camera or the environment is not available, and also for unit-test-style repeated verification of functionality of a system as it changes over time.

Still digital cameras were designed to be used by a human photographer. In our research, we have also found it useful to control these high-resolution low-frame-rate cameras computationally, both for structured image capture and for computer vision. However, currently, there is no widely adopted computational control standard for this class of device. In a few years, TWAIN and/or the Windows Imaging Acquisition standard will likely emerge as a commonly adopted standard. In the interim, we have provided the *PowerShotImageInput* class in the Papier-Mâché library. This acquires

images from Canon's PowerShot cameras. We chose these cameras because they are high-quality, readily available, and have the best developer support for computational control. Papier-Mâché calls native Windows code for controlling these Canon digital cameras (the binding of native code to Java code is accomplished through the Java Native Interface [236]); and for communicating with the Phidgets RFID tags (this is accomplished through the IBM Bridge2Java system [12], which is a tool that automatically generates JNI stubs for ActiveX objects).

8.2.1 Event generation

Once the developer has selected an input source, Papier-Mâché generates events representing the addition, updating, and removal of objects from a sensor's view. Event types are consistent across all technologies. Providing high-level events substantially lowers the application development threshold and facilitates technology portability. The class responsible for this event generation is the *PhobProducer* (see FIGURE 8.3, top row). *PhobProducer* is an abstract class—it contains the event dispatch mechanisms and maintains the set of objects currently in a sensor's view, but not the techniques for interpreting and packaging input from an *InputDevice*. These techniques are delegated to the subclasses (see FIGURE 8.3, bottom row). There is a 1:1 mapping between *InputDevice* instances and *PhobProducers* instances. The separation of input acquisition from event dispatch is an important one.

The bottom level of the InputDevice hierarchy—the library classes that wrap

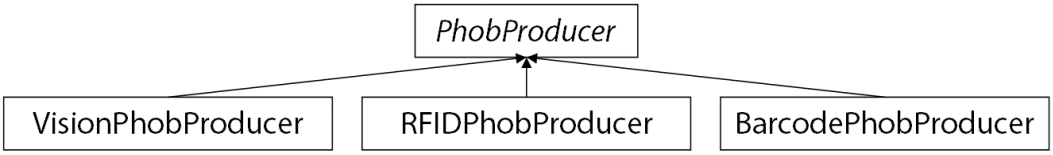


FIGURE 8.3 The inheritance hierarchy for PhobProducers. Producers are paired with InputDevices; they take input from a device and generate PhobEvents. The abstract PhobProducer base class manages the event listeners and the production of events.

particular types of input devices—can be seen as high-level device drivers in the tradition of UIMS systems. It is quite possible, as with mouse-based UIMS tools, that improvements in the device driver space will minimize the need for the bottom level of the *InputDevice* hierarchy. (The top two levels of the hierarchy comprise the Papier-Mâché architecture and will continue to be needed.) The RFID world is headed in this direction; both the ISO [14] and EPCglobal [8] are in the process of creating standards that cover the "air interface" (how tags and readers communicate), and industry appears to be slowly headed towards an XML standard for how RFID hardware and applications communicate.

While all technologies fire the same *events*, different technologies provide different *types of information* about the physical objects they sense. RFID provides only the tag and reader IDs. Vision provides much more information: the size, location, orientation, bounding box, and mean color of objects. Size, location, and orientation are computed using image moments [84]. Because this set is commonly useful, but not exhaustive, *VisionPhob*s support extensibility: each stores a reference to the image the object was found in. Application developers can use this for additional processing. Barcodes contain their ID, their type (EAN, PDF417, or CyberCode [206]), and a reference to the *VisionPhob* containing the barcode image. The *BarcodePhob* class includes an accessor method that returns this *VisionPhob*, allowing developers to access all of its information, such as location and orientation.

8.2.2 RFID events

Generating RFID events requires minimal inference. Each reader provides events about tags currently placed within range. We currently use Phidgets [99] RFID readers, which sense only one tag at a time. The inclusion of Phidgets demonstrates the architecture's ability to handle RFID tags and enables users to rapidly develop RFID-based interfaces. If

a final implementation required a particular brand of RFID (or a reader that supported simultaneous tag reads), it would be fairly straightforward for a developer to add an additional RFID library element.

When a tag is placed within range of a reader, Papier-Mâché generates a *phobAdded* event. Each subsequent sensing of the same tag generates a *phobUpdated* event. If the reader does not report a tag's presence within a certain amount of time, Papier-Mâché infers that the tag has been removed, generating a *phobRemoved* event. This inference technique was introduced by Want *et al.* [255]. RFID events contain both the tag ID and the reader ID. Applications can use either or both of these pieces of information to determine application behavior.

8.2.3 Vision events

Generating vision events requires much more interpretation of the input. Image analysis in Papier-Mâché has three phases: 1) camera calibration, 2) image segmentation, and 3) event creation and dispatching. Additionally, each of these processing steps can be overridden by application developers if they are so inclined. The contribution of the Papier-Mâché research is not in the domain of recognition algorithms; the vision techniques we use are drawn from the literature. The Papier-Mâché contribution here is a software architecture that provides A) a high-level API for the use of computer vision in the user interface so that non-vision experts can build vision-based interfaces, and B) provides a separation of concerns between UI design and algorithm design.

We have implemented camera calibration using perspective correction—an efficient method that most contemporary graphics hardware, and the JAI library, provide as a primitive. More computationally expensive and precise methods exist, see [83, CHAPTERS 1–3] for an excellent overview of the theory and methods.

The segmentation step partitions an image into objects and background; see [83, chapters 14–16] for an overview of image segmentation. There are two broad classes of segmentation techniques: stateless techniques that process images individually without regard to earlier information, and stateful techniques that take into account prior information. The Papier-Mâché library includes an example of each category of *SegmentationTechnique*: edge detection is stateless and background subtraction is stateful. Vision developers can create additional techniques by implementing the *SegmentationTechnique* interface. Each segmentation technique takes as input a raw camera image and generates a bi-level image where white pixels represent object boundaries and all other pixels are black. We will now discuss both of the library's techniques.

Edge detection

Edge detection is a segmentation technique inspired by the low-level human visual system, and was suggested as a potential computer vision algorithm by Marr [167]. Canny developed an edge detection method and demonstrated its optimality [48, 49]. Nearly all current edge detection implementations use the Canny technique, including the JAI library that Papier-Mâché uses. Edge detection is a stateless segmentation technique. In each frame, the raw image is processed for "edges": edges divide an image into regions that correspond to abrupt changes in pixel values. The Collaborage reimplementation developed by Andy Kung (see the code in appendix D.3 and discussion in section 9.6.3) used edge detection to find the 2D barcode glyphs that individuals moved on a wall to indicate whether they were in or out.

While the Canny method is an optimal edge detector, and sufficient for most TUIs, there are applications for which edges are not an appropriate segmentation technique. For example, segmenting an image containing a Zebra (an animal with black and white

stripes) using edge information will not yield a useful result. While human-quality segmentation is still an open problem in computer vision, substantial strides have been made. (See Carson *et al.'s* BlobWorld system [51] for an example of the current state of the art). Papier-Mâché would certainly benefit from improved techniques, both existing and future. However, the included techniques are sufficient for many TUIs, and more importantly, the basic object information that is used to compute behavior is largely consistent across segmentation techniques. More sophisticated applications that require specific domain information (*e.g.*, the species of an animal) could accomplish this by extending the *VisionEvent* class. For all of our inspiring applications, this would not be necessary.

Background subtraction

Background subtraction, the other technique included in the Papier-Mâché library, is an example of a stateful segmentation technique. At a high level, background subtraction works by comparing the current camera image to a prior camera image or an aggregate of prior images. The theory behind this technique is that the constant portions of an image represent the background (for example, the wooden surface of a desk) and the changed portions of an image represent the foreground (for example, documents placed on the desk). In practice, comparing against an aggregate performs better than comparing against a static background image because aggregates enable the inclusion of slowly changing information as background information (for example, as the sun moves across the sky the light on the desk changes). A standard technique for creating an aggregate image is to use an exponential weight moving average (EWMA) filter.

At each time step **t**, the current image I_t is subtracted from the aggregate image I_A; the resulting difference image I_{A-t} represents the change in the scene. A new aggregate

$I_{A'}$ is then computed by a weighted addition of the current image I_t with the earlier aggregate I_A: the current image is given weight α and the earlier aggregate is given weight $(1 - \alpha)$; the value of α is between 0 and 1. The technique is called "exponential weight" because this recursive addition is equivalent to the summation $\sum_{n=0}^{t}(I_n \times \alpha^{t-n})$. The fraction or "weight" of each individual image I_n is α raised to the exponent $t - n$; the weight of old images approaches 0 in the limit.

The Designers' Outpost used an EWMA-based background subtraction technique (see SECTION 3.8.2). Lederer and Heer developed the initial version of Papier-Mâché's background subtraction code for ceiling-mounted camera tracking of individuals in an office for their All Together Now system [147] (see SECTION 9.5.3).

EWMA techniques show up in many domains; one of the most common uses is for the prediction of file download times. The displayed remaining download time is the average rate multiplied by the remaining amount of data to be downloaded. This average rate is updated at each time step by factoring in the current rate. File download times are often seeded with a rate of 0 so that the prediction is conservative.

Labeled foreground pixels are grouped into objects (segments) using the connected components algorithm [111]. We create a *VisionPhob* class for each detected object. At each time step, the vision system fires a *phobAdded* event for new objects, a *phobUpdated* event for previously seen objects, and a *phobRemoved* event when objects are removed from view.

8.3 Declaratively Associating Input with Behavior

Papier-Mâché provides three levels of abstraction for handling behaviors associated with the objects it detects. 1) *PhobEvent* instances carry information about objects detected by a *PhobProducer*. 2) *AssociationFactory* instances provide a mechanism for creating and modifying application logic (*AssociationElts*). 3) The *BindingManager* is

built using the two previous primitives: it receives all *PhobEvents* from all *PhobProducers* and uses *AssociationFactory* instances to create *AssociationElts*. We discuss each of these architectural abstractions in turn.

8.3.1 Events

Events are the basic input dispatch primitive: at this level, developers manually instantiate producers and devices, register themselves as listeners, and receive the events that these producers generate. All application logic, and the relationships between input and application logic must be programmed manually.

8.3.2 Factories

A factory [86, p. 87-96] is a design pattern that provides an interface for creating families of related objects. In Papier-Mâché, an *AssociationFactory* (see figure 8.4) creates *AssociationElts*. The *AssociationFactory* is an interface that contains one method: *createAssociationEltForPhob(Phob phob)*; it creates an *AssociationElt* and returns it. In some cases, the *AssociationElt* created is parameterized by capture from the current environment. An example of this type is the marble answering machine, where each *AudioClip* created records an audio clip from a microphone. In other cases the created *AssociationElt* is parameterized by the properties of the *Phob* passed in. All of the inspiring spatial tuis require this behavior. These systems use the location (and often the orientation) of the physical object to control spatial aspects of graphical objects. In the remaining cases, when an *AssociationElt* is created, it prompts the user with a dialog box to specify its parameters. The WebStickers system for using barcodes as physical hyperlinks, if written with Papier-Mâché, might pop up a dialog box asking the user to specify a url. Some of the factories (such as the *AudioClipFactory*, *MediaFactory*, and *StringFactory* provided in the Papier-Mâché library) are agnostic to the type of input

device that created the *Phob*. Others require that a particular type of information be available in the *Phob*. The *VisualAnalogueFactory* requires a *VisionPhob*, as it uses the location and orientation of the *Phob*. While the creation and invocation of behaviors is handled through the factory, the management of multiple devices and/or multiple behaviors must be handled manually by a developer.

8.3.3 Bindings

In section 7.6, we introduced the technique of declaratively authoring application behaviors by binding a specific set of physical input to a particular piece of application logic. This need inspired the *BindingManager* class in Papier-Mâché. The binding manager automatically registers itself with all available *PhobProducers*, manages the flow of events, and automatically creates behaviors. The binding manager contains {classifier, behavior} pairs and it is the recipient of these events. It invokes application behavior for each element the classifier matches. Developers select *PhobProducer*(s) that will create input events, *ObjectClassifier*(s) that select a subset of generated input, and *AssociationElt*(s) that the factory should create.

The *BindingManager* contains a map data structure that maintains past and present bindings and creates new bindings in response to physical input. The manager listens for new *PhobEvents*. When a new *Phob* is seen, or when a *Phob* is updated, the

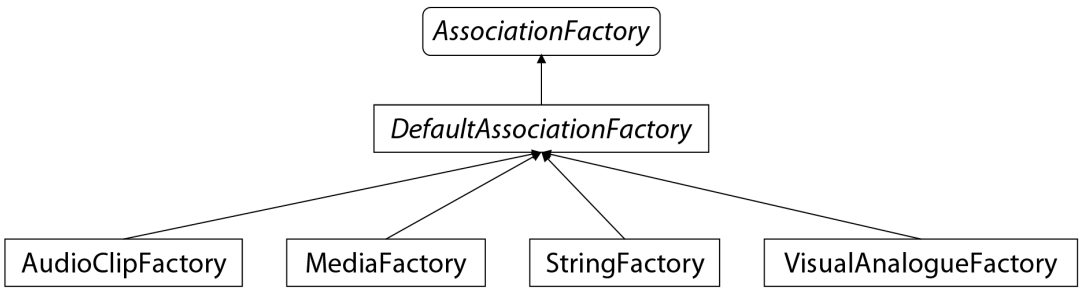


FIGURE 8.4 The inheritance hierarchy for factories: objects that create *AssociationElts* from *Phob* input. The top level is the *AssociationFactory* interface. The middle level is the *DefaultAssociationFactory* abstract class; this class provides the ability to be *VisuallyAuthorable* and the ability to serialize to XML using JAXB.

PhobProducer fires an event with the *Phob* as its payload. The *BindingManager* receives this event, and compares it to the table of classifiers (such as a *MeanColorClassifier* that selects for objects of a certain color).

Here, the developers' primary goal is instantiating and parameterizing classifiers and behaviors. Many of our inspiring applications can be created solely by parameterizing existing library classes. This parameterization-based approach is inspired by Interactors [182], and lends itself to visual programming [52]. We have built a visual programming tool that enables this (see section 8.8). When more complex functionality is required, developers can implement their own application logic by creating a custom behavior.

We will now illustrate how developers employ this declarative programming style in Papier-Mâché, using the In/Out board as an example. In this system, a barcode id represents a person, and its location represents whether they are *in* or *out*. Developers author these representation mappings by implementing a *BehaviorFactory*, which listens to events from the input sources. The example in section 9.3 and the Books with Voices rewrite presented in appendix D.2 both demonstrate applications that use the *BindingManager*. The factory receives a callback to create a new *AssociationElt* (see figure 8.5) representation instance (*e.g.*, a "person") for each new *Phob* created, and an update callback to modify that element's state (*e.g.*, whether they are in or out) each time a *Phob* is updated.

Each *AssociationElt* either represents a particular piece of content (we call these *Nouns*) or they operate on a piece of content (we call these *Actions*). This distinction is also used in Fishkin's survey of tangible user interfaces [77]. Operationally, a *Noun* can be the selection focus of an application, while an *Action* controls the current selection focus. In the In/Out board, each person would be a *Noun*.

The Papier-Mâché library includes four common types of nouns and five common media manipulation actions (see figure 8.5). The *FileBrowser* wraps files and urls, the

ImageBrowser wraps images, and the *MediaClip* wraps audio and video files. All of the topological and associative applications can be built with these three nouns (with the exception of Paper Flight Strips [158], which requires air traffic control information). The fourth noun, *AssociationWrapper*, is more general purpose. It wraps any functionality that the developer provides. For example, an *AssociationWrapper* can wrap a *JPanel* or other graphical element. An *AssociationWrapper* would be used to wrap each person in the In/Out board. The five media manipulation actions in Papier-Mâché's library were also chosen because media manipulation operations cover a majority of the behavior of our inspiring applications. *FastForward*, *Pause*, and *Rewind* perform their respective action on a *MediaClip*. *RandomAccess* moves the current position of a *MediaClip* to a designated place. *Reset* moves the current position of a *MediaClip* to the beginning.

There are two reasons why actions are encapsulated as objects. Logically, the programming task for behaviors is to provide a binding between the application input and a behavior that operates on a pre-existing focus object. This description implies that a clear technique for doing this would be to use the binding manager to store a mapping between input and a pointer to a function. However, Java does not have function pointers. Function pointers can be simulated by using a reflection to query an object about its methods, and creating an object that encapsulates the desired method. Using

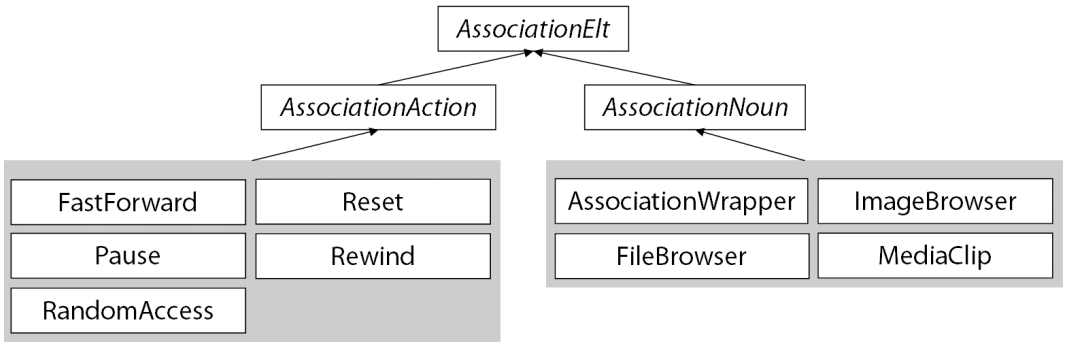


FIGURE 8.5 The inheritance hierarchy for associations. Associations are the elements in the Papier-Mâché architecture that input is bound to. These elements can either be nouns or actions. The Papier-Mâché library includes five common media manipulation actions, and four common types of nouns.

reflection for this purpose is both more complex and provides fewer runtime reliability guarantees than creating a set of separate action classes. Perhaps more importantly, action classes provide for using the same API for both objects and the actions that control their behavior.

8.4 Flow of Control in an Application

To recap, the basic flow of control in a Papier-Mâché application is that *PhobEvents* are created from a *PhobProducer*; the *BindingManager* contains a set of {*ObjectClassifier*, *AssociationFactory*} pairs; events matching a classifier are passed to the factory.

8.5 Switchable Classes of Underlying Technology

The Papier-Mâché library provides a software API to three classes of physical devices:

1 Electronic tags (*e.g.*, RFID tags)

2 Barcodes

3 Image Analysis of arbitrary physical objects

The first two involve manually tagging objects; before an object can be used with the system, it must be properly "suited up." The visual signature of an object can also be used as a tag (*e.g.*, using the content of Outpost notes as a tag signature), with the caveat that this higher-level recognition task may at times decrease robustness. The main benefit is that any object can be appropriated for use (see SECTION 2.9). The main drawback is that because these "tags" are human-generated, not machine-generated, there are no guarantees that the tag-space is well-partitioned, or even partitionable. Two blank notes in Outpost have the same signature, for example.

Image analysis deserves some comment because it is substantially more flexible than the other channels. In addition to being a recognition technology, cameras can be used as a capture technology. Papier-Mâché supports the use of vision for pure recognition,

for pure capture, and for using both together for structured capture, such as capturing the contents of recognized notes in the Designers' Outpost.

Image capture—acquiring input from an image source—is the first step in any vision system. Additionally, some TUIs use the raw capture as simply an image, and no further processing is required. The Peripheral Display Toolkit [171] is an example of a system that used Papier-Mâché for its flexible and low-threshold image acquisition API. This project's use of Papier-Mâché is described in further detail in SECTION 9.5.

One benefit of Papier-Mâché's vision architecture is that it provides a separation of concerns. Application developers can quickly develop a functional prototype using the provided libraries. Because the architecture is already provided, vision developers can work in parallel (or after the completion of the UI) to customize or replace the underlying vision algorithms as dictated by the domain.

8.6 How Papier-Mâché differs from a GUI Input Model

Papier-Mâché events have some similarities to GUI events, but they also differ in important ways. We will use the vision implementation of these events to illustrate this. Applications receive *VisionEvents* from an *ImageSourceManager* by registering *VisionListeners.* A *VisionListener* receives events about all objects larger than a specified minimum size. This minimum size constraint is solely for performance; it avoids an inappropriately large number of events from being generated. *VisionEvents* have a similar API to Java's *MouseEvents*, a descendant of the Interactors research. There are several important differences, however.

1 A mouse is a *temporally multiplexed* [81], generic input device; the meaning of its input is constructed entirely through the graphical display. In traditional GUIs there is always exactly one mouse (though some research systems have extended this, providing multiple mice). The behavior of moving the mouse or pressing a mouse button changes

over time, as a function of the mouse's position and the application state. In contrast, tangible interfaces nearly always employ multiple input devices, and these inputs are *spatially multiplexed*, as in the knobs and sliders of an audio control board or the pages of a book in Books with Voices (see CHAPTER 6). The audio board contains many knobs: each knob is always available and always performs the same function. In most TUIs, the functionality of an input object is conveyed by its physical form factor, markings on the object, and the object's location. In these systems, the input devices are lightweight; multiple objects appear and disappear frequently at runtime. While *MouseEvents* offer only position and button press updates, *VisionEvents* offer Add, Update, and Remove methods.

2 With a traditional mouse, the only input is (x, y) position and button presses. With physical objects on a plane, the captured information is position (x, y), orientation (θ), size, shape, and visual appearance. Papier-Mâché provides bounding box, edge pixel set, and major and minor axis lengths as shape information. It provides the mean color, as well as access to the source image data, for visual appearance.

3 While the position of a mouse and the state of its buttons is unambiguous, the information retrieved through computer vision is often uncertain. To address this, Papier-Mâché provides a lightweight form of classification ambiguity [166]. In Papier-Mâché, classifiers are responsible for reporting ambiguity; this is currently achieved through a scalar confidence value.

4 Similarly, with computer vision, the raw input (a camera image) contains a richness unavailable in the high-level events. These high-level events are an appropriate match for most of a developer's goals, but there are two cases where access to the original source data is beneficial: when a developer would like to conduct additional processing beyond object detection (such as recognizing an object as a unique instance, rather than

simply a member of a class) or when a developer would like to capture the raw image data for subsequent display. To accommodate this, Papier-Mâché provides access to the original pixel data along with the region of interest (ROI) that the object was located in.

8.7 Program Monitoring: Application State Display

In addition to the Java programming API, Papier-Mâché provides application developers monitoring facilities (see FIGURE 8.6). It displays the current input objects, image input and processing, and behaviors being created or invoked through the binding manager.

8.7.1 Current objects and vision I/O

At the left-hand side of the monitoring window, Papier-Mâché displays a three-level tree. This allows developers to see the current state of the system. The top level presents

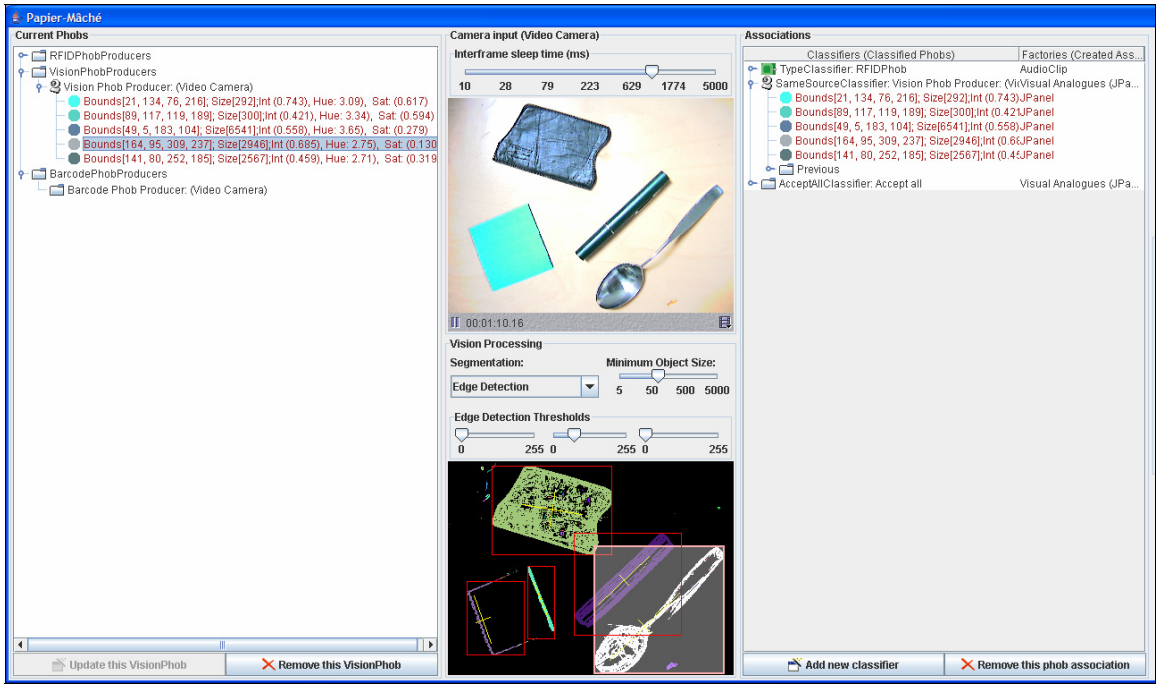


FIGURE 8.6 The monitoring window. In the 1st column, each current object appears in the hierarchy beneath the producer that sensed it. The 2nd column displays the vision input and output. The 3rd column displays classifiers (in this figure, RFID tags are associated with audio clips, and vision objects with graphical analogues).

the *PhobProducer* types. These are the broad input classes: RFID, vision, and barcode. The second level presents the *PhobProducer* instances. These are instances of objects creating input events; each instance is listed beneath its type. The bottom level presents the currently visible *Phob*s. Each *Phob* appears in the hierarchy beneath the producer that sensed it. The *Phob* displays a summary of its properties; *VisionPhob*s also have a circular icon showing their mean color.

Raw camera input is displayed at the top of the second pane. At the bottom of the second pane is the processed image; it displays each object's outline, bounding box, and orientation axis. Clicking on an object in either the "Current Phobs" view or the vision view highlights it in both views.

8.7.2 Wizard of Oz control

Papier-Mâché provides the richest Wizard of Oz (WOZ) input generation and removal of any tools for tangible interaction. This control is provided by the *add* and *remove* buttons at the bottom-left of the monitoring window. The functionality of these buttons change based on the selection in the left-hand column of the window. Selecting a producer type (the top-level of the hierarchy) and pressing the add button creates a new *PhobProducer*. The system queries the available devices, and presents a dialog box allowing the user to create a producer that uses real input or a "fake" producer that will be controlled by the user's WOZ input. When a producer is selected, the remove button will remove it from the system, and the add button will create a new *Phob* with the selected producer as the source. For example, with computer vision, selecting a *VisionPhobProducer* and pressing *add* generates a *Phob* with a reference to the camera's current image. When a *Phob* is selected, it can be removed by pressing remove, and its information can be updated by pressing update. In all cases, the created events appear exactly the same as if they had come from the sensor. This WOZ control is useful when

hardware is not available, and for reproducing scenarios during development and debugging.

For example, Andy Kung developed the Collaborage rewrite on this laptop, and he did not always bring a camera with him. He saved several camera frames of the whiteboard in different states onto his laptop, and this allowed him to test the application when the camera was not connected to his computer. It also allowed him to repeatedly test the same exact scenarios so that he could verify that his code was functioning correctly.

Papier-Mâché offers developers control over two axes that directly impact performance. The first is the time that the image processing thread should sleep between processing of images. Developers modify this through the slider at the top of the middle column of FIGURE 8.6 (interframe sleep time). Applications requiring interactive feedback, such as Outpost, would benefit from a short sleep time (five or ten milliseconds); applications where there is no interactive feedback could reduce processor load by opting for a longer sleep time (perhaps half a second). The second choice developers must make is the minimum size of objects. This choice helps limit a flood of events about "objects" that may simply be noise in the image sensor. The slider in the middle of the monitoring window controls this parameter (minimum object size in pixels). To aid developers in making this choice, the size of currently detected objects is listed with each object on the left-hand panel and the size of all objects that match a classification is displayed with each object on the right-hand panel. Developers then choose a value that is safely below the smallest item.

8.8 Visually Authoring and Modifying Application Behavior

Papier-Mâché's graphical interfaces provides both monitoring information and authoring information. Section 8.7 described the monitoring uses of this system. This section describes the authoring functionality. Interacting with the visual authoring facilities allows developers to create and modify the application's runtime code. Papier-Mâché uses XML as a persistent representation of the visually authored program. This is accomplished via JAXB, a tool for serializing Java objects [16].

To create new behaviors, developers select the technology that will provide the input on the left-hand side of the monitoring window and press the "add new classifier" button in the lower-right of the monitoring window. This invokes a dialog box to create a new binding between a class of physical input and an application behavior (see Figure 8.7). A developer selects an input classifier on the left-hand side of the dialog; the set of available classifiers is based on the technology selected in the monitoring window. The developer then selects the type of application behavior that should be created on the right-hand side. The list comprises the set of all *AssociationElt* objects that have registered themselves with the monitoring window. By default, this is the five

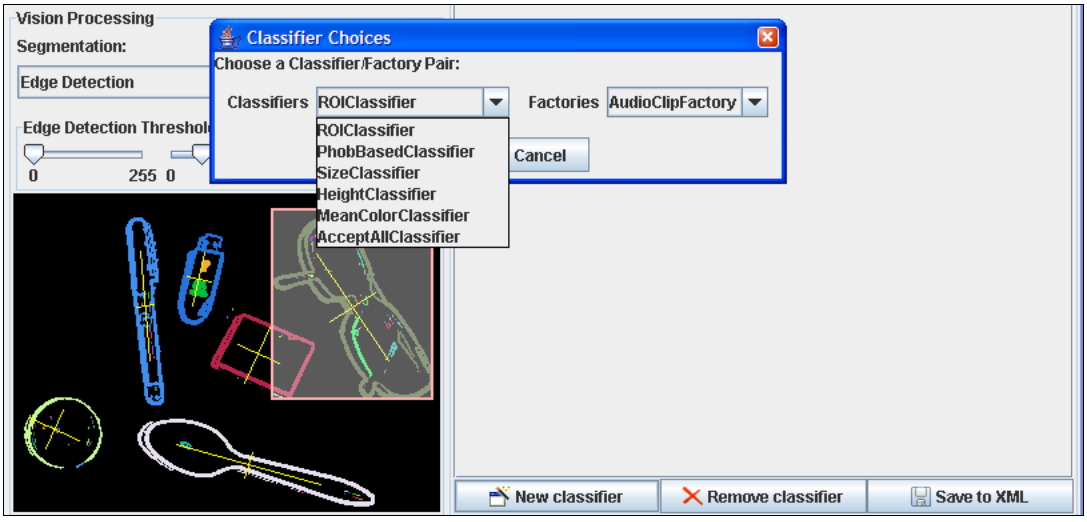


FIGURE 8.7 The dialog box for creating a new binding between input and behavior.

AssociationActions and four *AssociationNouns* that the Papier-Mâché library provides.

Developers can implement their own *AssociationElts* and register them to provide custom behaviors. Developers then specify the parameters for the physical input they are interested in; one of these dialogs—for finding objects of a specified color—is shown in FIGURE 8.8. The parameter specification dialogs were created by De Guzman and Ramírez [61]. Each dialog provides a graphical user interface where developers specify each of the classifiers parameters; visual feedback about the currently specified class is presented in the lower-left. The parameters of the classifier are initially set to the currently selected *Phob* in the monitoring window. For example, when the developer begins creating a new classifier in FIGURE 8.7, a pair of olive-brown sunglasses is selected. Upon selecting *MeanColorClassifier*, the corresponding dialog appears with olive-brown as the selected color (see FIGURE 8.8). This color-based classifier offers control over the mean color and the tolerance, specifying the span of colors to include.

In the future, it may be more appropriate to have this value selected automatically by the system using techniques similar to Crayons [75].

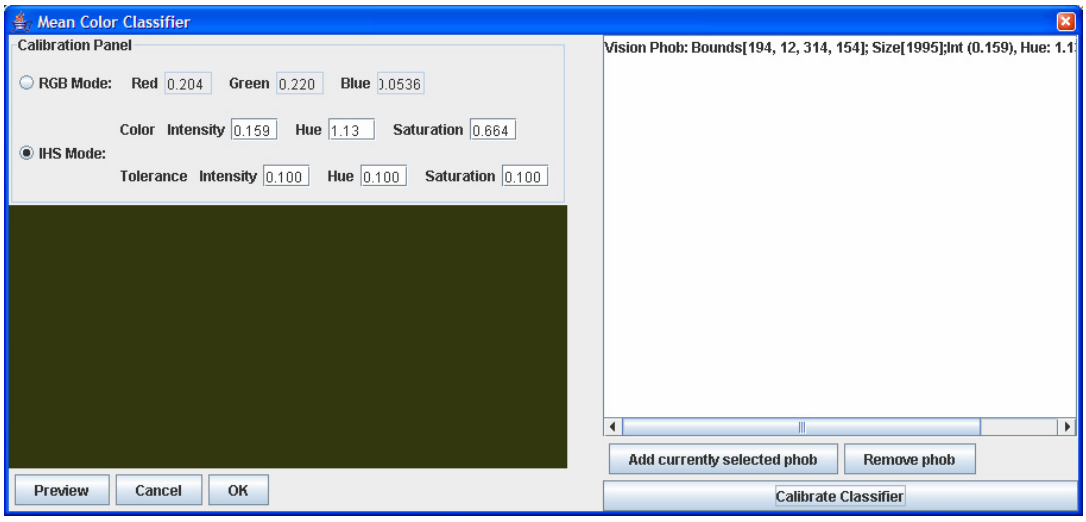


FIGURE 8.8 A dialog box where developers specify the color of objects of interest. Dialog box designed by De Guzman and Ramírez [61].

8.9 Summary

The Papier-Mâché architecture presented in this chapter provides a low threshold development tool for tangible user interfaces. This architecture provides high-level events for input technologies such as computer vision, and separates the acquisition of device input from the interpretation of device input. This modularity enables different members of an equivalent device class (such as cameras with different APIs) to use the same interpretation framework. It also enables vision developers to create different algorithms that perform the image interpretation task. The application receives information about this interpreted input through events. The event architecture is equivalent across all technologies; making it possible to rapidly explore different input technology alternatives. The binding manager contains {classifier, behavior} pairs and it is the recipient of these events. It invokes application behavior for each element the classifier matches. This manager facilitates multiplexed input, and, as with all elements of Papier-Mâché, it is instrumented such that its relevant information appears in the monitoring window. The monitoring window provides feedback about application behavior: input, how input is interpreted, and what behaviors are created. This Papier-Mâché architecture successfully meets all of the design goals articulated at the beginning of the chapter.

9 Papier-Mâché Evaluation

> *"Millions for compilers, but hardly a penny for understanding human programming language use. Now, programming languages are obviously symmetrical, the computer on one side, the programmer on the other. In an appropriate science of computer languages, one would expect that half the effort would be on the computer side, understanding how to translate the languages into executable form, and half on the human side, understanding how to design languages that are easy or productive to use."*
>
> —Alan Newell, 1985 [190].

In this chapter, we present the results of our evaluations of the Papier-Mâché architecture. The UI tools, psychology of programming, and empirical studies of programmers communities have demonstrated the utility of several types of evaluation. Through the application of these methods, this chapter contributes an understanding of the Papier-Mâché architecture and design considerations for TUI software architectures more generally. Additionally, this dissertation is the first work to "triangulate" the design and usability issues of a software API by employing these methods in concert. Through the findings of these studies, we demonstrate in this chapter that different methods yield different types of results, and composing the results of different techniques yields a fuller picture of the use issues of a software API.

Portions of this chapter were originally published by the author, Jack Li, James Lin, and James A. Landay in [133].

9.1 Overview of Evaluation Methods

While there has certainly been prior work on evaluating software tools, this area is much more limited than might be expected, perhaps because, as Détienne writes, "The dominant problems have been perceived as technical rather than as related to the usability of the systems. The introspective approach, which is the common approach in this field, carries the illusion that usability problems are automatically handled: tool developers will use their own experience as the basis for judging the usefulness of the tools they develop" [62, p. 118].

The design of the Papier-Mâché architecture was informed by our application of three methods:

- literature review (see section 2.8)
- more than three years of experiential knowledge of the domain (see chapters 3–6)
- structured interviews with nine tui developers (see chapter 7)

In building the Designers' Outpost and Books with Voices, we learned from our own experience. We learned from "explorers" in tangible interfaces through our literature review and interviews. We learned the scope of research in tangible interfaces, that there were substantial commonalities and repeated development effort across projects, and that researchers either tried or were curious about exploring different input technology options for their projects.

The resulting Papier-Mâché architecture was then evaluated using:

- performance metrics (see section 9.2)
- code metrics (see section 9.3)
- a laboratory study with seven participants (see section 9.4)
- analysis of longitudinal use by nine projects (see section 9.5).
- reimplementing three inspiring applications (see section 9.6)

The performance metrics assess how well the Papier-Mâché architecture meets the performance needs of intended applications. The code metrics provide an understanding of the complexity involved in building an application.

A laboratory study helped us understand the novice use of Papier-Mâché in a controlled setting. The results of this study demonstrate that even first-time users could build tangible interfaces and easily adapt applications to another technology. Testing with novices users provides a lot of usability information, such as the understandability of class names, the quality of documentation, and where the system model is different than users' initial conceptual model [197, ch. 1].

We also examined how application developers used Papier-Mâché in their own work. This technique has the opposite set of trade-offs from a laboratory study. The developers chose their own tasks, offering a realism and breadth unavailable in the laboratory. The time-scale was much longer, ranging from one week to several months. However, it is difficult to directly compare results between projects precisely because they are all different.

A difficulty of our fieldwork is that the researchers we interviewed were technology experts in their area. One of the goals of the Papier-Mâché architecture was to open up tui development to a larger community. The laboratory study and project use overcome this difficulty by evaluating a developer's first experience with programming (in the laboratory), and their longer use of the tool (in the developers' own applications).

9.2 Performance

On contemporary hardware, Papier-Mâché runs at interactive rates. On a dual Pentium iii computer running Windows xp, the vision system runs at 5.0 frames per second without monitoring and 4.5 fps with monitoring, at a cpu load of 80%. With the vision system and two rfid readers, the performance is 3.0 fps. The performance is more than

sufficient for *forms* and *associative* applications, and sufficient for *topological* and *spatial* applications with discrete events. There are two reasons why associative applications have weaker performance constraints. First, associative apps generally work as "physical hyperlinks," and a few hundred milliseconds latency is not an issue for this goal. Second, the input and output is not generally geo-referenced and there is no continuous motion in these systems, so there is no need to very rapidly update the electronic information. Where tangible input provides a continuous, interactive control, current performance may be acceptable in some cases, but a minimum of 10 fps is required for these controls to feel truly interactive [50]. Of the 24 applications we surveyed, five required this continuous manipulation. Tuis built with tethered, 3d tracking systems [81, 108] (which are outside the application space that Papier-Mâché supports) also require this higher level of performance.

The vast majority of this computation time is in the image processing code. While our code is reasonably optimal, Java is not a language known for its speed. The jai architecture partially addresses the traditional performance limitations of Java: jai is released as both a pure-Java cross-platform addition and with platform specific performance packs. At runtime, image manipulation operations use the native performance pack code if it is available, and use cross-platform code otherwise. Porting Papier-Mâché to Microsoft's C# language would retain the benefits of the Papier-Mâché architecture and programming using managed code (*e.g.*, garbage collection and security), and gain a significant performance increase. Jai's performance attempts are reasonable, but it is not comparable to all native code. The drawback to a native code approach is that it would not be platform independent. Needless to say, most machines sold today are faster than the dual-processor Pentium iii used for these benchmarks, and thus performance is unlikely to be an issue for even continuous interactive control.

9.3 Lowering the Threshold: A Simple Application

In addition to measuring how rapidly applications built with the toolkit execute, it is important to measure how rapidly a developer can build an application. A metric of the threshold for using a toolkit is the number of lines of code required by a basic application. The following Java code comprises the complete source for a simple application that graphically displays the objects found by the vision system. It is only four lines of code, three of which are constructor calls.

Have the vision system generate objects from camera input.

```
1 PhobProducer prod = new VisionPhobProducer (new
    CameraImageInput());
```

Create a factory that associates each object seen by the camera with a JPanel. The factory creates a JPanel for each object seen and adds the JPanel to the specified window.

```
2 AssociationFactory factory = new VisualAnalogueFactory(new
    PMacheWindow(prod, CALIBRATE), JPanel.class);
```

Create a binding manager that will receive events; this map contains the factory.

```
3 BindingManager bindingMgr = new AssociationMap(factory);
```

Attach the binding map to the camera, which will create, update, and remove JPanels according to what the camera sees.

```
4 prod.addPhobListener(bindingMgr);
```

This simple example illustrates that the threshold for creating an application with Papier-Mâché is quite low. The applications built in sections 9.4 to 9.6 demonstrate the small code size for more involved TUI applications.

9.4 In-lab Evaluation

We conducted a controlled evaluation of Papier-Mâché to learn about the usefulness of our input abstractions, event layer, and monitoring window. Seven graduate students in our university's computer science department participated in the study: one in graphics, three in programming languages, two in systems, and one in AI. (We excluded HCI students due to potential conflicts of interest, and theory students because their background is less appropriate.) All participants had experience programming in Java.

9.4.1 Method

We began each evaluation session by demonstrating an application associating RFID tags with audio clips, including an explanation of the monitoring window. We then asked the participant to read a seven page user manual introducing the toolkit (see APPENDIX E). Next, we gave participants a warm-up task and two full tasks. The evaluation was conducted in our lab on a 400 MHz dual Pentium II running Windows XP with the Eclipse 2.1.1 IDE. We verbally answered questions about Java and Eclipse; for toolkit questions we referred participants to the user manual and online Javadoc. We asked participants to "think aloud" about what they were doing, and we videotaped the sessions and saved participants' Java code for further review.

The warm-up task was to change an application that finds red objects so that it finds blue objects. The first full task was to change an In/Out board written using computer vision to use RFID tags instead. The second full task was to write an application that used RFID tags to control a slideshow. One tag represented a directory of images; the two other tags represented *next* and *previous* operations.

9.4.2 Results

Every participant completed every task, though not without moments of difficulty. We take this to be a success of the API. In our first task, participants converted an In/Out board from vision to RFID in a mean time of 31 minutes using a mean of 19 lines of code (see TABLE 9.1). This shows that technology portability is quite achievable with Papier-Mâché.

Participants appreciated the ease with which input could be handled. In addition to their verbal enthusiasm, we noted that no one spent time looking up how to connect to hardware, how input was recognized, or how events were generated. In our second task, participants authored an RFID-based image browser in a mean time of 33 minutes using a mean of 38 lines of code. Note that participants on average wrote code twice as fast in the second task as in the first, indicating that they quickly became familiar with the toolkit. Two of the participants directly copied code; one said, "So this is like the marble answering machine [in the user manual]."

Ironically, the warm-up task—changing a colored-object finder from red to blue—proved to be the most challenging. The problem was that the classifier took a color parameter represented in the intensity-hue-saturation (IHS) color space, highly effective for image analysis but not intuitive to most computer scientists, who are used to the RGB color space. Participants had difficulty even though we explained that the color space

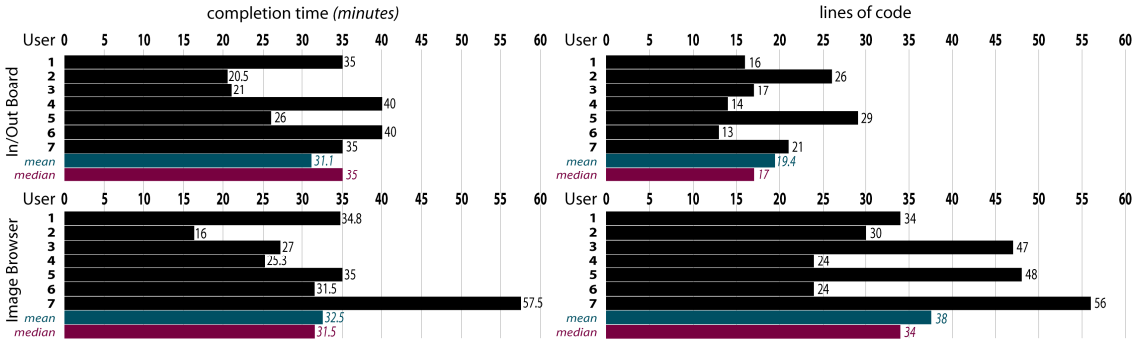


TABLE 9.1 The task completion times and lines of code for the seven users in the Papier-Mâché laboratory study.

was IHS, not RGB. Once a color in the proper color space was found, it took less than a minute to make the change. Ideally, these parameters should not be specified textually at all. These results inspired our investigation of visual authoring tools (see SECTION 8.7).

Overall, participants found the monitoring window to be very useful. For the warm-up task, they used it to understand the (confusing) color classifier. For the In/Out board task, they used the monitoring window to get information about the attached RFID readers. When participants had errors in their code, they also used the monitoring window to verify that the input was not the source of these errors.

We also uncovered several usability issues. The most glaring was an inconsistency in naming related elements: the superclass was named *PhobGenerator*, a subclass *RFIDReader*, and the accessor method *getSource*. The term *generator* is also inconsistent with how similar classes in the Java library are named (Java uses the term *producer* for similar classes). We addressed these issues by renaming the abstract superclass *PhobProducer*, the subclass *RFIDPhobProducer*, and the accessor method *getProducer*(). Other points of confusion highlighted places where our documentation was insufficient. We have since addressed these usability issues by improving the API, documentation, and method names based on the feedback from this study.

9.5 Applications Using Papier-Mâché

A more open-ended, longitudinal evaluation of Papier-Mâché was conducted by observing its use in class and research projects at UC Berkeley. In addition to providing valuable feedback about the toolkit, the availability of a low-threshold toolkit benefited the students and researchers to conduct research that otherwise would not have been possible. Between February 2003 and May 2004, nine groups of graduate and undergraduate students used Papier-Mâché for their class and research projects: two groups

in a graduate HCI course in the Spring of 2003 (SECTION 9.5.1), four groups in a graduate ubiquitous computing course in the Fall of 2003 (SECTION 9.5.2), and three other groups in the 2003-2004 academic year (SECTION 9.5.3).

9.5.1 Spring 2003, graduate human-computer interaction

Two groups in the Spring 2003 offering of the graduate HCI class at UC Berkeley built projects using Papier-Mâché.

Physical Macros [60] (see FIGURE 9.1) is a *topological* TUI for programming macros, such as "actions" in Adobe Photoshop. In this system, users compose physical function blocks that represent image editing functions. When examining their code, we found that presenting geo-referenced visual feedback was a substantial portion of the code. Reflecting on this, we realized that many of our inspiring applications, including The Designers' Outpost (see CHAPTERS 3, 4, and 5), also require this feature. For this reason,

FIGURE 9.1 The Physical Macros class project: a wall-scale, topological TUI. At *left*, a set of physical operation cards placed on the SMART Board; the resize operator is parameterized with an electronic slider. At *top right*, the image resulting from the operations. At *bottom right*, the set of physical operation cards available to the user.

we introduced bindings where the location of physical input and electronic output could be coordinated.

SiteView [33] (see FIGURE 9.2) is a *spatial* TUI for controlling home automation systems. On a floor plan of a room, users create rules by manipulating physical icons representing conditions and actions. The system provides feedback about how rules will affect the environment by projecting photographs onto a vertical display. SiteView employs a ceiling-mounted camera to find the location and orientation of the thermostat and the light bulbs, and three RFID sensors for parameter input (weather, day of week, and time).

The thermostat is distinguished by size; the bulbs are distinguished by size and color. In general, the system worked well, but human hands were occasionally picked up. This inspired our addition of an event filter that removes objects in motion. With this in place, human hands do not interfere with recognition. SiteView is roughly 3000 lines of code; the input portion of the application is written in Papier-Mâché with only about 30 lines access. As a point of comparison, the Designers' Outpost (see CHAPTERS 3, 4, and 5) was built with OpenCV and required several thousand lines of vision code to

FIGURE 9.2 SiteView, a *spatial* UI for end-user control of home automation systems. *Left:* A physical light-bulb icon on the floor plan, with projected feedback above. *Right:* The six physical icons.

provide comparable functionality. We consider this substantial reduction in code to be a success of the API.

9.5.2 Fall 2003, ubiquitous computing

Four students in the Fall 2003 offering of a graduate course on ubiquitous computing at UC Berkeley [63] used Papier-Mâché for a one-week mini-project. The goals of the mini-projects were tracking laser pointers, capturing Post-it notes on a whiteboard, invoking behaviors such as launching a web browser or email reader, and reading product barcodes. The quotes in this section are drawn from the students' project reports.

These programmers were impressed with the ease of writing an application using Papier-Mâché. One student was amazed that, "It took only a single line of code to set up a working vision system!" Another student remarked, "Papier-Mâché had a clear, useful, and easy-to-understand API. The ease with which you could get a camera and basic object tracking set up was extremely nice."

The students also extended the toolkit in compelling ways. One student's extension to the monitoring system played a tone whenever an object was recognized, mapping the size of the recognized object to the tone's pitch. This provided lightweight monitoring feedback to the recognition process.

These projects also unearthed some shortcomings of the Papier-Mâché library's current vision algorithms. For example, the system tended to lose track of an object and then immediately find it again, causing the undesired firing of *phobRemoved* and *phobAdded* events. One student observed that vision algorithms are inherently ambiguous and requested better ways of dealing with the ambiguity. The vision requirements for our inspiring applications and for the projects created here can be reliably handled by contemporary techniques. The challenge is that these techniques are

more computationally intensive than the techniques currently included with the Papier-Mâché library, indicating that the Java language would probably not be appropriate. Porting Papier-Mâché to C# would remove the performance limitations of the Java virtual machine while retaining the benefits of a managed, object-oriented language. This port would enable us to use more sophisticated techniques. Additionally, Papier-Mâché should offer a richer model of ambiguity, and support techniques for mediating ambiguous input such as those introduced by Mankoff [165, 166].

9.5.3 Additional projects

Three other Berkeley projects have used Papier-Mâché. The first is De Guzman and Ramírez's ObjectClassifierViews [61]. This system provides a set of graphical user interface dialogs that allow users to create classifiers and modify their parameters (see figure 8.8). This work inspired us to integrate their code into Papier-Mâché and to provide a mechanism for saving applications created visually. This system, as integrated with Papier-Mâché, is discussed in section 8.8.

The second is Lederer and Heer's All Together Now [147] (see figure 9.3). All Together Now is an awareness tool: the locations of individuals in a space are captured

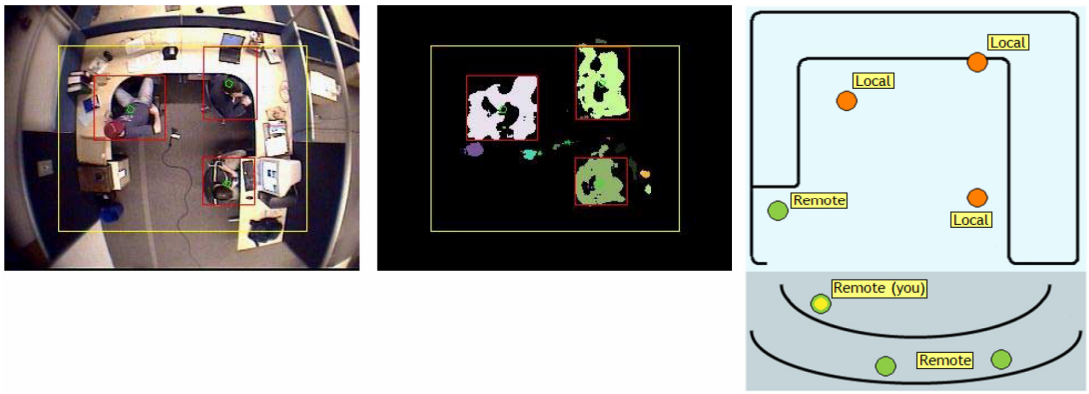


FIGURE 9.3 ATN captures a bird's-eye video feed of the physical space (*left*), locates people using computer vision (*middle*), and displays local actors' positions (orange) in a virtual space (*right*) shared with remote actors (green). Non-participating remote actors are placed in an observation deck. Each remote actor's circle is marked with a yellow core in his personal view. (Picture on right is annotated for grayscale printers). Image from [147].

through computer vision and presented abstractly on a web page. Remote individuals can "interact" with the local individuals by placing a marker of themselves on the space. Prior to All Together Now, the Papier-Mâché library only included edge detection, a stateless vision technique. The complexity of this scene and the low fidelity of the camera make stateless techniques impractical. Lederer and Heer implemented the background subtraction algorithm to overcome this. We incorporated their background subtraction code into the Papier-Mâché library. This experience showed us that it is possible for individuals interested in "getting under the hood" to change the vision algorithms used by Papier-Mâché, and that its overall architecture is modular enough to easily accommodate new algorithms.

The last application that used Papier-Mâché is Matthews *et al.*'s Peripheral Display Toolkit (PTK) [171]. The PTK lowers the threshold for developing peripheral displays—systems that provide ambient awareness of information (such as the fluctuation of a stock price) without being obtrusive. PTK uses the image acquisition portion of Papier-Mâché as one of its input sources; PTK then abstracts this input and renders aspects of the input to an ambient display. All of our vision-based inspiring applications use continuous image processing. PTK's needs are distinct in two ways: 1) it does not use the built-in processing, only the acquisition, as it does its own processing to find motion in images, and 2) it needs new images so sporadically that it is more appropriate to ask for them than to have them pushed at a regular interval. This use of Papier-Mâché demonstrates that the input acquisition and vision processing are sufficiently distinct that the former could be used without the latter. It also encouraged us to include the ability to request images, rather than enforcing an event-driven model.

9.6 Inspiring Applications Rewritten with Papier-Mâché

To complete the circle and validate Papier-Mâché's ability to build some of our inspiring applications, Jack Li and Andy Kung, undergraduates working with the author on Papier-Mâché, reimplemented three applications: the marble answering machine [204], Books with Voices (see CHAPTER 6), and Collaborage [180].

9.6.1 Marble Answering Machine

The marble answering machine is an associative TUI where physical marbles correspond to answering machine messages (see summary in SECTION 2.8.3). Bishop's prototype was never built. Jack Li's code for this application is presented in APPENDIX D.1. An elegant application, this is straightforward for a developer to create with Papier-Mâché. Excluding comments, white space, and imports, it comprises 18 lines of code. Note that this prototype does not communicate with the telephone system; it uses the audio system of a desktop PC. When an RFID tag is seen for the first time, the user records a message to it. Each subsequent time that a tag is seen, that recorded message is played back. Li also developed an alternate version that uses two readers: one reader designates recording, the other designates playback. This enables tag reuse. This marble answering machine implementation demonstrates a prototype that is more realistic than the original designer of the system was able to create.

9.6.2 Books with Voices

Books with Voices links physical transcripts to the recorded interviews they were derived from (see CHAPTER 6). Jack Li implemented two alternate versions of this application: one uses RFID tags and the other uses barcodes. This simplified version of the application handles the user interaction but not the document creation software. The code for both is presented in APPENDIX D.2. Excluding comments, white space,

and imports, the RFID version comprises 22 lines of code and the barcode version 30 lines. The difference between the two versions lies in the initialization. The barcode version uses barcodes detected in a camera image. The developer must specify in the initialization what camera they would like to use for input, and then connect that input to the *BarcodePhobProducer*. This minimal difference between versions of this application buttress the results of SECTION 9.4.2, demonstrating that Papier-Mâché facilitates easily retargeting applications.

9.6.3 Collaborage

The Collaborage application connects documents on walls with database information (see summary in SECTION 2.8.1). Andy Kung reimplemented a version of the Collaborage In/Out board, including connecting to a SQL database back-end. This example is the most complex of the three. It consists of three files; the code is included in APPENDIX D.3. *Run.java* is the primary application file; it contains 146 lines of code. *Network.java* provides the connection to the SQL database; it contains 136 lines of code. *Log.java* prints a time-stamped log as elements are shifted between "In" and "Out"; this file contains 81 lines of code. This example, along with the results of SECTION 9.5, illustrates that Papier-Mâché can be used to build more complex applications, and that it can be integrated with other tools that are needed to build these applications.

9.7 Summary

The results of the multiple evaluation techniques described in this chapter show that Papier-Mâché achieves its primary goals of 1) providing a low-threshold architecture for developing tangible interfaces, and 2) enables developers to rapidly switch input technologies. They also demonstrated that the monitoring facilities helped developers better understand application behavior. The longitudinal use of Papier-Mâché

demonstrates its ability to support novel TUI applications, and that its modularity separating input acquisition, input interpretation, and application behavior enabled developers to incorporate and/or modify each of these independently. This longitudinal use also found that more sophisticated vision algorithms would lower recognition errors. Increased transfer of technology from the computer vision community to the HCI community would greatly benefit the HCI community, and as interfaces such as those presented in this dissertation become more commonplace, TUIs could provide an important domain for computer vision researchers.

10 Conclusions and Future Work

This dissertation demonstrated that an event-based software architecture employing high-level input abstractions can lower the threshold for tangible user interface development. This architecture also supports switching input technologies with minimal code changes. These architectural contributions are embodied in the Papier-Mâché toolkit, which makes debugging easier through monitoring facilities that include Wizard of Oz control. Papier-Mâché is open-source software available at http://hci.stanford.edu/research/papier-mache. An important benefit of this low-threshold, flexible architecture is that it opens development in this area to a wider community and enables rapid, iterative design.

These architectural insights are grounded in our experience building two tangible interface applications: The Designers' Outpost and Books with Voices. These systems contributed novel interaction techniques for merging the whiteboards and books found in our physical world with benefits of the electronic world: documents that flexibly move between tools, that offer design history and remote collaboration, and that provide direct-manipulation hyperlinks to a richer media experience.

10.1 Contributions

In this section, we restate the contributions listed at the beginning of the dissertation and summarize how each of these contributions was achieved.

1 *Toolkit support for tangible user interface input. The Papier-Mâché toolkit introduces a novel software architecture that:*

A *Lowers the threshold for developing applications that employ tangible user interface input. This is accomplished with high-level abstractions of input technologies.*

Papier-Mâché provides input at the object level, rather than at the pixel or bits level; it also provides a library of interactive-rate computer vision algorithms and an architecture that separates the creation of additional algorithms (the domain of computer vision development) from the application (the domain of interface development). The evaluation results described in sections 9.3 to 9.6 demonstrate that the toolkit achieved this goal.

B *Supports switching input technologies with minimal code changes. The architecture structures input from all devices in a similar fashion.*

The Papier-Mâché architecture provides a common, event-based architecture for input technologies. The evaluation results described in sections 9.4, where novice TUI developers were able to migrate an application to a different technology, and 9.6.2, which present two variants of an inspiring application, demonstrate that the architecture achieved this goal.

C *Makes debugging easier through monitoring facilities that include Wizard of Oz control.*

The monitoring window provides a visual interface that displays raw and interpreted input, as well as application behaviors. The Papier-Mâché toolkit is instrumented so that this information is updated as the application executes. The laboratory evaluation results described in section 9.4 demonstrate that it achieved this goal.

2 *Interaction techniques that employ tangible user interface input to support professional work practices. The difficulty of implementing these applications inspired our research on toolkit support for tangible input.*

A *The Designers' Outpost integrates wall-scale, paper-based design practices with novel electronic tools to better support collaboration for early-phase design. This integration is especially helpful for fluidly transitioning to other design tools; access and exploration of design history; and remote collaboration.*

The Designers' Outpost system (see CHAPTERS 3, 4, and 5) was evaluated with 27 professional web site designers over a three-year period. The study of the initial interactive system (described in SECTION 3.5) demonstrated the utility of the tangible interaction model. The study described in SECTION 4.7 demonstrated the benefits of integrating design history. The study described in SECTION 5.6 demonstrated the benefits of integrating remote collaboration.

B *The Books with Voices system introduces an augmented paper UI providing fast, random access to digital video while retaining the paper-based transcripts preferred by oral historians.*

The Books with Voices system (see CHAPTER 6) was evaluated with 13 participants (see SECTION 6.6). The study found that this lightweight, structured access to original recordings offered substantial benefits with minimal overhead. The study showed that integrating recordings offered a level of emotion in the video not available in the printed transcript. The video also helped readers clarify the text and observe nonverbal cues.

3 *Improved user-centered methods for the design and evaluation of software tools.*

A *Fieldwork with developers as a basis for the design of software tools, in order to better learn what software developers are really doing and what tool support would be beneficial.*

The fieldwork described in CHAPTER 7 provided us with an important understanding of developers' successes, limitations, design practices and requirements for tools supporting tangible interaction.

B *A triangulation method comprising controlled laboratory study, monitoring of longer-term use in projects, and external metrics such as performance and code size for evaluating the usability of software APIs.*

Each of the evaluation methods that we employed provided different information about the usability of Papier-Mâché. The performance metrics (SECTION 9.2) demonstrate that the performance is sufficient for most of our inspiring applications. The code size metrics (SECTION 9.3) demonstrate the low development threshold. The laboratory study (SECTION 9.4) showed that first-time Papier-Mâché users could create and modify applications; it also allowed us to compare performance across users. Observing Papier-Mâché's use in others' work (SECTION 9.5) enabled us to understand longitudinal use of the toolkit, and its use in a wider variety of applications of the developers' own choosing. Lastly, reimplementing some of our inspiring applications (SECTION 9.6) demonstrated that the toolkit could indeed build the systems it was designed for. This application of multiple methods offers a much fuller picture of a system's usability.

10.2 Limitations

The Papier-Mâché toolkit attempts to satisfy three disparate groups of users: those interested in very rapid interaction prototyping, those interested in more detailed

interaction implementation, and those interested in computer vision algorithms. When architecture trade-offs forced us to privilege the needs of one of these groups, we chose those interested in interaction implementation. The results of CHAPTER 9 show that we were successful in this endeavor. Our work on visual programming shows that there is a space of applications that can be prototyped very rapidly through a visual UI. This UI has a very low threshold, but also a low ceiling. The difficulty with any prototyping tool that generates a different format than the programming language does is that there is a seam between the two. Visual programs written with Papier-Mâché cannot be easily extended with Java code. While some environments support both visual authoring and scripting (such as Macromedia Director [5]), there is much work to be done on more seamlessly integrating these two programming styles.

While prior work has largely concentrated on aiding the technologists (*e.g.*, Intel's OpenCV [40]), Papier-Mâché concentrates on supporting novel interaction design. This dissertation provides tools that enable a wider range of developers to create tangible interfaces. Additionally, the separation of concerns between vision development and interaction design in Papier-Mâché does indeed enable the two groups to work simultaneously. However, the tradeoff is that Java is an awkward language for computer vision. For those solely interested in prototyping vision algorithms, Matlab is a better choice, primarily because of its language-level support for computations involving arrays and matrices, and because of its extensive mathematical libraries. For those interested in writing production code, C and C++ are preferable because, while awkward for vision programming, they are fast at runtime. A general open problem in ubiquitous computing software tools is that the component pieces of these heterogeneous applications are best individually served by different programming models. One solution is to offer each community its ideal programming model, but this has the drawback that an individual must alternate between many languages and tools

to accomplish a single task or that applications can only be built by large groups. Balancing these design issues is an area of future research.

10.3 Future Work

This dissertation suggests several areas of future work. First, because the Papier-Mâché toolkit lowers the threshold for TUI development and makes development available to a broader range of users, there is increased opportunity for applications research, for creative designs, for technology transfer of TUI research, and for longitudinal deployment and evaluation of these systems.

Second, ubiquitous computing tools would benefit from tools that provide integrated support for design, evaluation, and analysis. Klemmer *et al.*'s SUEDE system for designing speech user interfaces [135] first introduced this integrated support. Ubicomp tools in this vein would support both Wizard of Oz evaluation and evaluation of functioning systems. Such tools could also provide visualizations illustrating what aspects of applications are actually used, the user and system performance aspects of these systems, and when recognition errors occur or when users have difficulty understanding the system (these could be flagged by the user or by an observer). Logging behavior over periods of extended behavior and visualizing that information is also an important area for future research.

Third, as discussed in SECTION 10.2, the heterogeneous technologies used in ubiquitous computing suggest research on improving the methods by which members of a design team collaborate through design tools. Tools should aid conversations by affording designers some understanding of the technical constraints of a system and technologists an understanding of the user needs, without requiring that either be an expert in the other's domain. This is especially true for recognition-based technologies,

where the perplexity of the system (a term used in speech UI design to describe grammar size) has an impact on recognition rate.

Fourth, the heterogeneous ubicomp technologies make it challenging to limit the size of a toolkit's library components. With graphical user interfaces, there is a standard set of widgets, and these widgets span nearly all common applications. As research progresses, new interaction techniques are developed, but this rate of development is very small when compared to the size of applications. Papier-Mâché was able to limit the required library size by limiting applications to everyday objects: nearly all of our inspiring applications can be completely built with the three technologies in our library. It is an open research question how to limit the library size of TUI support beyond everyday objects.

Fifth, the heterogeneity of ubicomp technologies will benefit from continued research on model-based design techniques [241]. One example where model-based techniques could be applied is that Papier-Mâché's architecture for rapidly changing input technologies could be extended to mechatronic and haptic user interfaces. Continued work on model-based techniques could aid designers in exploring applications of different form factors with radically different input and output technologies. This would benefit both designer's abilities to explore alternatives and work iteratively and their ability to create interfaces that can be customized for individual situations and user needs.

Lastly, there is continued research to be conducted on user-centered methods for interface design tools, and toolkits more generally.

The work suggested here represents just a few of the many open research problems in creating design tools for tangible interaction.

10.4 Closing Remarks

Our direct interactions with computers largely consist of planar pointing devices and alphabetic and numerical keyboards. Our interaction in the physical world is much richer and more heterogeneous. In exchange for this constrained model of interaction, computers offer highly dynamic media that we can search, visualize, edit, and share. The disciplines of graphic and industrial design create documents and artifacts that honor our intuitions and capabilities, but the benefits of computation have largely been absent from these creations. This dissertation takes steps toward interaction techniques and design tools for creating artifacts that offer computation in concert with intuitions.

Bibliography

1 *Avalon*, 2004. Microsoft. http://msdn.microsoft.com/Longhorn/understanding/pillars/avalon

2 *The Bancroft Library Regional Oral History Office, UC Berkeley*, 2002. http://bancroft.berkeley.edu/ROHO

3 *Creating a GUI with JFC/Swing*, 2004. Sun Microsystems, Inc. http://java.sun.com/docs/books/tutorial/uiswing

4 *Cue Cat*. DigitalConvergence. http://www.digitalconvergence.com/

5 *Director*. Macromedia. http://www.macromedia.com/director

6 *Disability Rights and Independent Living Movement*, 2002. Regional Oral History Office. http://bancroft.berkeley.edu/collections/drilm.html

7 *Dreamweaver*. Macromedia. http://www.macromedia.com/dreamweaver

8 *Electronic Product Code*. EPCglobal, Inc. http://www.epcglobalinc.org

9 *Enterprise Java Beans*. Sun Microsystems, Inc. http://java.sun.com/products/ejb

10 *Flash*. Macromedia. http://www.macromedia.com/flash

11 *H-Oralhist*. http://www2.h-net.msu.edu/~oralhist

12 *Interface Tool for Java*. IBM. http://www.alphaworks.ibm.com/tech/bridge2java

13 *iPod*. Apple: Cupertino. http://www.apple.com/ipod

14 *ISO/IEC 15961:2004: Information technology -- Radio frequency identification (RFID) for item management -- Data protocol: application interface*. ISO. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=30528&ICS1=35&ICS2=40&ICS3=

15 *Java Advanced Imaging*, 2003. Sun Microsystems, Inc. http://java.sun.com/products/java-media/jai

16 *Java Media APIs*, 2003. Sun Microsystems, Inc. http://java.sun.com/products/java-media

17 *Java Media Framework*, 2003. Sun Microsystems, Inc. http://java.sun.com/jmf

18 *Java TWAIN*. Gnome Ltd.: Bratislava, Slovakia. http://www.gnome.sk/Twain/jtp_try&buy.html

19 *Max/MSP*. Cycling '74. http://www.cycling74.com/products/maxmsp.html

20 *Natural Interaction Systems*. Portland, OR. http://www.naturalinteraction.com

21 *Rear Projection Smart Board 1802*, 2000. Smart Technologies Inc. http://www.smarttech.com/rearprojection

22 *ShuttlePRO*. Contour Design. http://www.contourdesign.com/shuttlepro

23 *SpotCode*. High Energy Magic Ltd.: Cambridge, UK. http://www.highenergymagic.com/spotcode

24 *Start Stop Universal Digital Transcription*. HTH Engineering. http://www.startstop.com

25 *Whiteboard Photo*. PolyVision Corporation. http://www.pixid.com

26 Abowd, Gregory D. Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment. *IBM Systems Journal* **38**(4). pp. 508–30, 1999.

27 Arias, Ernesto, Hal Eden, Gerhard Fischer, Andrew Gorman, and Eric Scharff. Transcending the Individual Human Mind - Creating Shared Understanding through Collaborative Design. *ACM Transactions on Computer-Human Interaction* **7**(1). pp. 84–113, 2000.

28 Avrahami, Daniel and Scott Hudson. Forming Interactivity: A Tool for Rapid Prototyping of Physical Interactive Products. In Proceedings of *ACM Symposium on Designing Interactive Systems*. London, UK: ACM Press. pp. 141–46, June, 2002.

29 Back, Maribeth, Jonathan Cohen, Rich Gold, Steve Harrison, and Scott Minneman. Listen Reader: an electronically augmented paper-based book. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **3**(1). pp. 23–29, 2001.

30 Ballagas, Rafael, Meredith Ringel, Maureen Stone, and Jan Borchers. iStuff: a physical user interface toolkit for ubiquitous computing environments. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **5**(1). pp. 537–44, 2003.

31 Ballagas, Rafael, Andy Szybalski, and Armando Fox. The Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments. In Proceedings of *Percom 2004: IEEE International Conference on Pervasive Computing and Communications*. Orlando, FL, March, 2004.

32 Beck, Kent, *extreme programming eXplained: embrace change*. Reading, MA: Addison-Wesley. 190 pp. 2000.

33 Beckmann, Chris and Anind K. Dey. SiteView: Tangibly Programming Active Environments with Predictive Visualization. In Proceedings of *Fifth International Conference on Ubiquitous Computing*. Seattle, WA: Springer-Verlag. pp. 167–68, 2003.

34 Bellotti, Victoria and Yvonne Rogers. From Web Press to Web Pressure: Multimedia Representations and Multimedia Publishing. In Proceedings of *CHI: Human Factors in Computing Systems*. Atlanta, GA: ACM Press. pp. 279–86, 1997.

35 Belongie, Serge, Jitendra Malik, and Jan Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(4). pp. 509-22, 2002.

36 Berlage, Thomas and Andreas Genau. A framework for shared applications with a replicated architecture. In Proceedings of *UIST: User Interface Software and Technology*. Atlanta, GA: ACM Press. pp. 249-57, November 3-5, 1993.

37 Beyer, Hugh and Karen Holtzblatt, *Contextual Design: Defining Customer-Centered Systems*. 1st ed. San Francisco: Morgan Kaufmann. 496 pp. 1997.

38 Bishop, Christopher M., *Neural Networks for Pattern Recognition*. New York: Oxford University Press. 504 pp. 1995.

39 Blackwell, Alan F. and Thomas R. G. Green. A Cognitive Dimensions Questionnaire Optimised for Users. In Proceedings of *Workshop on the Psychology of Programming Interest Group*. Cozenza, Italy. pp. 137–54, April, 2000.

40 Bradski, Gary, *Open Source Computer Vision Library*, 2001. http://www.intel.com/research/mrl/research/opencv

41 Brave, Scott, Hiroshi Ishii, and Andrew Dahley. Tangible Interfaces for Remote Collaboration and Communication. In Proceedings of *CSCW: Computer Supported Cooperative Work*. Seattle, WA: ACM Press. pp. 169–78, 14–18 November, 1998.

42 Breit, Manfred, Daniel Kündig, and Fritz Häubi. Project oriented learning environment (POLE-Europe). In Proceedings of *International Conference on Computing in Civil and Building Engineering*. Weimar: ISCCBE, 2004.

43 Brewster, Karen, *Internet Access to Oral Recordings: Finding the Issues*, 2000. http://www.uaf.edu/library/libweb/collections/apr/internet.oralhist

44 Brooks, Frederick P. The Computer "Scientist" as Toolsmith: Studies in Interactive Computer Graphics. In Proceedings of *International Federation of Information Processing Congress '77*. Toronto, Canada: North-Holland. pp. 625-34, 1977.

45 Brooks, Frederick P. Essence and Accidents of Software Engineering. *IEEE Computer* **20**(4). pp. 10-19, 1987.

46 Brotherton, Jason A., *Enriching Everyday Experiences through the Automated Capture and Access of Live Experiences: eClass: Building, Observing and Understanding the Impact of Capture and Access in an Educational Domain*, Unpublished PhD, Georgia Institute of Technology, Computer Science, Atlanta, 2001. http://www.cc.gatech.edu/fce/eclass/pubs/brotherton-thesis.pdf

47 Burnett, Margaret, Andrei Sheretov, Bing Ren, and Gregg Rothermel. Testing Homogeneous Spreadsheet Grids with the "What You See Is What You Test" Methodology. *IEEE Transactions on Software Engineering* **28**(6): IEEE Press. pp. 576 - 94, 2002.

48 Canny, John F. A computational approach to edge detection. *Transactions on Pattern Analysis and Machine Intelligence* **8**(6): IEEE. pp. 679-98, 1986.

49 Canny, John F., *Finding edges and lines in images.*, Unpublished Masters thesis, MIT, AI Lab, Cambridge, MA, 1983.

50 Card, Stuart K., Thomas P. Moran, and Allen Newell, Chapter 2: The Human Information Processor, in *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum: Hillsdale. pp. 23-97, 1983.

51 Carson, Chad, Serge Belongie, Hayit Greenspan, and Jitendra Malik. Blobworld: Image Segmentation Using Expectation-Maximization and Its Application to Image Querying. *Transactions on Pattern Analysis and Machine Intelligence* **24**(8): IEEE. pp. 1026-38, 2002.

52 Chang, Shi-Kuo, ed. *Principles of visual programming systems*. ed. Prentice Hall: Englewood Cliffs, NJ. 372 pp., 1990.

53 Churchill, Elizabeth F. and Les Nelson. Tangibly Simple, Architecturally Complex: Evaluating a Tangible Presentation Aid. In Proceedings of *Extended Abstracts of CHI: Conference on Human Factors in Computing Systems*: ACM Press. pp. 750-51, 2002.

54 Clarke, Steven. Evaluating a New Programming Language. In Proceedings of *Workshop of the Psychology of Programming Interest Group*. Bournemouth, UK. pp. 275–89, 2001.

55 Clarke, Steven. Measuring API Usability, *Dr. Dobbs Journal*, vol. 29(5): pp. S6-S9, May, 2004.

56 Clements, Paul, Rick Kazman, and Mark Klein, *Evaluating Software Architectures: Methods and Case Studies*. SEI series in software engineering. Boston: Addison-Wesley. 323 pp. 2002.

57 Cohen, Philip R. and David R. McGee. Tangible Multimodal Interfaces for Safety Critical Applications, *Communications of the ACM*, vol. 47(1): pp. 41-46, January, 2004.

58 Covi, Lisa M., Judith S. Olsen, Elena Rocco, William J. Miller, and Paul Allie. A Room of Your Own: What do we learn about support of teamwork from assessing teams in dedicated project rooms. In Proceedings of *Cooperative Buildings: Integrating Information, Organization and Architecture: First International Workshop, Co'Build '98*. Darmstadt, Germany, 1998.

59 Culler, David E. and Hans Mulder. Smart Sensors to Network the World. *Scientific American* **290**(6). pp. 84–91, 2004.

60 De Guzman, Edward and Gary Hsieh, *Function Composition in Physical Chaining Applications*, UC Berkeley, Berkeley, CA, April 14 2003. http://guir.berkeley.edu/projects/papier-mache/pubs/cs260paper.pdf

61 De Guzman, Edward S., Ana Ramírez, and Scott R. Klemmer, *ObjectClassifierViews: Support for Visual Programming of Image Classifiers*, UC Berkeley, Berkeley, CA, December 2003.
http://www.cs.berkeley.edu/~anar/publications/esdfinalreportubicomp.pdf

62 Détienne, Françoise, *Software Design—Cognitive Aspects*. Practitioner series. London: Springer Verlag. 200 pp. 2001.

63 Dey, Anind K., *CS294-2: Ubiquitous Computing*, 2003. Berkeley, CA.
http://www.cs.berkeley.edu/~dey/cs294-fall2003

64 Dey, Anind K., *Providing Architectural Support for Building Context-Aware Applications*, Georgia Institute of Technology, College of Computing, Atlanta, 2000.
http://www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf

65 Dey, Anind K., Daniel Salber, and Gregory D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction* **16**(2–4). pp. 97–166, 2001.

66 Dill, David, *Verified Voting*, 2004. http://www.verifiedvoting.org

67 Dourish, Paul, *Where the action is: the foundations of embodied interaction*. Cambridge, Mass.: MIT Press. 233 pp. 2001.

68 Dunaway, David K. and Willa K. Baum, ed. *Oral History: An Interdisciplinary Anthology*. 2nd ed. Altamira Press. 440 pp., 1996.

69 Edwards, W. Keith, Victoria Bellotti, Anind K. Dey, and Mark W. Newman. Stuck in the Middle: The Challenges of User-Centered Design and Evaluation for Infrastructure. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **5**(1). pp. 297–304, 2003.

70 Edwards, W. Keith and Elizabeth D. Mynatt. Timewarp: techniques for autonomous collaboration. In Proceedings of *CHI: Human Factors in Computing Systems*. Atlanta, GA: ACM Press. pp. 218–25, March 22–27, 1997.

71 Ellis, Jason B. and Amy S. Bruckman. Designing Palaver Tree Online: Supporting social roles in a community of oral history. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **3**(1). pp. 474-81, 2001.

72 Elrod, Scott, Richard Bruce, Rich Gold, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin Pedersen, Ken Pier, John Tang, and Brent Welch. Liveboard: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration. In Proceedings of *CHI: Human Factors in Computing Systems*: ACM Press. pp. 599–607, 1992.

73 Everitt, Katherine M., Scott R. Klemmer, Robert Lee, and James A. Landay. Two Worlds Apart: Bridging the Gap Between Physical and Virtual Media for Distributed Design Collaboration. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **5**(1). pp. 553–60, 2003.

74 Everitt, Katherine Mary, *Two Worlds Apart: Bridging the Gap Between Physical and Virtual Media for Distributed Design Collaboration*. Unpublished MSc Report, University of California, Berkeley 2003.

75 Fails, Jerry Alan and Dan R. Olsen. A Design Tool for Camera-based Interaction. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **5**(1). pp. 449–56, 2003.

76 Feiner, Steven, Blair Macintyre, and Doree Seligman. Knowledge-Based Augmented Reality. *Communications of the ACM* **36**(7). pp. 53–62, 1993.

77 Fishkin, Kenneth P. A Taxonomy for and Analysis of Tangible Interfaces. *Personal and Ubiquitous Computing* **8**(5). pp. 347–58, 2004.

78 Fishkin, Kenneth P., Thomas P. Moran, and Beverly L. Harrison. Embodied User Interfaces: Towards Invisible User Interfaces. In Proceedings of *Conference on Engineering for Human-Computer Interaction*. Heraklion, Crete. pp. 1–18, September 14-18, 1998.

79 Fitzmaurice, George W., *Graspable User Interfaces*, Unpublished PhD, University of Toronto, Computer Science, Toronto, 1996. http://www.dgp.toronto.edu/~gf/papers/PhD%20-%20Graspable%20UIs/Thesis.gf.html

80 Fitzmaurice, George W. and William Buxton. An empirical evaluation of Graspable User Interfaces: towards specialized, space-multiplexed input. In Proceedings of *CHI: Human Factors in Computing Systems*: ACM Press. pp. 43–50, 1997.

81 Fitzmaurice, George W., Hiroshi Ishii, and William Buxton. Bricks: laying the foundations for graspable user interfaces. In Proceedings of *CHI: Human Factors in Computing Systems*. pp. 442–49, 1995.

82 Foley, James D. and Victor L. Wallace. The art of natural graphic man-machine conversation. *IEEE Computer* **62**(4). pp. 462–71, 1974.

83 Forsyth, David A. and Jean Ponce, *Computer Vision: A Modern Approach*. Upper Saddle River: Prentice Hall. 693 pp. 2003.

84 Freeman, William T., Yasunari Miyake, Ken-ichi Tanaka, David B. Anderson, Paul A. Beardsley, Chris N. Dodge, Michal Roth, Craig D. Weissman, William S. Yerazunis, Hiroshi Kage, and Kazuo Kyuma. Computer vision for interactive computer graphics. *IEEE Computer Graphics and Applications* **18**(3). pp. 42–53, 1998.

85 Gajos, Krzysztof and Daniel S. Weld. SUPPLE: automatically generating user interfaces. In Proceedings of *IUI: International Conference on Intelligent user interfaces*. Funchal, Madeira, Portugal: ACM Press. pp. 93-100, 2004.

86 Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed: Addison-Wesley Pub Co. 395 pp. 1995.

87 Garcia-Molina, Hector, Jeffrey D. Ullman, and Jennifer D. Widom, *Database Systems: The Complete Book*. 1st ed. Upper Saddle River, NJ: Prentice Hall. 1100 pp. 2001.

88 Gershenfeld, Neil, Bits and Books, in *When Things Start to Think*. Henry Holt & Co: New York. pp. 13-25, 1999.

89 Gibbs, Graham R., Susanne Friese, and Wilma C. Mangabeira, *Using Technology in the Qualitative Research Process*, 2002. Forum: Qualitative Social Research. http://www.qualitative-research.net/fqs/fqs-eng.htm

90 Gleicher, Michael. A Graphics Toolkit Based on Differential Constraints. In Proceedings of *UIST: User Interface Software and Technology*: ACM Press. pp. 109-20, 1993.

91 Goldberg, Adele, *Smalltalk in the Classroom*. Technical Report SSL 77-2, Xerox Palo Alto Research Center, Palo Alto, CA, June 1977.

92 Goldberg, Adele and Alan Kay, *Methods for Teaching the Programming Language Smalltalk*. Technical Report SSL 77-2, Xerox Palo Alto Research Center, Palo Alto, CA, June 1977.

93 Gorbet, Matthew, Maggie Orth, and Hiroshi Ishii. Triangles: Tangible Interface for Manipulation and Exploration of Digital Information Topography. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. Los Angeles, CA: ACM Press. pp. 49–56, 1998.

94 Grady, Robert B., Practical Software Metrics for Project Management and Process Improvement. Prentice Hall: Englewood Cliffs, NJ. p. 17, 1992.

95 Graham, Jamey and Jonathan J. Hull. Video Paper: A Paper-Based Interface for Skimming and Watching Video. In Proceedings of *International Conference on Consumer Electronics*. Los Angeles, CA. pp. 214-15, June 16-18, 2002.

96 Grasso, Antonietta, Alain Karsenty, and Marco Susani. Augmenting paper for community information sharing. In Proceedings of *DARE 2000: Designing augmented reality environments*. Elsinore, Denmark: ACM Press. pp. 51–62, 2000.

97 Green, Thomas R. G. and Marian Petre. Usability Analysis of Visual Programming Environments. *Journal of Visual Languages and Computing* **7**(2). pp. 131–74, 1996.

98 Greenberg, S. and H. Kuzuoka. Using digital but physical surrogates to mediate awareness, communication and privacy in media spaces. *Personal Technologies* **3**(4). pp. 182–98, 1999.

99 Greenberg, Saul and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **3**(2). pp. 209–18, 2001.

100 Grimm, Robert, Janet Davis, Eric Lemar, Adam MacBeth, Steven Swanson, Thomas Anderson, Brian Bershad, and Gaetano Borriello. System Support for Pervasive Applications. *Trasactions on Computer Systems* **22**(4): ACM Press. pp. 421-86, 2004.

101 Guimbretière, François, Maureen Stone, and Terry Winograd. Fluid Interaction with High-resolution Wall-size Displays. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters*: ACM Press. pp. 21–30, 2001.

102 Gustman, Samuel, Dagobert Soergel, Douglas Oard, William Byrne, Michael Picheny, Bhuvana Ramabhadran, and Douglas Greenberg. Building and using cultural digital libraries: Supporting access to large digital oral history archives. In Proceedings of *ACM Joint Conference on Digital Libraries*. Houston, TX: ACM Press. pp. 18-27, 2002.

103 Hecht, David L. Embedded data glyph technology for hardcopy digital documents. In Proceedings of *Color Hard Copy and Graphic Arts III*. San Jose, CA: SPIE. pp. 341–52, 1994.

104 Hecht, David L. Printed Embedded Data Graphical User Interfaces. *IEEE Computer* **34**(3). pp. 47–55, 2001.

105 Heidegger, Martin, The Origin of the Work of Art, in *Poetry, Language, Thought*. Harper & Row: New York. p. 55, 1971.

106 Heidegger, Martin, The Worldhood of the World, in *Being and Time*. HarperSanFrancisco: New York. pp. 63-112, 1962.

107 Heiner, Jeremy M., Scott E. Hudson, and Kenichiro Tanaka. Linking and Messaging from Real Paper in the Paper PDA. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **1**(1). pp. 179–86, 1999.

108 Hinckley, Ken, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive Real-World Interface Props for Neurosurgical Visualization. In Proceedings of *CHI: Human Factors in Computing Systems*. Boston, MA: ACM Press. pp. 452–58, April, 1994.

109 Holmquist, Lars E., Johan Redström, and Peter Ljungstrand. Token-based access to digital information. In Proceedings of *Handheld and Ubiquitous Computing. First International Symposium, HUC'99*. Karlsruhe, Germany: Springer-Verlag. pp. 234–45, September 27–29, 1999.

110 Hong, Jason I. and James A. Landay. SATIN: A Toolkit for Informal Ink-based Applications. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **2**(2). pp. 63–72, 2000.

111 Horn, Berthold Klaus Paul, Binary Images: Topological Properties, in *Robot Vision*. MIT Press: Cambridge. pp. 65-89, 1986.

112 Hudson, Scott E. and Ian Smith, *SubArctic UI Toolkit User's Manual*, 1996. Georgia Institute of Technology: Atlanta, GA.
http://www.cc.gatech.edu/gvu/ui/sub_arctic/sub_arctic/doc/users_manual.html

113 Hudson, Scott E. and Ian Smith. Ultra-Lightweight Constraints. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*. Seattle, WA: ACM Press. pp. 147–55, 1996.

114 Hymes, Charles M. and Gary M. Olson. Quick but not so dirty web design: Applying empirical conceptual clustering techniques to organise hypertext content. In Proceedings of *DIS: Symposium on Designing Interactive Systems*: ACM Press. pp. 159–62, 1997.

115 Ishii, Hiroshi, *Tangible Media Group*. MIT: Cambridge, MA. http://tangible.media.mit.edu/

116 Ishii, Hiroshi, Minoru Kobayashi, and Kazuho Arita. Iterative Design of Seamless Collaboration Media, *Communications of the ACM*, vol. 37(8): pp. 83–97, August, 1994.

117 Ishii, Hiroshi and Brygg Ullmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In Proceedings of *CHI: Human Factors in Computing Systems*. pp. 234–41, 1997.

118 Jacob, Robert, Hiroshi Ishii, Gian Pangaro, and James Patten. A Tangible Interface for Organizing Information Using a Grid. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **4**(1). pp. 339–46, 2002.

119 Jacob, Robert J. K., Leonidas Deligiannidis, and Stephen Morrison. A Software Model and Specification Language for Non-WIMP User Interfaces. *Transactions on Computer-Human Interaction* **6**(1). pp. 1–46, 1999.

120 Johanson, Brad, Armando Fox, and Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms, *IEEE Pervasive Computing Magazine*, vol. 1(2): pp. 67–74, April, 2002.

121 Johnson, Jeff, Teresa L. Roberts, William Verplank, David Canfield Smith, Charles Irby, Marian Beard, and Kevin Mackey. The Xerox Star: A Retrospective. *IEEE Computer* **22**(9). pp. 11–28, 1989.

122 Johnson, Walter, Herbert Jellinek, Leigh Klotz Jr., Ramana Rao, and Stuart Card. Bridging the Paper and Electronic Worlds: The Paper User Interface. In Proceedings of *INTERCHI: Human Factors in Computing Systems*. pp. 507–12, 1993.

123 Jones, Simon Peyton, Alan Blackwell, and Margaret Burnett. A User-Centred Approach to Functions in Excel. In Proceedings of *SIGPLAN International Conference on Functional programming*. Uppsala, Sweden: ACM Press. pp. 165-76, July, 2003.

124 Kaasten, Shaun and Saul Greenberg. Integrating Back, History and Bookmarks in Web Browsers. In Proceedings of *Extended Abstracts of CHI: Conference on Human Factors in Computing Systems*. Seattle, WA: ACM Press. pp. 379–80, 2001.

125 Kantarjiev, Christopher Kent, A. Demers, R.T. Krivacic, R. Frederick, and Mark Weiser. Experiences with X in a wireless environment. In Proceedings of *USENIX Symposium on Mobile & Location-independent Computing*. pp. 117–28, August, 1993.

126 Kato, Hirokazu, Mark Billinghurst, and Ivan Poupyrev, *ARToolKit*, 2000. University of Washington HIT Lab. http://www.hitl.washington.edu/artoolkit/

127 Kay, Alan. The early history of Smalltalk. In Proceedings of *ACM SIGPLAN conference on History of programming languages*. Cambridge, MA: ACM. pp. 69–95, 1993.

128 Kelley, John F. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Office Information Systems* **2**(1). pp. 26–41, 1984.

129 Kim, Jiwon, Steve Seitz, and Maneesh Agrawala. Video-Based Document Tracking: Unifying Your Physical and Electronic Desktops. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **6**(2). pp. 99-108, 2004.

130 Kim, Scott, *From Physical Game to Computer Game*. http://www.scottkim.com/thinkinggames/fromphysicaltocomputer

131 Kindberg, Tim. Implementing physical hyperlinks using ubiquitous identifier resolution. In Proceedings of *11th International World Wide Web Conference*. Honolulu. pp. 191–99, 2002.

132 Klemmer, Scott R., Jamey Graham, Gregory J. Wolff, and James A. Landay. Books with Voices: Paper Transcripts as a Tangible Interface to Oral Histories. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **5**(1): ACM Press. pp. 89–96, 2003.

133 Klemmer, Scott R., Jack Li, James Lin, and James A. Landay. Papier-Mâché: Toolkit Support for Tangible Input. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **6**(1). pp. 399–406, 2004.

134 Klemmer, Scott R., Mark W. Newman, Ryan Farrell, Mark Bilezikjian, and James A. Landay. The Designers' Outpost: A Tangible Interface for Collaborative Web Site Design. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **3**(2). pp. 1–10, 2001.

135 Klemmer, Scott R., Anoop K. Sinha, Jack Chen, James A. Landay, Nadeem Aboobaker, and Annie Wang. SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **2**(2). pp. 1–10, 2000.

136 Klemmer, Scott R., Michael Thomsen, Ethan Phelps-Goodman, Robert Lee, and James A. Landay. Where Do Web Sites Come From? Capturing and Interacting with Design History. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **4**(1). pp. 1–8, 2002.

137 Krasner, Glenn E. and Stephen T. Pope. A description of the model-view-controller user interface paradigm in the Smalltalk-80 system. *Object Oriented Programming* **1**(3). pp. 26–49, 1988.

138 Krueger, Myron W., *Artificial Reality*. 1st ed. Reading, MA: Addison-Wesley Publishing Co. 312 pp. 1983.

139 Krueger, Myron W. Responsive Environments. In Proceedings of *AFIPS National Computer Conference*. Montvale, NJ. pp. 423-33, 1977.

140 Kurlander, David, *ASP.NET Mobile Controls*, 2003. Microsoft: Redmond, WA. http://msdn.microsoft.com/mobility/othertech/asp.netmc

141 Kurlander, David and Steven Feiner. A history-based macro by example system. In Proceedings of *UIST: User Interface Software and Technology*. Monterey, CA, USA: ACM Press. pp. 99–106, November 15–18, 1992.

142 Landay, James A., *Interactive Sketching for the Early Stages of User Interface Design*, Unpublished PhD, Carnegie Mellon University, Computer Science, Pittsburgh, PA, 1996. http://www.cs.berkeley.edu/~landay/research/publications/Thesis.pdf

143 Landay, James A. and Brad A. Myers. Extending an Existing User Interface Toolkit to Support Gesture Recognition. In Proceedings of *Adjunct Proceedings of INTERCHI '93: Human Factors in Computing Systems*. pp. 91–92, April 24–29, 1993.

144 Landay, James A. and Brad A. Myers. Interactive Sketching for the Early Stages of User Interface Design. In Proceedings of *CHI: Human Factors in Computing Systems*. Denver, CO: ACM Press. pp. 43–50, May 7–11, 1995.

145 Landay, James A. and Brad A. Myers. Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer* **34**(3). pp. 56–64, 2001.

146 Lange, Beth M., Mark A. Jones, and James L. Meyers. Insight Lab: An lmmersive Team Enviroriment Linking Paper, Displays, and Data. In Proceedings of *CHI: Human Factors in Computing Systems*. Los Angeles, CA: ACM Press. pp. 550-57, 1998.

147 Lederer, Scott and Jeff Heer. All together now: visualizing local and remote actors of localized activity. In Proceedings of *Extended Abstracts of CHI: Conference on Human Factors in Computing Systems*, 2004.

148 Lee, Johnny, Daniel Avrahami, Scott Hudson, Jodi Forlizzi, Paul Dietz, and Darren Leigh. The Calder Toolkit: Wired and Wireless Components for Rapidly Prototyping Interactive Devices. In Proceedings of *ACM Symposium on Designing Interactive Systems*. Cambridge, MA: ACM Press. pp. 167–75, August, 2004.

149 Li, Francis C., Anoop Gupta, Elizabeth Sanocki, He Li-Wei, and Rui Yong. Browsing digital video. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **2**(1). pp. 169-76, 2000.

150 Liblit, Ben, Alex Aiken, Alice X. Zheng, and Michael I. Jordan. Bug Isolation via Remote Program Sampling. In Proceedings of *PLDI: SIGPLAN 2003 Conference on Programming Language Design and Implementation*. San Diego: ACM Press. pp. 141-54, 2003.

151 Lin, James. Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. In Proceedings of *Conference Supplement to UIST 2003: ACM Symposium on User Interface Software and Technology: Doctoral Consortium*. Vancouver, BC: ACM Press. pp. 13–16, 2003.

152 Lin, James and James A. Landay. Damask: A Tool for Early-Stage Design and Prototyping of Multi-Device User Interfaces. In Proceedings of *8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing)*. San Francisco, CA. pp. 573–80, 2002.

153 Lin, James, Mark W. Newman, Jason I. Hong, and James A. Landay. DENIM: Finding a tighter fit between tools and practice for web site design. CHI: Human Factors in Computing Systems, *CHI Letters* **2**(1): ACM. pp. 510–17, 2000.

154 Ljungstrand, Peter, Johan Redström, and Lars Erik Holmquist. WebStickers: using physical tokens to access, manage and share bookmarks to the Web. In Proceedings of *DARE: Designing Augmented Reality Environments*. Elsinore, Denmark: ACM Press. pp. 23-31, 2000.

155 MacIntyre, Blair, Maribeth Gandy, Steven Dow, and Jay David Bolter. DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **6**(2). pp. 197-206, 2004.

156 Mackay, Wendy E. Is Paper Safer? The Role of Paper Flight Strips in Air Traffic Control. *ACM Transactions on Computer-Human Interaction* **6**(4). pp. 311–40, 1999.

157 Mackay, Wendy E. and Anne-Laure Fayard. Designing interactive paper: lessons from three augmented reality projects. In Proceedings of *IWAR'98, International Workshop on Augmented Reality*. Natick, MA: A K Peters, Ltd. pp. 81–90, November, 1998.

158 Mackay, Wendy E., Anne-Laure Fayard, Laurent Frobert, and Lionel Médini. Reinventing the Familiar: Exploring an Augmented Reality Design Space for Air Traffic Control. In Proceedings of *CHI: Human Factors in Computing Systems*. Los Angeles, California: ACM Press. pp. 558–65, 1998.

159 Mackay, Wendy E. and Daniele Pagani. Video Mosaic: Laying out time in a physical space. In Proceedings of *Multimedia*. San Francisco, CA: ACM Press. pp. 165–72, October, 1994.

160 Mackay, Wendy E., Guillaume Pothier, Catherine Letondal, Kaare Bøegh, and Hans Erik Sørensen. The Missing Link: Augmenting Biology Laboratory Notebooks. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **4**(2). pp. 41-50, 2002.

161 MacKenzie, Christine L. and Thea Iberall, *The Grasping Hand*. Advances in Psychology, ed. G.E. Stelmach and P.A. Vroon. Vol. 104. Amsterdam: Elsevier Science B.V. 482 pp. 1994.

162 MacLean, Allan, Richard M. Young, Victoria Bellotti, and Thomas P. Moran. Questions, options, and criteria: elements of design space analysis. *Human-Computer Interaction* **6**(3–4). pp. 201–50, 1991.

163 MacLean, Allan, Richard M. Young, and Thomas P. Moran. Design rationale: the argument behind the artifact. In Proceedings of *CHI: Human Factors in Computing Systems*. Pittsburgh, PA: ACM Press. pp. 247–52, 1989.

164 Mankoff, Jennifer, *An architecture and interaction techniques for handling ambiguity in recognition-based input*, Unpublished PhD, Georgia Institute of Technology, Computer Science, Atlanta, GA, 2001. http://www.cc.gatech.edu/fce/people/jmankoff/thesis.pdf

165 Mankoff, Jennifer, Scott E. Hudson, and Gregory D. Abowd. Interaction techniques for ambiguity resolution in recognition-based interfaces. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **2**(2). pp. 11-20, 2000.

166 Mankoff, Jennifer, Scott E. Hudson, and Gregory D. Abowd. Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **2**(1). pp. 368–75, 2000.

167 Marr, David, *Vision: A computational investigation into the human representation and processing of visual information*. San Francisco: W.H. Freeman. 397 pp. 1982.

168 Martin, David, *SMART Technologies*. Calgary, CA. http://www.smarttech.com

169 Martin, David L., Adam J. Cheyer, and Douglas B. Moran. The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence* **13**(1-2): Taylor & Francis, Inc. pp. 91-128, 1999.

170 Martin, Paul, Frederick Crabbe, Stuart Adams, Eric Baatz, and Nicole Yankelovich. SpeechActs: A Spoken-Language Framework. *IEEE Computer* **29**(7). pp. 33-40, 1996.

171 Matthews, Tara, Anind K. Dey, Jennifer Mankoff, Scott Carter, and Tye Rattenbury. A Toolkit for Managing User Attention in Peripheral Displays. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **6**(2): ACM Press. pp. 247-56, 2004.

172 Maulsby, David, Saul Greenberg, and Richard Mander. Prototyping an intelligent agent through Wizard of Oz. In Proceedings of *INTERCHI: ACM Conference on Human Factors in Computing Systems*. Amsterdam, The Netherlands: ACM. pp. 277–84, 1993.

173 McGee, David R., Philip R. Cohen, R. Matthews Wesson, and Sheilah Horman. Comparing paper and tangible, multimodal tools. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **4**(1). pp. 407–14, 2002.

174 McGee, David R., Philip R. Cohen, and Lizhong Wu. Something from nothing: Augmenting a paper-based work practice via multimodal interaction. In Proceedings of *Designing Augmented Reality Environments*. Helsinor, Denmark: ACM Press. pp. 71–80, April 12–14, 2000.

175 McGrath, Joseph E., Methodology Matters: Doing Research in the Behavioral and Social Sciences, in *Readings in Human-Computer Interaction: Toward the Year 2000*, R.M. Baecker, *et al.*, Editors. Morgan Kaufmann: San Francisco. p. 152–69, 1994.

176 McPhail, Susannah. Buddy Bugs: A Physical User Interface for Windows® Instant Messenger. In Proceedings of *Western Computer Graphics Symposium (Skigraph'02)*, March, 2002.

177 Moran, Thomas P. and John M. Carroll, ed. *Design Rationale: Concepts, Techniques, and Use*. ed. Lawrence Erlbaum Associates.: Mahwah, N.J., 1996.

178 Moran, Thomas P., Patrick Chiu, William van Melle, and Gordon Kurtenbach, Implicit Structures for Pen-Based Systems within a Freeform Interaction Paradigm, in *CHI: Human Factors in Computing Systems*. ACM Press. p. 487–94, 1995.

179 Moran, Thomas P., Laysia Palen, Steve Harrison, Patrick Chiu, Don Kimber, Scott Minneman, William van Melle, and Polle Zellweger. "I'll get that off the audio": a case study of salvaging multimedia records. In Proceedings of *CHI: Human Factors in Computing Systems*. Atlanta, GA: ACM Press. pp. 202–09, March 22–27, 1997.

180 Moran, Thomas P., Eric Saund, William van Melle, Anuj Gujar, Kenneth P. Fishkin, and Beverly L. Harrison. Design and Technology for Collaborage: Collaborative Collages of Information on Physical Walls. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **1**(1): ACM Press. pp. 197–206, 1999.

181 Murphy, Kate, Bigger Bar Code Inches Up on Retailers, *The New York Times*, August 12, 2002. http://www.nytimes.com/2002/08/12/technology/12CODE.html

182 Myers, Brad A. A new model for handling input. *ACM Transactions on Information Systems* **8**(3). pp. 289–320, 1990.

183 Myers, Brad A., Ellen Borison, Alan Ferrency, Rich McDaniel, Robert C. Miller, Andrew Faulring, Bruce D. Kyle, Patrick Doane, Andy Mickish, and Alex Klimovitski, *The Amulet V3.0 Reference Manual*, 1997. Carnegie Mellon University: Pittsburgh, PA. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/amulet/amulet3/manual/ammanual.pdf

184 Myers, Brad A., Dario Giuse, Roger B. Dannenberg, Brad Vander Zanden, David Kosbie, Ed Pervin, Andrew Mickish, and Philippe Marchal. Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces, *IEEE Computer*, vol. 23(11): pp. 71–85, November, 1990.

185 Myers, Brad A. and David S. Kosbie. Reusable hierarchical command objects. In Proceedings of *CHI: Human Factors in Computing Systems*. Vancouver, Canada: ACM Press. pp. 260–67, April 13–18, 1996.

186 Myers, Brad A., Richard G. McDaniel, Robert C. Miller, Alan S. Ferrency, Andrew Faulring, Bruce D. Kyle, Andrew Mickish, Alex Klimovitski, and Patrick Doane. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering* **23**(6). pp. 347–65, 1997.

187 Myers, Brad A. and Mary Beth Rosson. Survey on User Interface Programming. In Proceedings of *CHI: Human Factors in Computing Systems*. Monterey, CA: ACM Press. pp. 195-202, May 3-7, 1992.

188 Myers, Brad, Scott E. Hudson, and Randy Pausch. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction* **7**(1). pp. 3–28, 2000.

189 Nelson, Les, Satoshi Ichimura, Elin Rønby Pedersen, and Lia Adams. Palette: a paper interface for giving presentations. In Proceedings of *CHI: Human Factors in Computing Systems*. Pittsburgh, PA: ACM Press. pp. 354–61, 1999.

190 Newell, Alan and Stuart K. Card. The Prospects for Psychological Science in Human-Computer Interaction. *Human-Computer Interaction* **1**(3). pp. 209–42, 1985.

191 Newman, Mark W. and James A. Landay. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In Proceedings of *DIS: Symposium on Designing Interactive Systems*. New York, NY: ACM Press. pp. 263–74, August 17–19, 2000.

192 Newman, Mark W., James Lin, Jason I. Hong, and James A. Landay. DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. *Human-Computer Interaction* **18**(3). pp. 259–324, 2003.

193 Nielsen, Jakob. Finding Usability Problems Through Heuristic Evaluation. In Proceedings of *CHI: Human Factors in Computing Systems*. Monterey, CA: ACM Press. pp. 373–80, 1992.

194 Nielsen, Jakob, *Heuristic Evaluation*. http://www.useit.com/papers/heuristic

195 Nielsen, Jakob, Heuristic Evaluation, in *Usability Inspection Methods*, J. Nielsen and R.L. Mack, Editors. John Wiley & Sons, Inc.: New York. p. 25–62, 1994.

196 Nielsen, Jakob, *Test it! Jakob Nielsen's UI Tips*, 1998. http://www.builder.com/Graphics/UserInterface/ss01a.html

197 Norman, Donald A., *The Design of Everyday Things*. New York: Doubleday. 257 pp. 1990.

198 Norman, Donald A., *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution*. Cambridge, MA: MIT Press. 302 pp. 1998.

199 Olsen, Dan R., Chapter 5: Basic Interaction, in *Developing User Interfaces*. Morgan Kaufmann. p. 132–62, 1998.

200 Pane, John, *A Programming System for Children that is Designed for Usability*, Unpublished PhD, Carnegie Mellon University, Computer Science, Pittsburgh, 2002. http://www.cs.cmu.edu/~pane/thesis

201 Pauline, Mark, *Survival Research Laboratories*. San Francisco, CA. http://www.srl.org

202 Pedersen, Elin Rønby. People presence or room activity supporting peripheral awareness over distance. In Proceedings of *Extended Abstracts of CHI: Conference on Human Factors in Computing Systems*. Los Angeles: ACM Press. pp. 283–84, April 18–23, 1998.

203 Pedersen, Elin Rønby and Tomas Sokoler. AROMA: abstract representation of presence supporting mutual awareness. In Proceedings of *CHI: Conference on Human Factors in Computing Systems*. Atlanta, GA: ACM Press. pp. 51-58, 1997.

204 Polynor, Rick. The Hand That Rocks the Cradle, *I.D.* pp. 60–65, June, 1995.

205 Rekimoto, Jun. Time-Machine Computing: A Time-centric Approach for the Information Environment. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **1**(1). pp. 45–54, 1999.

206 Rekimoto, Jun and Yuji Ayatsuka. CyberCode: Designing Augmented Reality Environments with Visual Tags. In Proceedings of *Designing Augmented Reality Environments (DARE 2000)*. Elsinore, Denmark: ACM Press. pp. 1–10, 2000.

207 Rekimoto, Jun and Masanori Saitoh. Augmented surfaces: a spatially continuous work space for hybrid computing environments. In Proceedings of *CHI: Human Factors in Computing Systems*: ACM Press. pp. 378–85, 1999.

208 Rekimoto, Jun, Brygg Ullmer, and Haruo Oba. DataTiles: a modular platform for mixed physical and graphical interactions. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **3**(1). pp. 269–76, 2001.

209 Rettig, Mark, *Walls beyond whiteboards*, 2000. AIGA Advance for Design Summit: Telluride, CO. http://www.marcrettig.com/writings/rettig.walls.72dpi.pdf

210 Reznik, Dan S., *The Universal Planar Manipulator*, Unpublished PhD, UC Berkeley, EECS Department, Berkeley, 2000.

211 Rhyne, James R. and Catherine G. Wolf. Tools for supporting the collaborative process. In Proceedings of *UIST: User Interface Software and Technology*. Monterey, CA: ACM Press. pp. 161–70, November 15–18, 1992.

212 Richter, Heather, Gregory D. Abowd, Werner Geyer, Ludwin Fuchs, Shahrokh Daijavad, and Steven Poltrock. Integrating Meeting Capture within a Collaborative Team Environment. In Proceedings of *Ubicomp 2001*. pp. 123–38, 2001.

213 Riley, Jocelyn, *human voice*, 2002. H-OralHist Mailing List. http://h-net.msu.edu/cgi-bin/logbrowse.pl?trx=vx&list=h-oralhist&month=0209&week=c&msg=DKYtxLCtdA8Uwy3IP52fLA&user=&pw=

214 Rittel, Horst W. J. and Melvin M. Webber. Dilemmas in a general theory of planning. *Policy Sciences* **4**. pp. 155–69, 1973.

215 Rodden, Kerry and Alan Blackwell. Class Libraries: A Challenge for Programming Usability Research. In Proceedings of *Workshop of the Psychology of Programming Interest Group*. London, UK. pp. 186–95, 2002.

216 Salber, Daniel, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In Proceedings of *CHI: Human Factors in Computing Systems*. Pittsburgh, PA: ACM Press. pp. 21–28, 1999.

217 Saund, Eric. Bringing the Marks on a Whiteboard to Electronic Life. In Proceedings of *CoBuild'99: Second International Workshop on Cooperative Buildings*. Pittsburgh: Springer, October 1-2, 1999.

218 Saund, Eric, *Image Mosaicing and a Diagrammatic User Interface for an Office Whiteboard Scanner*. Technical Report, Xerox Palo Alto Research Center, Palo Alto 1998. http://www2.parc.com/spl/members/saund/papers/zombieboard98.pdf

219 Schilit, Bill N., Gene Golovchinsky, and Morgan N. Price. Beyond paper: Supporting active reading with free form digital ink annotations. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 249–56, 1998.

220 Schilit, Bill N., Morgan N. Price, Gene Golovchinsky, Kei Tanaka, and Catherine C. Marshall. The reading appliance revolution. *Computer* **32**(1). pp. 65–73, 1999.

221 Schmucker, Kurt J. MacApp: An Application Framework, *Byte*, vol. 11(8): pp. 189–93, August, 1986.

222 Sellen, Abigail J. and Richard Harper, *The Myth of the Paperless Office*. 1st ed. Cambridge, Mass.: MIT Press. 245 pp. 2001.

223 Shaer, Orit, Nancy Leland, Eduardo H. Calvillo-Gamez, and Robert J. K. Jacob. The TAC Paradigm: Specifying Tangible User Interfaces. *Personal and Ubiquitous Computing* **8**(5). pp. 359–69, 2004.

224 Sharp, Richard. New Ways to Maximize Camera Phone Technology, *Technology@Intel Magazine*, July, 2004.

225 Shiloh, Michael, *Teleo: Rapid Prototyping Toolkit*. San Francisco, CA. http://www.makingthings.com/teleo.htm

226 Shipman, Frank M. and Haowei Hseih. Navigable History: A Reader's View of Writer's Time. *The New Review of Hypermedia and Multimedia* **6**(1). pp. 147–67, 2000.

227 Shneiderman, Ben. Empirical Studies of Programmers: The Territory, Paths, and Destinations. In Proceedings of *First Workshop on Empirical Studies of Programmers*. Washington, DC: Ablex Pub. pp. 1–12, June 5–6, 1986.

228 Shneiderman, Ben and Catherine Plaisant, *Designing the User Interface*. 4th ed. Boston: Addison-Wesley. 652 pp. 2004.

229 Sinha, Anoop K., Scott R. Klemmer, and James A. Landay. Embarking on Spoken-Language NL Interface Design. *The International Journal of Speech Technology* **5**(2). pp. 159–69, 2002.

230 Sinha, Anoop K. and James A. Landay. Capturing User Tests in a Multimodal, Multidevice Informal Prototyping Tool. In Proceedings of *ICMI-PUI: ACM International Conference on Multimodal Interfaces*. Vancouver, BC: ACM Press. pp. 117–24, 2003.

231 Smith, David Canfield, Charles Irby, Ralph Kimball, Bill Verplank, and Eric Harslem. Designing the Star User Interface, *Byte*, vol. 7(4): pp. 242–82, April, 1982.

232 Snibbe, Scott S., Karon E. MacLean, Rob Shaw, Jayne Roderick, William L. Verplank, and Mark Scheeff. Haptic Techniques for Media Control. UIST: ACM Symposium on User Interface Software and Technology, *CHI Letters* **3**(2). pp. 199–208, 2001.

233 Spielberg, Steven, *Survivors of the Shoah Visual History Foundation*. http://www.vhf.org

234 Stallman, Richard, *GNU Emacs: The extensible self-documenting text editor*, 1993. Free Software Foundation: Cambridge, MA. ftp://prep.ai.mit.edu/pub/gnu

235 Starr, Louis, Oral History, in *Encyclopedia of Library and Information Sciences*, P.R. Penland, Editor. Marcel Dekker: New York. p. 440–63, 1977.

236 Stearns, Beth, *Java Native Interface*. Sun Microsystems, Inc. http://java.sun.com/docs/books/tutorial/native1.1

237 Stifelman, Lisa, Barry Arons, and Chris Schmandt. The Audio Notebook: Paper and Pen Interaction with Structured Speech. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **3**(1). pp. 182–89, 2001.

238 Streitz, Norbert A., Jörg Geißler, Torsten Holmer, Shin'ichi Konomi, Christian Müller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-LAND: An interactive Landscape for Creativity and Innovation. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. Pittsburgh, PA: ACM Press. pp. 120-27, May 15-20, 1999.

239 Suchman, Lucy A., Conversation as "ensemble" work, in *Plans and Situated Actions*. Cambridge University Press: Cambridge. p. 72, 1987.

240 Sutherland, Ivan E. SketchPad: A Man-Machine Graphical Communication System. In Proceedings of *AFIPS Spring Joint Computer Conference*. pp. 329–46, 1963.

241 Szekely, Pedro. Retrospective and challenges for model-based interface development. In Proceedings of *Eurographics Workshop on Design, Specification and Verification of Interactive Systems*. pp. 1-27, 1996.

242 Tang, John C. and Scott L. Minneman. VideoWhiteboard: video shadows to support remote collaboration. In Proceedings of *CHI: Human Factors in Computing Systems*. New Orleans, LA: ACM Press. pp. 315–22, 27 April-2 May, 1991.

243 Terry, Michael and Elizabeth D. Mynatt. Supporting Experimentation with Side Views. *Communications of the ACM* **45**(10). pp. 106-08, 2002.

244 Terry, Michael, Elizabeth D. Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. Variation in Element and Action: Supporting Simultaneous Development of Alternative Solutions. CHI: ACM Conference on Human Factors in Computing Systems, *CHI Letters* **6**(1). pp. 711–18, 2004.

245 Torok, Gabor P. and Andrew B. White. Remote chalkboard automatic cursor. US Patent 4,317,956, 1982, Bell Telephone Laboratories, Incorporated

246 Ullmer, Brygg, *Tangible Interfaces for Manipulating Aggregates of Digital Information*, Unpublished PhD, MIT, Media Arts and Sciences, Cambridge, MA, 2002. http://xenia.media.mit.edu/~ullmer/dissertation

247 Ullmer, Brygg and Hiroshi Ishii. Emerging Frameworks for Tangible User Interfaces. *IBM Systems Journal* **39**(3&4). pp. 915-31, 2000.

248 Ullmer, Brygg and Hiroshi Ishii, Emerging Frameworks for Tangible User Interfaces, in *Human-Computer Interaction in the New Millennium*, J.M. Carroll, Editor. Addison-Wesley. p. 579–601, 2001.

249 Ullmer, Brygg and Hiroshi Ishii. The metaDESK: Models and Prototypes for Tangible User Interfaces. In Proceedings of *UIST: User Interface Software and Technology*: ACM Press. pp. 223–32, October, 1997.

250 Ullmer, Brygg, Hiroshi Ishii, and Dylan Glas. mediaBlocks: physical containers, transports, and controls for online media. In Proceedings of *SIGGRAPH 98: 25th International Conference on Computer Graphics and Interactive Techniques*. Orlando, FL: ACM Press. pp. 379–86, 19–24 July, 1998.

251 Ulrich, Iwan and Illah Nourbakhsh. Firewire Untethered: High-Quality Images for Notebook Computers. *Advanced Imaging Magazine*. pp. 69-70, 2000.

252 Underkoffler, John and Hiroshi Ishii. Urp: A Luminous-Tangible Workbench for Urban Planning and Design. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. Pittsburgh, PA: ACM Press. pp. 386–93, May 15–20, 1999.

253 Underkoffler, John, Brygg Ullmer, and Hiroshi Ishii. Emancipated Pixels: Real-World Graphics in the Luminous Room. In Proceedings of *SIGGRAPH: Computer Graphics and Interactive Techniques*. Los Angeles, CA: ACM Press. pp. 385-92, August 8-13, 1999.

254 Wallace, Victor L. The semantics of graphic input devices. In Proceedings of *ACM Symposium on Graphic Languages*. Florida: ACM Press. pp. 61–65, 1976.

255 Want, Roy, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging Physical and Virtual Worlds With Electronic Tags. In Proceedings of *CHI: Human Factors in Computing Systems*. Pittsburgh, PA: ACM Press. pp. 370-77, 1999.

256 Want, Roy, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser. An overview of the PARCTAB Ubiquitous Computing Experiment. *IEEE Personal Communications* **2**(6). pp. 28–43, 1995.

257 Weiser, Mark. The Computer for the 21st Century. *Scientific American* **265**. pp. 94–104, 1991.

258 Weiser, Mark and John Seely Brown, The Coming Age of Calm Technology, in *Beyond Calculation: The Next Fifty Years of Computing*, P.J. Denning and R.M. Metcalfe, Editors. Copernicus Books: New York, NY. p. 331, 1997.

259 Weld, Daniel S., Corin Anderson, Pedro Domingos, Oren Etzioni, Krzysztof Gajos, Tessa Lau, and Steve Wolfman. Automatically Personalizing User Interfaces. In Proceedings of *IJCAI: International Joint Conference on Artificial Intelligence*, 2003.

260 Wellner, Pierre. Interacting with Paper on the DigitalDesk, *Communications of the ACM*, vol. 36(7): pp. 87–96, July, 1993.

261 Wellner, Pierre and Steve Freeman, *The DoubleDigitalDesk: Shared Editing of Paper Documents*. Technical Report EPC-93-108, EuroPARC 1993.

262 Whittaker, Steve and Heinrich Schwarz, Back to the Future: Pen and Paper Technology Supports Complex Group Coordination, in *CHI: Human Factors in Computing Systems*: ACM Press. p. 495–502, 1995.

263 Wilcox, Lynn D., Bill N. Schilit, and Nitin Sawhney. Dynomite: A dynamically organized ink and audio notebook. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. Atlanta, GA: ACM Press. pp. 186–93, March 22–27, 1997.

APPENDIX

A The Designers' Outpost Evaluation Questionnaires

1 Core Outpost Functionality

This questionnaire was used in the study conducted in SECTION 3.5

1 How likely is it that you would integrate Outpost as a regular part of your web site design process?

O Very Likely

O Somewhat Likely

O Neither Likely nor Unlikely

O Somewhat Unlikely

O Very Unlikely

2 If you would use Outpost as part of your design process, what would you use it for?

3 What would you see as the primary advantage of using Outpost as compared with your current practices?

4 What effect do you think incorporating Outpost into your process would have on each of the following factors? (Use a scale from 0 to 10 where 0 means "A very negative effect" and 10 means "A very positive effect.")

a) Ability to communicate with clients

b) Ability to communicate with programming/development team members

c) Ability to communicate with information architects

d) Ability to communicate with user interface designers

e) Ability to communicate with graphic designers

f) Ability to communicate with copywriters

g) Ability to communicate with design team members

h) Ability to communicate with internal managers

i) Ability to communicate with usability engineers/testers

j) Ability to communicate with users

k) Ability to conduct usability tests

l) Overall expressiveness

m) Overall efficiency

5 What aspects of Outpost do you particularly *like*?

6 What aspects of Outpost do you particularly *dislike*?

7 What additional features would you like to see in Outpost? What would it take for Outpost to be truly useful to you?

8 Which of the following activities do you engage in as part of your current web site design responsibilities?

[ ] Information architecture

[ ] Navigation design

[ ] User interface design

[ ] Interaction design

[ ] Graphic design

[ ] Usability testing

[ ] Usability evaluation (*e.g.*, Heuristic Evaluation, Cognitive Walkthroughs)

[ ] Project management

[ ] Web site programming/development

[ ] Development of interactive prototypes

[ ] Copywriting

[ ] Other

9 Which of the following best describes your primary responsibility with respect to web site design?

O Information architecture

O Navigation design

O User interface design

O Interaction design

O Graphic design

O Usability testing

O Usability evaluation (*e.g.*, Heuristic Evaluation, Cognitive Walkthroughs)

O Project management

O Web site programming/development

O Development of interactive prototypes

O Copywriting

O Other

10 What percentage of your current workload is made up of web site design projects?

O 0–25%

O 26–50%

O 51–75%

O 76–100%

11 What other types of design projects are part of your current workload?

[ ] User interface design for non-web-based productivity software

[ ] Game design

[ ] Other multimedia design

[ ] Print design

[ ] Industrial design

[ ] Other type of design

12 How long have you been involved in web site design?

O Less than one year

O 1-2 years

O 2-3 years

O 3-5 years

O More than 5 years

13 How long have you been involved in any kind of design (including user interface, print, multimedia, etc.)?

O Less than one year

O 1-2 years

O 2-3 years

O 3-5 years

O 5-10 years

O More than 10 years

14 Other than web site design, which of the following best describes your background?

O Graphic design

O Industrial design

O User interface design for non-web software

O Other design

O Programming/Computer Science

O Other engineering

O Business

O Other

15 Including ongoing projects, how many web sites have you helped design?

O 1-5

O 6-10

O 11-20

O 20-50

O More than 50

16 What is the highest level of education that you have completed?

O Some High School

O High School Diploma

O Some College

O College Degree

O Some Graduate School

O Graduate or Professional Degree

17 Which of the following degrees/higher education experiences do you have?

[ ] A.A., field:

[ ] Professional Certificate, field:

[ ] B.A., field:

[ ] B.F.A., field:

[ ] B.S., field:

[ ] Other bachelor's or equivalent

[ ] Some Graduate School, field:

[ ] M.A., field:

[ ] M.F.A. field:

[ ] M.S., field:

[ ] M.F.A., field:

[ ] Ph.D., field:

[ ] Other graduate or professional degrees

2 Design History Mechanism

This questionnaire was used in the study conducted in SECTION 4.7

1 How likely is it that you would integrate Outpost as a regular part of your web site design process?

O Very Likely

O Somewhat Likely

O Neither Likely nor Unlikely

O Somewhat Unlikely

O Very Unlikely

2 If you would use Outpost as part of your design process, what would you use it for?

3 What would you see as the primary advantage of using Outpost as compared with your current practices?

4 What effect do you think incorporating Outpost into your process would have on each of the following factors? (Use a scale from 0 to 10 where 0 means "A very negative effect" and 10 means "A very positive effect.")

a) Ability to communicate with clients

b) Ability to communicate with programming/development team members

c) Ability to communicate with information architects

d) Ability to communicate with user interface designers

e) Ability to communicate with graphic designers

f) Ability to communicate with copywriters

g) Ability to communicate with design team members

h) Ability to communicate with internal managers

i) Ability to communicate with usability engineers/testers

j) Ability to communicate with users

k) Ability to conduct usability tests

l) Overall expressiveness

m) Overall efficiency

5 What aspects of **Outpost** do you particularly *like*?

6 What aspects of **Outpost** do you particularly *dislike*?

7 What aspects of Outpost's **history system** do you particularly *like*?

8 What aspects of Outpost's **history system** do you particularly *dislike*?

9 What additional features would you like to see in Outpost? What would it take for Outpost to be truly useful to you?

10 Which of the following activities do you engage in as part of your current web site design responsibilities?

[ ] Information architecture

[ ] Navigation design

[ ] User interface design

[ ] Interaction design

[ ] Graphic design

[ ] Usability testing

[ ] Usability evaluation (*e.g.*, Heuristic Evaluation, Cognitive Walkthroughs)

[ ] Project management

[ ] Web site programming/development

[ ] Development of interactive prototypes

[ ] Copywriting

[ ] Other

11 Which of the following best describes your primary responsibility with respect to web site design?

O Information architecture

O Navigation design

O User interface design

O Interaction design

O Graphic design

O Usability testing

O Usability evaluation (*e.g.*, Heuristic Evaluation, Cognitive Walkthroughs)

O Project management

O Web site programming/development

O Development of interactive prototypes

O Copywriting

O Other

12 What percentage of your current workload is made up of web site design projects?

O 0-25%

O 26-50%

O 51-75%

O 76-100%

13 What other types of design projects are part of your current workload?

[ ] User interface design for non-web-based productivity software

[ ] Game design

[ ] Other multimedia design

[ ] Print design

[ ] Industrial design

[ ] Other type of design

14 How long have you been involved in web site design?

O Less than one year

O 1-2 years

O 2-3 years

O 3-5 years

O More than 5 years

15 How long have you been involved in any kind of design (including user interface, print, multimedia, etc.)?

O Less than one year

O 1-2 years

O 2-3 years

O 3-5 years

O 5-10 years

O More than 10 years

16 Other than web site design, which of the following best describes your background?

O Graphic design

O Industrial design

O User interface design for non-web software

O Other design

O Programming/Computer Science

O Other engineering

O Business

O Other

17 Including ongoing projects, how many web sites have you helped design?

O 1-5

O 6-10

O 11-20

O 20-50

O More than 50

18 What is the highest level of education that you have completed?

O Some High School

O High School Diploma

O Some College

O College Degree

O Some Graduate School

O Graduate or Professional Degree

19 Which of the following degrees/higher education experiences do you have?

[ ] A.A., field:

[ ] Professional Certificate, field:

[ ] B.A., field:

[ ] B.F.A., field:

[ ] B.S., field:

[ ] Other bachelor's or equivalent

[ ] Some Graduate School, field:

[ ] M.A., field:

[ ] M.F.A. field:

[ ] M.S., field:

[ ] M.F.A., field:

[ ] Ph.D., field:

[ ] Other graduate or professional degrees

3 Remote Collaboration

This questionnaire was used in the study conducted in SECTION 5.6

Background Questionnaire

1 Which of the following best describes your background?

O Graphic design

O Industrial design

O User interface design for non-web software

O Other design

O Programming/Computer Science

O Other engineering

O Business

O Other _____

2 Which of the following activities do you engage in as part of your current design responsibilities?

[ ] Information architecture

[ ] Navigation design

[ ] User interface design

[ ] Interaction design

[ ] Graphic design

[ ] Usability testing

[ ] Usability evaluation (e.g. Heuristic Evaluation, Cognitive Walkthroughs)

[ ] Project management

[ ] Web site programming/development

[ ] Development of interactive prototypes

[ ] Copywriting

[ ] Other _____

3 Approximately **how often** do you use:

A. a computer ? Never Daily Weekly Monthly Yearly

B. affinity diagramming or post-it information architecture

Never Daily Weekly Monthly Yearly

C. a computer diagramming language like MS Visio

Never Daily Weekly Monthly Yearly

4 Approximately **how often** do you:

A. Collaborate with other people for the majority of a project

Never Daily Weekly Monthly Yearly

B. Collaborate with someone located in a different office

Never Daily Weekly Monthly Yearly

C. Travel to a design session Never Daily Weekly Monthly Yearly

Outpost Board Questionnaire

1 Did you find the Outpost board more like a

Whiteboard **Computer**

1 2 3 4 5 6 7

2 The Outpost Board is:

	Strongly Disagree	**Disagree**	**Neutral**	**Agree**	**Strongly Agree**
Useful	1	2	3	4	5
Easy to Understand	1	2	3	4	5
Easy to Learn	1	2	3	4	5
Flexible	1	2	3	4	5
Awkward	1	2	3	4	5
Irrelevant	1	2	3	4	5
Distracting	1	2	3	4	5

3 What did you particularly **like** about the Outpost Board Interface?

4 What did you particularly **dislike** about the Outpost Board Interface? What would you change?

Outpost Desk Questionnaire

1 Did you find the Outpost desk more like a

Whiteboard **Computer**

1 2 3 4 5 6 7

2 The Outpost Desk is:

	Strongly Disagree	**Disagree**	**Neutral**	**Agree**	**Strongly Agree**
Useful	1	2	3	4	5
Easy to Understand	1	2	3	4	5
Easy to Learn	1	2	3	4	5
Flexible	1	2	3	4	5
Awkward	1	2	3	4	5
Irrelevant	1	2	3	4	5
Distracting	1	2	3	4	5

3 What did you particularly **like** about the Outpost Desk Interface?

4 What did you particularly **dislike** about the Outpost Desk Interface? What would you change?

Post-Test Questionnaire

1 How likely is it that you would integrate Outpost as a regular part of your work?

	Very Unlikely	Somewhat Unlikely	Neutral	Likely	Very Likely
A. As is	1	2	3	4	5
B. As is with two desks	1	2	3	4	5
C. As is with two boards	1	2	3	4	5
D. Ideal Situation *i.e.*, Bug Free and Adequate Performance, any desk/board configuration	1	2	3	4	5

2 I preferred the

Outpost Board Configuration **Outpost Desk Configuration**

1 2 3 4 5 6 7

3 When working with a remote person, I found the following cues:

	Very Distracting	Distracting	Neutral	Useful	Very Useful
1. Voice/audio	1	2	3	4	5
2. Blob representing other's	1	2	3	4	5

location					
3. Transient ink	1	2	3	4	5

4 Please rate these tools in order of importance to you in using this system:

(1=most important, 3 = least important)

1. Voice/audio

2. Blob representing other's location

3. Transient ink

5 Please rate these systems for **remote** collaboration:

(1 = best, 5 = worst)

A. Outpost Board _____

B. Outpost Desk _____

C. Visio _____

D. Whiteboard _____

E. Email _____

6 What aspects of Outpost do you particularly *like* or find the most useful?

7 What aspects of Outpost in general do you particularly *dislike* or want changed?

8 What additional features would you like to see in Outpost? What would it take for Outpost to be truly *useful* to you?

9 How would you compare working with *Outpost* for a **remote** design task vs. a *diagramming system* like Microsoft Visio?

10 How would you compare working with *Outpost* for a **remote** design task vs. a *whiteboard*?

APPENDIX

B Books with Voices Evaluation Questionnaire

This questionnaire was used in the study conducted in SECTION *6.6*

Background

1 What is your occupation and what discipline are you in? (*e.g.*, history, art, film, transcriber.) What institution is your occupation associated with? (*e.g.*, university, television studio, museum)

2 How many oral histories have you used in your research before?

O None

O 1-5

O 5-10

O over 10

3 How many oral histories have you conducted before?

O None

O 1-5

O 5-10

O over 10

4 Where do you read and access oral histories?

O In front of your computer at your desk

O In a quiet room in the library

O In a noisy room (*e.g.*, a café)

O While you are traveling (*e.g.*, train, plane)

O Other, please specify _____

5 How long have you been making or using oral histories?

O Less than one year

O 1-2 years

O 2-3 years

O 3-5 years

O More than 5 years

6 What is the highest level of education that you have completed?

O Some High School

O High School Diploma

O Some College

O College Degree

O Some Graduate School

O Graduate or Professional Degree

Working with Oral Histories

7 What is the average length of the oral histories that you create and/or use in your research?

O 1-25 pages

O 25-50 pages

O 50-100 pages

O 100-200 pages

O More than 200 pages

8 How often have you looked at the original audio/video recording?

O None

O 1-5

O 5-10

O over 10

9 For what purpose do you look at the original media?

O To clear up something you don't understand in the transcript

O To see the emotional state of the interviewee during a certain passage

O To understand the context of the passage

O Out of curiosity, to see and hear the interviewee

O To better understand or clarify the topic of discussion

O To verify the accuracy of the transcript

O To compare the final transcript with the original interview

O Other, please specify _____

10 What role do oral histories play in your end product? (Choose all that apply)

O Data for analysis

O Direct quotes

O Background information

O Specific historical information

O Mood setting

O Other, please specify _____

11 What part of the interview do you find most useful?

O Full interview transcript

O Full video

O Summary of transcript (if available)

O Certain passages

O Certain video clips

O Other, please specify _____

12 What information do you look for in an interview? (Choose all that apply)

O Specific dates as given by the interviewee

O Emotional state of the interviewee

O Abstract themes within the interview

O Concrete ideas dealt with in the interview

13 Currently, how do you read oral histories?

O From start to finish slowly

O Skim briefly from start to finish

O Only read certain sections carefully

O Read/skim over more than once

Do you ever skip sections? If so, why?

14 Why do you use oral histories?

O For background information

O Out of your own personal interest

O To use as an example in a class

O As a source for a research paper

O To understand a specific historical event

O To learn more about a specific person

O Other, please specify _____

15 Write a little bit about the kinds of information you gather from oral histories, e.g., dates, names, references to specific events, quotes, …

16 In your research, do you ever share sections of oral histories with others?

O No.

O Very occasionally.

O Somewhat regularly.

O All the time.

17 Please attach to this questionnaire a sample of how you've used an oral history in your research. (*e.g.*, quotes) How was oral history incorporated into this sample?

Books with Voices software

18 Were the barcodes in convenient places for you while scanning the video? Was the spacing of the barcodes acceptable? If not, how often would you like to see a barcode?

O I'd like barcodes a lot more often

O I'd like barcodes a bit more often

O The number of barcodes was about right

O I'd like barcodes a bit less often

O I'd like barcodes a lot less often

19 Do you have any thoughts on the aesthetics or visual design of the paper transcripts?

20 Was there a significant difference between how the video and the transcript helped you understand the context? Which did you use, the video or the transcript?

O Only video

O Only transcript

O Used the video more than the transcript

O Used the transcript more than the video

O Used both equally to judge the content

Did you find yourself confused by either one or the other? If so, which?

O Video O Transcript

21 Would you want to have a summary of the interview (*e.g.*, timeline, abstract) as well as the full transcript? Or would it make you less motivated to read through the full interview?

22 Do you feel that Books with Voices would change your usage practice? How?

23 How likely is it that you would use Books with Voices while reading oral histories?

O Very Likely

O Somewhat Likely

O Neither Likely nor Unlikely

O Somewhat Unlikely

O Very Unlikely

24 What effect do you think incorporating this system into your process would have on each of the following factors? (Use a scale from 0 to 10 where 0 means "A very negative effect" and 10 means "A very positive effect.")

a.) Ability to understand the full context of the interview

b.) Ability to find specific information

c.) Ability to search through the interview

d.) Overall efficiency

25 What aspects of the system do you particularly like?

26 What aspects of the system do you particularly dislike?

27 How did it feel to use this device? Did you find the system easy to use? What aspects of the system were confusing?

28 Would you like headphones as an additional feature? Did you feel a need for volume control?

29 What other functionality would you like the device to have? (Choose all that apply)

O Take notes

O Keep a record of clips that you scanned

O Email you with information from the interview

O Other, please specify _____

30 What additional features would you like to see in the system? What would it take for the system to be truly useful to you?

31 What role, if any, do you think Books with Voices could play in editing transcripts?

32 What role, if any, do you think Books with Voices could play in data/quote gathering for your research?

33 What would be the largest barrier to your using the software?

O Converting the video or audio interviews into digital format.

O Creating time-stamped transcriptions.

O Acquiring a PDA with a barcode scanner.

O Legal issues (*e.g.*, clearance from the interviewee)

O Other _____

34 What are your thoughts on using Books with Voices to help you keep track of important parts of an oral history? (*e.g.*, the software could keep a list of video sections that you've watched, or swiping a section with a barcode could add it to a list.) Would something like this be valuable? How might it work?

35 Would the above mentioned functionality be useful for sharing? (*e.g.*, emailing sections to someone when they're swiped with a barcode.) How might it work?

APPENDIX

C Papier-Mâché Fieldwork Questionnaire

This questionnaire was used in the fieldwork conducted in CHAPTER 7

Background

1 How did you get the idea for your system?

2 Did you speak with any users before building your system? If so, who did you speak with, and what did you learn?

3 Please summarize what interaction techniques your system offers. Include a reference to a paper/URL describing your system if you like.

4 What kinds of prototypes did you create? (*i.e.*, Paper prototypes, rapid software prototypes, etc.). Briefly, what did you learn from each prototype?

5 What were the main principles you kept in mind during the design and implementation phase?

6 What was the application that you wanted to build, but couldn't due to technology limitations?

7 What are your user conceptual models in your application? What are these models, and how did you come up with them?

Planning and Organizational Structure:

8 What was the time frame for your project?

9 How many people worked on the project? What were their backgrounds, and what role did they play in the project?

10 What types of problems did you run into during software integration? How often did your software team integrate code?

Software Design

11 What hardware did you use? Why did you choose this hardware?

12 Please draw a diagram of how the software works. (Feel free to use another sheet of paper.) Are the different components self sufficient, or do they depend on one another?

13 Why did you choose this software design? What were the downfalls of other choices?

14 Over the course of the project, what parts of the system were refactored or redesigned? Why?

15 What types of events are used in your implementation?

16 What software tools did you use, if any?

17 What code did you reuse from prior projects or prototypes? Or did you find yourself rejecting code from prior prototypes?

18 What programming language(s) did you use? Why?

19 What other types of possible applications were already available for your use?

User and System Evaluation:

20 What types of system tests and performance tests did you run? What metrics were measured in these tests? What performance goals were difficult or important to achieve?

21 What tests did you run with users? What did the users find that worked well? What problems did the users find?

22 What would you have done differently in building your system? (*i.e.*, user interactions or design)

Difficulties

23 What was most frustrating about the development process? And what was the must frustrating about the design process?

24 What types of unexpected problems did you run into during implementation?

25 What problems were created due to the hardware you chose?

26 What were the software/hardware performance requirements for your system? Was meeting these requirements a challenge? (*e.g.*, sensor responsiveness or resolution.) How?

27 What was the hardest part of design? Of implementation? Of testing?

28 Do you have any specific requests or suggestions for functionality that a toolkit could provide?

APPENDIX

D Sample Papier-Mâché Applications

1 Marble Answering Machine

```
import edu.berkeley.guir.papier_mache.*;
import edu.berkeley.guir.papier_mache.assoc.*;
import edu.berkeley.guir.papier_mache.assoc.classifier.*;
import edu.berkeley.guir.papier_mache.assoc.factory.*;
import edu.berkeley.guir.papier_mache.assoc.factory.
     mediaclip.AudioClip;
import edu.berkeley.guir.papier_mache.rfid.RFIDManager;


/**
 * @author  Jack Li
 *
 */
public class MarbleAnswering extends
     DefaultAssociationFactory {
 public static void main(String args[]) {
  PapierMache.displayMonitoringWindow();
  final RFIDManager manager = RFIDManager.getManager();
  final BindingManager assocMap = new BindingManager();

  final ObjectClassifier classifier = new
     AcceptAllClassifier();
  final AssociationFactory factory = new StringFactory();
  assocMap.addClassifierFactoryPair(classifier, factory);
  manager.addPhobListenerForAllProducers(assocMap);
 }

 public AssociationElt createAssociationEltForPhob(final Phob
     phob) {
  return new AudioClip();
 }

 public String toShortString() {
  return "MediaFactory";
 }
}
```

2 Books with Voices

RFID Version

```
import edu.berkeley.guir.papier_mache.*;
import edu.berkeley.guir.papier_mache.assoc.*;
import edu.berkeley.guir.papier_mache.assoc.classifier.*;
import edu.berkeley.guir.papier_mache.assoc.factory.
     AssociationElt;
import edu.berkeley.guir.papier_mache.assoc.factory.
     mediaclip.*;
import edu.berkeley.guir.papier_mache.rfid.RFIDManager;

/**
 * @author  Jack Li
 */
public class BooksWithVoices {
 public BooksWithVoices() {
  PapierMache.displayMonitoringWindow();
  final RFIDManager manager = RFIDManager.getManager();
  final BindingManager assocMap = new BindingManager();

  final MediaClip media = new VideoClip();
  final ObjectClassifier secondProducer = new
     AcceptAllClassifier();
  final AssociationFactory randomAccessTimepoints = new
     AssociationFactory() {

   public AssociationElt createAssociationEltForPhob(Phob
     phob) {
    return new RandomAccessPlayElt(media);
   }

   public String toShortString() {
    return "Random Access Times";
   }
  };
  assocMap.addClassifierFactoryPair(secondProducer,
    randomAccessTimepoints);

  manager.addPhobListenerForAllProducers(assocMap);
 }

 public static void main(String[] args) {
  new BooksWithVoices();
 }
}
```

Barcode Version

```
import edu.berkeley.guir.papier_mache.*;
import edu.berkeley.guir.papier_mache.assoc.*;
import edu.berkeley.guir.papier_mache.assoc.classifier.*;
import edu.berkeley.guir.papier_mache.assoc.factory.
     AssociationElt;
```

```
import edu.berkeley.guir.papier_mache.assoc.factory.
     mediaclip.*;
import edu.berkeley.guir.papier_mache.barcode.
     BarcodePhobProducer;
import edu.berkeley.guir.papier_mache.input.*;
import edu.berkeley.guir.papier_mache.input.CameraImageInput.
     NoCameraFoundException;
import edu.berkeley.guir.papier_mache.vision.
     VisionPhobProducer;

/**
 * @author  Jack Li
 */
public class BooksWithVoicesBarcode {
 public BooksWithVoicesBarcode() {
  PapierMache.displayMonitoringWindow();
  ImageInputDevice imageInput;
  try {
   imageInput = new CameraImageInput();
  } catch (NoCameraFoundException e) {
   System.out.println("Error: no camera found");
   return;
  }
  final VisionPhobProducer visionGen = new
     VisionPhobProducer(imageInput);
  final PhobProducer producer = new
     BarcodePhobProducer(visionGen);
  final BindingManager assocMap = new BindingManager();

  final MediaClip media = new VideoClip();
  final ObjectClassifier secondProducer = new
     AcceptAllClassifier();
  final AssociationFactory randomAccessTimepoints = new
     AssociationFactory() {

   public AssociationElt createAssociationEltForPhob(Phob
     phob) {
    return new RandomAccessPlayElt(media);
   }

   public String toShortString() {
    return "Random Access Times";
   }
  };
  assocMap.addClassifierFactoryPair(secondProducer,
    randomAccessTimepoints);

  producer.addPhobListener(assocMap);
 }

 public static void main(String[] args) {
  new BooksWithVoicesBarcode();
 }
}
```

3 Collaborage

Run.java

```
package edu.berkeley.guir.examples.collaborage;
import edu.berkeley.guir.papier_mache.Phob;
import edu.berkeley.guir.papier_mache.barcode.*;
import edu.berkeley.guir.papier_mache.event.*;
import edu.berkeley.guir.papier_mache.input.*;
import edu.berkeley.guir.papier_mache.vision.*;

/**
 * @author  Andy Kung
 */
public class Run implements PhobListener {
 private static final int SLEEP_MILLISECS = 15;
 private static final int MIN_OBJECT_SIZE = 200;

 //TABLE DATA: Assume these data are constant and given.
 private static final int TABLE_X = 0;
 private static final int TABLE_Y = 0;
 private static final int TABLE_WIDTH = 344;
 private static final int TABLE_HEIGHT = 504;
 private static final int IN_COL_WIDTH = TABLE_WIDTH / 3;
 private static final int OUT_COL_WIDTH = TABLE_WIDTH / 3;
 private static final int COMMENT_WIDTH = TABLE_WIDTH / 3;
 private static final int OFFSET = 20;

 //Camera or image files
 private static VisionPhobProducer visionGen;

 //COMPONENTS
 private static Log log;
 private static Network network;
 private static CommentImage commentImage;

 public static void main(String[] arg) {

  //Used with image files

  visionGen = new VisionPhobProducer(new
     FilesImageInput("C:\\dev\\production\\Collages\\test\\te
     st"));

  //Used with a camera
  /*
  final VisionPhobProducer visionGen = new
     VisionPhobProducer(
   new CameraImageInput(SLEEP_MILLISECS), MIN_OBJECT_SIZE);
  */
  final BarcodePhobProducer barcodeGen =
   new BarcodePhobProducer(visionGen, BarcodePhob.CYBERCODE);

  barcodeGen.addPhobListener(new Run());
  visionGen.addPhobListener(new Run());
```

```
 //Network Database
 //network = new Network();

 //Comment Image
 commentImage = new CommentImage();

 //Log
 log = new Log();

}

public void phobAdded(PhobEvent event) {
 Phob phob = event.getPhob();

 if (phob instanceof BarcodePhob) {
  String id = ((BarcodePhob) phob).getTagID();
  VisionPhobCollection barcodeVision =
   ((BarcodePhob) phob).getVisionPhobCollection();
  if (isInInColumn((VisionPhob) barcodeVision)) {
   System.out.println("Type "
         + Integer.toString(((BarcodePhob) phob).getType())
         + " Barcode "
         + id
         + " Added in IN Column.");
   //Log Action
   Person p = network.getUser(id);
   log.writeLog("[id: " + id + "] " + p.name_first + " " +
    p.name_last + " is IN.");
   network.insertToBoard(id, p.name_first+" "+p.name_last,
    true, "hello world");
   //commentImage.writeJPG("hi", visionGen.getImage());
  } else if (
   isInOutColumn((VisionPhob) barcodeVision)) {
   System.out.println(
    "Type "
         + Integer.toString(((BarcodePhob) phob).getType())
         + " Barcode "
         + id
         + " Added in OUT Column.");
   //Log Action
   Person p = network.getUser(id);
   log.writeLog("[id: " + id + "] " + p.name_first + " " +
    p.name_last + " is OUT.");
   network.insertToBoard(id, p.name_first+" "+p.name_last,
    false, "hello world");

  }

 } else if (phob instanceof VisionPhob) {
  if (isInInColumn((VisionPhob) phob)) {
   //System.out.println("Non-Barcode Phob Added in IN
    Column.");
  } else if (
   isInOutColumn((VisionPhob) phob)) {
```

```
    //System.out.println("Non-Barcode Phob Added in OUT
     Column.");
   }
   if (isInCommentColumn((VisionPhob) phob)) {
    //System.out.println("Comment Phob Added.");
   }

  }
}

public void phobUpdated(PhobEvent event) {}

public void phobRemoved(PhobEvent event) {
 Phob phob = event.getPhob();
 if (phob instanceof BarcodePhob) {
  //Use this block when testing with a camera!
  String id = ((BarcodePhob) phob).getTagID();
  VisionPhobCollection barcodeVision =
   ((BarcodePhob) phob).getVisionPhobCollection();
  if (isInInColumn((VisionPhob) barcodeVision)) {
   System.out.println(
    "Type "
         + Integer.toString(((BarcodePhob) phob).getType())
         + " Barcode "
         + id
         + " Removed in IN Column.");
  } else if (
   isInOutColumn((VisionPhob) barcodeVision)) {
   System.out.println(
    "Type "
         + Integer.toString(((BarcodePhob) phob).getType())
         + " Barcode "
         + id
         + " Removed in OUT Column.");
  }
  //Log Action

  Person p = network.getUser(id);
  log.writeLog("[id: " + id + "] " + p.name_first + " " +
    p.name_last + " is REMOVED.");
  network.deleteFromBoard(id);

 } else if (phob instanceof VisionPhob) {
  if (isInInColumn((VisionPhob) phob)) {
   //System.out.println("Non-Barcode Phob Removed in IN
    Column.");
  } else if (
   isInOutColumn((VisionPhob) phob)) {
   //System.out.println("Non-Barcode Phob Removed in OUT
    Column.");
  } else if (isInCommentColumn((VisionPhob) phob)) {
   //System.out.println("Comment Phob Removed.");
  }
 }
}
```

```
 public boolean isInTable(VisionPhob phob) {
  int x = phob.getBounds().x;
  int y = phob.getBounds().y;
  int width = phob.getBounds().width;
  int height = phob.getBounds().height;
  if (!((x > TABLE_X) && (x + width < TABLE_X +
     TABLE_WIDTH))) {
   return false;
  }
  if (!((y > TABLE_Y) && (y + height < TABLE_Y +
     TABLE_HEIGHT))) {
   return false;
  }
  return true;
 }

 public boolean isInInColumn(VisionPhob phob) {
  int x = phob.getBounds().x;
  int width = phob.getBounds().width;
  if (!((x > TABLE_X - OFFSET) && (x + width < TABLE_X +
     IN_COL_WIDTH + OFFSET))) {
   return false;
  }
  return true;
 }

 public boolean isInOutColumn(VisionPhob phob) {
  int x = phob.getBounds().x;
  int width = phob.getBounds().width;
  if (!((x > TABLE_X + IN_COL_WIDTH - OFFSET)
   && (x + width < TABLE_X + IN_COL_WIDTH + OUT_COL_WIDTH +
     OFFSET))) {
   return false;
  }
  return true;
 }

 public boolean isInCommentColumn(VisionPhob phob) {
  int x = phob.getBounds().x;
  int width = phob.getBounds().width;
  if (!((x > TABLE_X + IN_COL_WIDTH + OUT_COL_WIDTH)
   && (x + width < TABLE_X + IN_COL_WIDTH + OUT_COL_WIDTH +
     COMMENT_WIDTH))) {
   return false;
  }
  return true;
 }
}
```

Network.java

```
package edu.berkeley.guir.examples.collaborage;

import java.sql.*;

/**
```

```
 * @author  Andy Kung
 */

//Network class requires org.gjt.mm.mysql.Driver.
//The database include 2 tables, collaborage and
//collaborage_users.
//Database currently running on www.andykung.net
public class Network {
 Connection con;
 String DATABASE, LOGIN, PASSWORD;

 //Establish connection information
 public Network() {
  DATABASE = "jdbc:mysql://mysql.andykung.net/net_andykung";
  LOGIN = "andykung_net";
  PASSWORD = " ";
 }

 //Query user database and find out user names.
 public Person getUser(String id_tag) {
  Person p = new Person("00000", "Unknown", "Tag");
  try {
   new org.gjt.mm.mysql.Driver();
   Connection test = DriverManager.getConnection(DATABASE,
     LOGIN,
     PASSWORD);
   PreparedStatement stm;
   stm = test
     .prepareStatement("select * from collaborage_users where
     TAG=?;");
   stm.setString(1, id_tag);
   ResultSet rs = stm.executeQuery();
   con = DriverManager.getConnection(DATABASE, LOGIN,
     PASSWORD);

   if (rs.next()) {
    p = new Person(id_tag, rs.getString("FIRST_NAME"), rs
          .getString("LAST_NAME"));
   } else {
    System.out.println("Error: Tag ID Not Found in
     Database.");
   }
   con.close();
  } catch (SQLException sqlException) {
   System.out.println("SQL Exception");
  }
  return p;
 }

 //delete all entries in the collaborage table.
 public void clearBoard() {
  try {
   new org.gjt.mm.mysql.Driver();
   con = DriverManager.getConnection(DATABASE, LOGIN,
     PASSWORD);
```

```
  Statement updateHost = con.createStatement();
  updateHost.executeUpdate("delete from collaborage");

  con.close();
 } catch (SQLException sqlException) {
  System.out.println("SQL Exception");
 }
}

//insert new entries in the collaborage table
public void insertToBoard(String tag, String name, boolean
    in,
  String comments) {
 String in_name = " ";
 String out_name = " ";
 if (in) {
  in_name = name;
  out_name = "-";
 } else {
  in_name = "-";
  out_name = name;
 }
 try {
  new org.gjt.mm.mysql.Driver();
  Connection test = DriverManager.getConnection(DATABASE,
    LOGIN,
    PASSWORD);
  PreparedStatement stm;
  stm = test
    .prepareStatement("select * from collaborage where
    TAG=?;");
  stm.setString(1, tag);
  ResultSet rs = stm.executeQuery();

  PreparedStatement updateHost;
  con = DriverManager.getConnection(DATABASE, LOGIN,
    PASSWORD);
  if (rs.next()) {
   updateHost = con
         .prepareStatement("update collaborage set IN_NAME
    = ?, OUT_NAME = ?, COMMENTS = ? where TAG = ?;");
   updateHost.setString(1, in_name);
   updateHost.setString(2, out_name);
   updateHost.setString(3, comments);
   updateHost.setString(4, tag);
   updateHost.executeUpdate();
  } else {
   updateHost = con
         .prepareStatement("insert into collaborage(TAG,
    IN_NAME, OUT_NAME, COMMENTS) values(?,?,?,?);");
   updateHost.setString(1, tag);
   updateHost.setString(2, in_name);
   updateHost.setString(3, out_name);
   updateHost.setString(4, comments);
   updateHost.executeUpdate();
  }
```

```
  con.close();
  test.close();
 } catch (SQLException sqlException) {
  System.out.println("SQL Exception");
 }
}

//update collaborage table
public void updateBoard(String tag, String name, boolean in,
    String comments) {
 String in_name = " ";
 String out_name = " ";
 if (in) {
  in_name = name;
  out_name = "-";
 } else {
  in_name = "-";
  out_name = name;
 }
 try {
  new org.gjt.mm.mysql.Driver();
  con = DriverManager.getConnection(DATABASE, LOGIN,
    PASSWORD);

  PreparedStatement updateHost;
  updateHost = con
    .prepareStatement("update collaborage set IN_NAME = ?,
    OUT_NAME = ?, COMMENTS = ? where TAG = ?;");
  updateHost.setString(1, in_name);
  updateHost.setString(2, out_name);
  updateHost.setString(3, comments);
  updateHost.setString(4, tag);
  updateHost.executeUpdate();

  con.close();
 } catch (SQLException sqlException) {
  System.out.println("SQL Exception");
 }
}

//delete entries from collaborage table.
public void deleteFromBoard(String tag) {
 try {
  new org.gjt.mm.mysql.Driver();
  con = DriverManager.getConnection(DATABASE, LOGIN,
    PASSWORD);

  PreparedStatement updateHost;
  updateHost = con
    .prepareStatement("delete from collaborage where TAG =
    ?");
  updateHost.setString(1, tag);
  updateHost.executeUpdate();

  con.close();
 } catch (SQLException sqlException) {
```

```
    System.out.println("SQL Exception");
   }
  }

}
```

Log.java

```
package edu.berkeley.guir.examples.collaborage;

import java.io.*;
import java.util.Date;

/**
 * @author  Andy Kung
 */

//The Log class creates a txt file every day and keeps track
     of every session.
public class Log {
 File logFile;

 RandomAccessFile fd;

 int ptr;

 String lineSeparator;

 public Log() {
  lineSeparator = (String) java.security.AccessController
    .doPrivileged(new sun.security.action.GetPropertyAction(
          "line.separator"));
  try {
   logFile = new File("log/" + getDate() + "_away_log.txt");
   fd = new RandomAccessFile("log/" + getDate() +
     "_away_log.txt",
     "rw");
   fd.seek(fd.length());
   writeLog("[ Log session starts ]", 1, 1);
  } catch (IOException e) {

  }
 }

 // Write a string to log file
 public void writeLog(String s) {
  try {
   fd.writeBytes(getTime() + " ---> " + s);
   fd.writeBytes(lineSeparator);
  } catch (IOException e) {

  }
 }

 // Write a string to log file with s blank lines before
 // and j blank lines after the string.
```

```
public void writeLog(String s, int i, int j) {
 try {
  while (i >= 0) {
   fd.writeBytes(lineSeparator);
   i--;
  }
  fd.writeBytes(getTime() + " ---> " + s);
  while (j >= 0) {
   fd.writeBytes(lineSeparator);
   j--;
  }
 } catch (IOException e) {

 }
}

// close log file.
public void closeLog() {
 try {
  fd.close();
 } catch (IOException e) {

 }
}

// get a date string.
public String getDate() {
 Date dateObj = new Date();
 String month = Integer.toString(dateObj.getMonth() + 1);
 if (month.length() == 1) {
  month = "0" + month;
 }
 String date = Integer.toString(dateObj.getDate());
 if (date.length() == 1) {
  date = "0" + date;
 }
 String year = Integer.toString(dateObj.getYear() + 1900);
 return month + "-" + date + "-" + year;
}

// get a time string
public String getTime() {
 Date dateObj = new Date();
 String hour;
 String am_pm;
 String minute = Integer.toString(dateObj.getMinutes());
 if (dateObj.getHours() > 12) {
  hour = Integer.toString(dateObj.getHours() - 12);
  am_pm = "pm";
 } else {
  hour = Integer.toString(dateObj.getHours());
  am_pm = "am";
 }
 if (minute.length() == 1) {
  minute = "0" + minute;
 }
```

```
  if (hour.length() == 1) {
   hour = "0" + hour;
  }
  return hour + ":" + minute + " " + am_pm;
 }

}
```

APPENDIX

E Papier-Mâché User Manual

This is the user manual presented to participants in the study described in SECTION 9.4. As some aspects of the API were changed based on the results of this study, some of the code examples are out of date with respect to the current API.

Introduction

Papier-Mâché is a toolkit for building applications with tangible interfaces.

First Application: Hello World!

For RFID technology, here's a simple application that prints "Hello World!" when a tag gets added and "Goodbye World!" when a tag gets removed:

```
import edu.berkeley.guir.papier_mache.event.*;
import edu.berkeley.guir.papier_mache.rfid.RFIDManager;

public class HelloWorld {
   public static void main(String[] args) {
      final RFIDManager manager = new RFIDManager();
      final PhobListener tagListener = new PhobListener() {
         public void phobAdded(final PhobEvent event) {
            System.out.println("Hello World!");
         }

         public void phobUpdated(PhobEvent phobEvent) {}

         public void phobRemoved(final PhobEvent event) {
            System.out.println("Goodbye World!");
         }
      };

      manager.addPhobListenerForAllReaders(tagListener);
   }
}
```

Let's dive into the first section for a quick explanation of this application.

Managers, Generators and Listeners: The essentials

The general structure of a Papier-Mache application usually involves creating a DeviceManager or PhobGenerator, creating a listener, and then adding the listener to that manager or generator. For HelloWorld.java, first create a new RFIDManager. Next, create a PhobListener that prints "Hello World!" and "Goodbye World!" when the appropriate phob events get generated; a phob (**ph**ysical **ob**ject) for RFID technology represents a tag. As such, phobAdded events occur whenever a tag gets placed on the reader, and phobRemoved events occur whenever a tag is taken off the reader. Lastly, add this PhobListener to all readers so that these events may be generated.

Vision technology works similarly:

```
import edu.berkeley.guir.papier_mache.event.*;
import edu.berkeley.guir.papier_mache.input.CameraImageSource;
import edu.berkeley.guir.papier_mache.vision.VisionPhobGenerator;
public class HelloWorld {
   private static final int MIN_OBJECT_SIZE = 200;
   private static final int SLEEP_MILLISECS = 800;

   public static void main(String[] args) {
      final VisionPhobGenerator generator = new
VisionPhobGenerator(
         new CameraImageSource(SLEEP_MILLISECS), MIN_OBJECT_SIZE);
      final PhobListener phobListener = new PhobListener() {
         public void phobAdded(final PhobEvent event) {
           System.out.println("Hello World!");
         }

         public void phobUpdated(PhobEvent phobEvent) {
         }

         public void phobRemoved(final PhobEvent event) {
           System.out.println("Goodbye World!");
         }
      };

      generator.addPhobListener(phobListener);
   }
}
```

The VisionPhobGenerator, instantiated with an ImageSource and an optional minimum object size, now replaces the RFIDManager. In addition, the VisionPhobGenerator adds the listener directly with addPhobListener. HelloWorld.java now prints "Hello World!" anytime a sizeable object gets registered in the vision system and "Goodbye World!" anytime an added object gets removed.

Associations: Mapping tags to meaningful objects and operations

Association Elements: Meaningful objects (nouns) and operations (actions)

AssociationNouns are objects that can have focus; AssociationActions operate on the AssociationNoun that has the application focus. For example, an AudioClip is an AssociationNoun whose phobAdded action is to start playing and its phobRemoved action is to stop playing. FastForwardElt is an AssociationAction that fast forwards the current focused clip.

Maps and Factories

The AssociationMap, a PhobListener, assigns associations to new tags and executes the associations to tags that have already been assigned; effectively, the map serves as a database for associations. The AssociationMap chooses how it assigns the association based upon an AssociationFactory that gets passed into its constructor. The factory may do something as simple as creating an instance of AssociationElt type A the first time, creating an instance of AssociationElt type B the second time, and so on.

The Marble Answering Machine assigns (and makes) recordings to new objects and executes (plays back) the recordings of objects that have already been assigned. This description falls in line with an AssociationMap model as a PhobListener.

```
import edu.berkeley.guir.papier_mache.Phob;
import edu.berkeley.guir.papier_mache.assoc.*;
import edu.berkeley.guir.papier_mache.event.PhobListener;
import edu.berkeley.guir.papier_mache.rfid.RFIDManager;
public class MarbleAnswering extends DefaultAssociationFactory {
   public static void main(String args[]) {
      final RFIDManager manager = new RFIDManager();
      final PhobListener assocMap = new AssociationMap(new
MarbleAnswering());
      manager.addPhobListenerForAllReaders(assocMap);
   }
   public AssociationElt createAssociationEltForPhob(final Phob
phob) {
      return new AudioClip();
   }
}
```

First, creating the manager has no difference. Using the given AssociationMap as the PhobListener requires passing in an AssociationFactory. Here the desired behavior of the factory is to return a new AudioClip upon each new phob, hence the reason for the single 'return new AudioClip()' line. Lastly, adding the phob listener has no difference.

Classifiers: A way to group tags

ObjectClassifiers implement the isMemberOfClass method, returning whether the particular Phob passed in is a member of this classification. For example, a TypeClassifier checks that the given phob is an instance of a particular class type.

VisionPhobs may get filtered out by calling the isMemberOfClass method of a TypeClassifier initialized with VisionPhob.class.

ClassifierFactoryMap

The ClassifierFactoryMap is an AssociationFactory that keeps track of all added ObjectClassifiers and their counterpart AssociationFactories. Add different ObjectClassifier-AssociationFactory pairs by calling the addClassification method. The ClassifierFactoryMap's own createAssociationEltForPhob creation method iterates through all registered ObjectClassifiers until one gets accepted; the creation method of the accepted classifier's associated factory gets called on the same phob and returned.

Here's the Marble Answering Machine that allows the user to choose what type of media element they wish to associate.

```
import edu.berkeley.guir.papier_mache.Phob;
import edu.berkeley.guir.papier_mache.assoc.*;
import
edu.berkeley.guir.papier_mache.assoc.ObjectClassifier.AcceptAllClas
sifier;
import edu.berkeley.guir.papier_mache.event.PhobListener;
import edu.berkeley.guir.papier_mache.rfid.RFIDManager;
public class MarbleAnswering extends DefaultAssociationFactory {
private static boolean firstTag = true;
public static void main(String args[]) {

         AssociationMap.registerMediaElement(AudioClip.class);
         AssociationMap.registerMediaElement(VideoClip.class);
         AssociationMap.registerMediaElement(WebPage.class);


   final RFIDManager manager = new RFIDManager();

   final ClassifierFactoryMap classMap = new
ClassifierFactoryMap();
   classMap.addClassification(new AcceptAllClassifier(), new
UserAssociationFactory(classMap));
   final PhobListener assocMap = new AssociationMap(classMap);

   manager.addPhobListenerForAllReaders(assocMap);
}
public AssociationElt createAssociationEltForPhob(final Phob phob)
{
   if (firstTag) {
      firstTag = false;
      return new PauseElt();
   }
      return new AudioClip();
   }
}
```

Events and Phobs

A Phob is the virtual representation of the physical object. One key difference between a phob and the actual physical object is that the phob knows the source that generated it. Phobs events know both the source that generated the phob and the phob itself. See the Class Reference for more details on specific phobs and phob generators.

Class Reference:

A. Classifiers (abstract class)

Given constants :
double TRUE, FALSE - used to compare

Methods to implement:
double isMemberOfClass(Phob phob) - returns TRUE (double constant) if phob is a member of this ObjectClassifer, else returns FALSE
String toShortString() - returns a short String description of an instance of this ObjectClassifier that can be displayed easily displayed in a table

Instantiable subclasses:
AcceptAllClassifier - accepts all Phobs (isMemberOfClass always returns TRUE)
IdentityClassifier - accepts only the Phob it was initialized with
MeanColorClassifier - accepts all VisionPhobs that have around the same color
ROIClassifier - accepts all VisionPhobs that have the same region of image
SameSourceClassifier - accepts only Phobs that were created from the same phob generator
SizeClassifier - accepts all VisionPhobs that have the same size
TypeClassifier - accepts only Phobs of the same class type it was initialized with

B. Factories (interface)

Methods to implement:
AssociationElt createAssociationEltForPhob(Phob phob) - returns a new instance of some AssociationElt

Instantiable subclasses:
ClassInstanceFactory - creates new instances of only the passed in class type
ClassifierFactoryMap - creates AssociationElts with the first registered accepted classifier's factory
VisualAnalogueFactory - creates AssociationWrappers used to build a visual representation of the phobs registered by the vision system
UserAssociationFactory - uses a ClassifierFactoryMap (passed in the constructor) to keep track of classifier-factory associations created through the GUI

C. Association Elements (abstract class)

C.i. Nouns (abstract class)

Methods to implement:
void phobAdded() - action that occurs when this AssociationNoun's phob gets added
void phobUpdated() - action that occurs when this AssociationNoun's phob gets updated
void phobRemoved() - action that occurs when this AssociationNoun's phob gets removed

Instantiable subclasses:
WebPage - on creation, a dialog prompts for a URL and a window pops up with that page; on phobAdded a window pops up with that page
(*MediaClip*) AudioClip - two constructors allow creation either by getting a file from the file system or starting a recording; on phobAdded, the clip gets played and on phobRemoved, the clip gets stopped
(*MediaClip*) VideoClip - on creation a file dialog prompts for a video file; on phobAdded, the clip gets played and on phobRemoved, the clip gets stopped

C.ii. Actions (abstract class)

Methods to implement:
void phobAdded(AssociationNoun focus) - action that occurs when this AssociationAction's phob gets added with the given focus
void phobRemoved() - action that occurs when this AssociationAction's phob gets removed

Instantiable subclasses:
FastForwardElt - on phobAdded, the focus's fastforward method gets called; on phobRemoved, the focus's resume method gets called
PauseElt - on phobAdded, the focus's pause method gets called; on phobRemoved, the focus's resume method gets called
ResetElt - on phobAdded, the focus's resetToBeginning method gets called
RewindElt - on phobAdded, the focus's rewind method gets called; on phobRemoved, the focus's resume method gets called

D. Phob Events

Given Methods:
Object getSource() - returns the PhobGenerator that generated the phob (note, the Phob also knows its source)
Phob getPhob() - returns the Phob associated with this event

Instantiable subclasses:
VisionPhob and TagPhob (for RFID and barcodes)

E. Phobs (abstract class)

Given Methods:
PhobGenerator getSource() - all Phobs know about the source from which they were generated

Instantiable subclasses:
VisionPhob and TagPhob (for RFID and barcodes)

E.i. VisionPhob

Ascertainable fields:
getBounds() returns the bounds (Rectangle)
getCenter() returns the center (Point2D)
getMajorAxisLength() returns major axis length (double)
getMajorAxisLength() returns minor axis length (double)
getMeanColorRGB() returns mean color (Point2D)
getTheta() returns theta (double)

E.ii. TagPhob

Given Methods:
getTagID() returns tag ID (String)

F. Phob Generators (abstract class)

Given Methods:
void addPhobListener(PhobListener listener) - adds the given listener to this generator
void removePhobListener(PhobListener listener) - removes the given listener to this generator

Instantiable subclasses:
VisionPhobGenerator, BarcodePhobGenerator and RFIDReader

F.i. VisionPhobGenerator

Constructor takes in an ImageSource and an optional minimum object size

F.ii. BarcodePhobGenerator

Constructor takes in a VisionPhobGenerator

F.iii. RFIDReader

Constructor takes in a unique reader ID (int)

Given Methods:
getReaderID() returns int

G. Device Managers (abstract class)

Given Methods:
void addDeviceListener(DeviceListener deviceListener) - adds the given device listener to this generator

void removeDeviceListener(DeviceListener deviceListener) - removes the given device listener to this generator

G.i. RFIDManager

Given Methods:
void addPhobListenerToAllReaders(PhobListener phobListener) - adds the given phob listener to all readers