CHAPTER 1 INTRODUCTION

A decade and half after Weiser's call to integrate computation into the fabric of our lives [248], the design and evaluation of ubiquitous computing systems remains challenging. Difficulties arise partially because of a lack of appropriate design tools: the progress of any creative discipline changes significantly with the quality of the tools available [51,167,228,229]. As the creation of ubiquitous computing devices moves from research labs into design consultancies and product teams, better tools that support user experience professionals are needed. My fieldwork at professional design companies showed that design generalists currently lack the tools to fluently experiment with interactions for sensor-based interfaces and information appliances.

Prototyping is the pivotal activity that structures innovation, collaboration, and creativity in professional design. Design studios pride themselves on their prototype-driven culture; it is through the creation of prototypes that designers learn about the problem they are trying to solve. Effective prototyping tools aid and improve design space exploration, design team communication, and ultimately, lead to better products. The goal of this dissertation is to develop principles and authoring methods that guide the creation of more appropriate prototyping tools for interaction design. This dissertation contributes to the advancement of prototyping tools by considering two different research questions:

1) How can design tools enable a wider range of designers to create functional prototypes of ubiquitous computing user interfaces?
2) How can design tools support the larger process of learning from these prototypes?

Our exploration of the first question is concerned with improving prototyping methods—finding new ways to model and structure the authoring task. Two complementary methods for authoring sensor-based interactions are introduced: a control-flow-based visual authoring environment for interaction logic that is based on existing storyboard practices; and a programming-by-demonstration environment that helps designers extract useful high-level interaction events from continuous sensor data. These methods lower the expertise threshold required to author sensor-based interfaces. The tools enable more designers to author a wider range of interfaces, faster.

The second question concerns the function a prototype plays in the larger design process. Prototyping is not primarily about the artifacts that get built: it is about eliciting feedback (from the situation, from users, team members & clients). Prototypes embody design hypotheses and enable designers to test these hypotheses. Prototyping tools can thus become more valuable to designers when they explicitly offer support for eliciting and managing feedback. Today, software prototyping tools are mostly agnostic to this role. This dissertation contributes techniques for creating and managing multiple alternative design solutions, and for managing feedback from both external testers and design team members. A brief overview of the contributions and the contents of this dissertation follows.

1.1 THESIS CONTRIBUTIONS

This dissertation contributes principles and systems for prototyping user interfaces that span physical and digital interactions. The technical contributions are based on evidence collected through interviews with designers and online surveys. This evidence suggests that interaction designers lack tools to create interfaces that leverage sensor input; to explore alternative interface behaviors; to efficiently review interface test videos; and to effectively communicate interface revisions.

The dissertation makes the following technical contributions in three areas:

1) Techniques for authoring user interfaces with non-traditional input/output configurations.
   a. Rapid authoring of interaction logic through a *novel combination of storyboard diagrams* for information architecture *with procedural programming* for interactive behaviors.
   b. *Demonstration-based definition of discrete input events from continuous sensor data streams* enabled by a combination of pattern recognition with a direct manipulation interface for the generalization criteria of the recognition algorithms.
   c. *Management of input/output component configurations for interface prototypes* through an editable virtual representation of the physical device being built. This representation reduces cognitive friction by collapsing levels of abstraction; it is enabled by a custom hardware interface with a plug-and-play component architecture.

2) Principles and techniques for exploring multiple user interface alternatives.
    a. Techniques for *efficiently defining and managing multiple alternatives of user interfaces* in procedural source code and visual control flow diagrams.
    b. *User-directed generation of control interfaces* to modify relevant variables of user interfaces at runtime.
    c. *Support for sequential and parallel comparison of user interface alternatives* through parallel execution, selectively parallel user input, and management of parameter configurations across executions.
    d. *Implementations of the runtime techniques for three different platforms*: desktop PCs, mobile phones, and microcontrollers.
3) Techniques for capturing feedback from users and design team members on user interface prototypes, and for integrating that feedback into the design environment.
    a. *Timestamp correlation between live video, software states, and input events* generated during a usability test of a prototype to enable rapid semantic access of that video during later analysis.
    b. *Novel query techniques* to access such video recordings: *query by state selection* where users access video segments by selecting states in a visual storyboard; and *query by input demonstration* where sections of usability video are retrieved through demonstrating, on a physical device prototype, the kind of input that should occur in the video.
    c. *A visual notation and a stylus-controlled gestural command set for revising user interfaces* expressed as control flow diagrams.

This dissertation also provides evidence, through laboratory studies and class deployments, that the introduced techniques are successful. In particular, the dissertation contributes:

1) Evidence that the introduced authoring methods for sensor-based interaction are accessible and expressive through two laboratory evaluations and two class deployments.
2) Evidence from a laboratory study that the techniques for managing interface alternatives enable designers to explore a wider range of design options, faster.
3) Evidence from two laboratory studies that an interactive revision notation for interfaces leads to more concrete and actionable revisions.

1.2 DISSERTATION ROADMAP

This section presents a brief overview of the structure of this dissertation by chapters.

1.2.1 BACKGROUND: PROTOTYPES IN THE DESIGN PROCESS (CHAPTER 2)

The terms design, prototyping, and sketching have many meanings for different audiences. Drawing on literature in design research, this chapter stakes out a perspective on design practice and how prototyping activities occur throughout the design process. We briefly discuss the history of industrial design as a discipline grounded in the development of material goods and the later transfer of principles from industrial design to software [251].

While there are many different conceptions of the design process, models tend to agree on two core characteristics. First, design is exploratory and emergent — the structure of the design problem itself has to be uncovered and this uncovering happens through generating concrete design proposals and evaluating them. Designers think with, and communicate through artifacts and models [66]. These artifacts are prototypes. Second, generation and evaluation of solution proposals exemplifies a recurrent, fundamental interplay between divergent and convergent stages in design: first a range of different potential solutions is generated, then desirable solutions are selected from that set of alternatives. This cycle repeats as the focus shifts from design concepts to implementation strategies.

The chapter concludes with a survey of literature about the role that prototypes play in design and software engineering. This survey leads to a classification of prototypes according to three questions: What purpose do prototypes serve? What aspects of a design do they address? And what level of functionality should they offer?

1.2.2 RELATED WORK (CHAPTER 3)

How are existing tools supporting prototyping activity? What needs are still unmet? We describe the state of the art in professional practice and present an overview of research in prototyping tools. Our discussion of related research addresses the following concerns in separate sections:

USER INTERFACE PROTOTYPING TOOLS

Prior research has introduced environments for graphical user interfaces [155], web site information architecture design [171], and context-aware applications [219], among others. We review tools that focus on storyboard-based authoring, direct manipulation UI layout, and Wizard of Oz simulation [140] of interface functionality.

TOOLS SUPPORT FOR PHYSICAL COMPUTING

Physical computing combines physical and digital interactions. Consequently, tools in this area often focus on the interdependence of hardware and software. We review research into hardware toolkits and programming models for working with sensors and actuators [93,178,185].

VISUAL AUTHORING

Visual authoring or visual programming is thought to offer a lower threshold than textual programming. The reality is more nuanced. We provide an overview of visual formalisms and visual programming languages. We distinguish between control flow environments [87], data flow environments [209], augmented source editors [238], and hybrid environments [1].

PROGRAMMING BY DEMONSTRATION

Programming by demonstration promises to lower the barrier of specifying complex logic or behavior by demonstrating that behavior to a computer. The crucial step in the success of failure of programming by demonstration lies in the generalization step that transforms observed examples to general rules. We discuss how previous systems have addressed this challenge for programming [67] and computer vision applications [75].

DESIGNING MULTIPLES

If design indeed oscillates between generating multiple alternatives and then selecting between these alternatives, design tools should explicitly support working with sets of potential designs. We survey existing work in image processing [161,239], rendering [181] and information querying [177] that address this challenge.

CAPTURING & MANAGING FEEDBACK

How can design tools capture user test data or team feedback on prototypes? We review work in document annotation [198] and usability video structuring through event logs [39,179].

1.2.3 AUTHORING SENSOR-BASED INTERACTIONS (CHAPTER 4)

This chapter presents two novel prototyping methods that enable faster creation of functional interaction designs for sensor-based user interfaces. Myers et al. introduced the terms threshold and ceiling to describe use properties of a tool: the *threshold* is the difficulty of learning and using a system, while the *ceiling* captures the complexity of what can be built using the system [191]. d.tools and Exemplar help designers construct functional prototypes

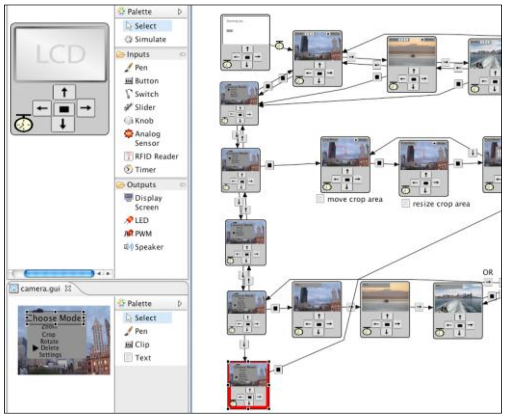


Figure 1.1: The d.tools visual authoring environment enables rapid construction of UI logic.

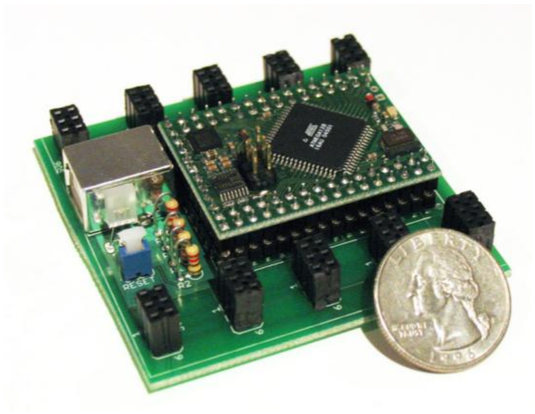


Figure 1.2: The d.tools hardware interface offers a plug-and-play architecture for interface components.

by lowering the threshold of required expertise. The goal of both systems is to enable users to focus on design thinking (how an interaction should work) rather than implementation tinkering (how hardware and sensor signal processing work).

d.tools is a software and hardware toolkit that embodies an iterative-design-centered approach to prototyping information appliances. d.tools enables non-programmers to work with the bits and the atoms of physical user interfaces in concert. Supporting early-stage prototyping through a visual, statechart-based approach, d.tools extends designers' existing storyboarding practices (Figure 1.1). As designers move from early-stage prototypes to higher fidelity prototypes, d.tools augments visual authoring with scripting. A hardware platform based on smart components that communicate on a shared bus offers plug-and-play use of sensors and actuators (Figure 1.2). The architecture exposes extension points for experts to grow the library of supported electronic components.

d.tools provides software abstractions for hardware and offers rapid authoring of interaction logic. An additional barrier for practitioners became apparent when we deployed d.tools to an HCI class: students often struggled to transform raw, noisy sensor data into useful high-level events for interaction design. Exemplar, an extension to d.tools, bridges the conceptual gap between conceiving of a sensor-based interaction and formally specifying that interaction through programming-by-demonstration. With Exemplar, a designer first demonstrates a sensor-based interaction to the system (e.g., she shakes an accelerometer — Figure 1.3). The system graphically displays the resulting sensor signals. The designer then marks up the part of the visualization that corresponds to the action — Exemplar learns appropriate recognizers from these markups. The designer can review the learned actions through real-time visual feedback and modify recognition parameters through direct manipulation of the visualization.

Both d.tools and Exemplar have been evaluated through individual laboratory studies and deployment to interaction design courses and to industry. In a first-use evaluation of Exemplar, participants with little or no prior experience with sensing systems were able to design new motion-based controllers for games in less than 30 minutes (Figure 1.4). In our collaboration with educational toy company Leapfrog, we provided d.tools hardware schematics and software to Leapfrog's advanced development group. In return, Leapfrog manufactured a complete set of hardware toolkits for us to distribute to a second year of Stanford HCI students. In collaboration with Nokia, we also extended d.tools to author prototype interfaces for mobile devices.

1.2.4 CREATING ALTERNATIVE DESIGN SOLUTIONS (CHAPTER 5)

Creating multiple prototypes facilitates comparative reasoning, grounds team discussion, and enables situated exploration. However, current interface design tools focus on creating single artifacts. How might interaction design tools explicitly support creation and management of multiple user interface alternatives? This chapter discusses two approaches.

We first investigated how to support exploration in Juxtapose, a source code editor and runtime environment for designing multiple alternatives of interaction designs in parallel. Juxtapose offers a code editor for user interfaces authored in ActionScript in which

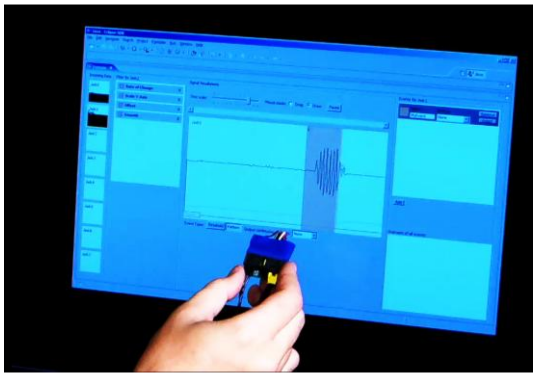


Figure 1.3: Exemplar combines programming-by-demonstration with direct manipulation to author sensor-based interactions.

Figure 1.4: This evaluation participant used Exemplar to control 2D aiming in a game with an accelerometer, and shooting with the flick of a bend sensor.

interaction designers can define multiple program alternatives through linked editing, a technique to selectively modify source files simultaneously. The set of source alternatives are then compiled into a set of programs that are executed in parallel (Figure 1.5). Optimizing user experience often requires trial-and-error search in the parameter space of application variables. To improve this tuning practice, Juxtapose generates a control interface for application parameters through source code analysis and language reflection (Figure 1.6). A summative study of Juxtapose with 18 participants demonstrated that parallel editing and execution are accessible to interaction designers and that designers can leverage these techniques to survey more options, faster. To show that general principles of working with alternatives carry over into other domains, we also developed Juxtapose runtime environments for mobile phones and microcontrollers.

We then discuss how ideas for exploring alternatives can be transferred from a textual programming environment such as Juxtapose to the visual authoring environment of d.tools. Visual control flow environments offer the opportunity to present alternative states side-by-side in the same canvas. They also present some challenges in managing the additional visual complexity resulting from capturing multiple behavior options.

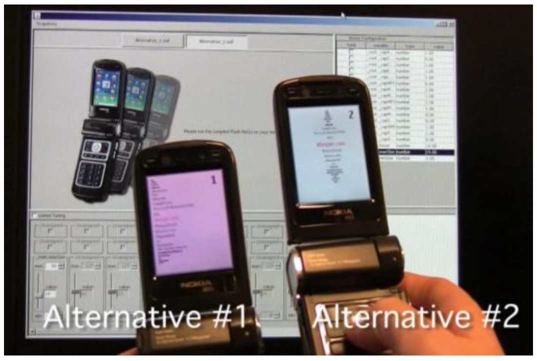
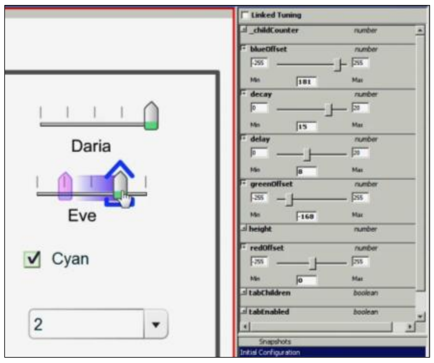


Figure 1.5: Side-by-side execution in Juxtapose enables rapid comparison of alternatives.



Linked Tuning
_childCounter number
blueOffset number -255 255 Min 181 Max
decay number 0 20 Min 15 Max
delay number 0 20 Min 8 Max
greenOffset number -255 255 Min -168 Max
height number
redOffset number -255 255 Min 0 Max
tabChildren boolean
tabEnabled boolean
Snapshots
Initial Configuration

Figure 1.6: Juxtapose automatically generates control interfaces for program variables.

1.2.5 GAINING INSIGHT THROUGH FEEDBACK (CHAPTER 6)

If prototyping is about eliciting feedback, then tools that manage the feedback process explicitly can help designers gain insight, capture that insight, and act on it. We present two methods for integrating feedback capture and management directly into design tools.

Many prototypes go through team discussions and reviews before being tested. In word processing, revision management algorithms and interactions techniques effectively enable asynchronous collaboration over text documents. But no equivalent functionality exists yet for revising interaction designs. d.note introduces a revision notation for expressing tentative design proposals within d.tools. The tool comprises commands for insertion, deletion, modification and commenting on appearance and behavior of interface prototypes (Figure 1.7). d.note realizes three benefits: it visually distinguishes tentative changes to retain design history, allows for Wizard of Oz simulation of proposed functionality, and manages display of alternative design choices to facilitate comparison. In a laboratory evaluation, twelve design students critiqued existing d.tools prototypes with and without d.note. Participants reported that the ability to express and test functional changes was a clear benefit of d.note. In a follow-up study, eight design students interpreted the annotated diagrams produced in the first study, showing that d.note diagrams were less ambiguous to interpret, but that they lacked high-level justification when compared to free-form annotation.

When prototypes are tested with team mates or external users, test sessions are often recorded on video. Historically, the hours and days of work required for manual video analysis has limited the practical value of these recordings. The d.tools video suite provides integrated

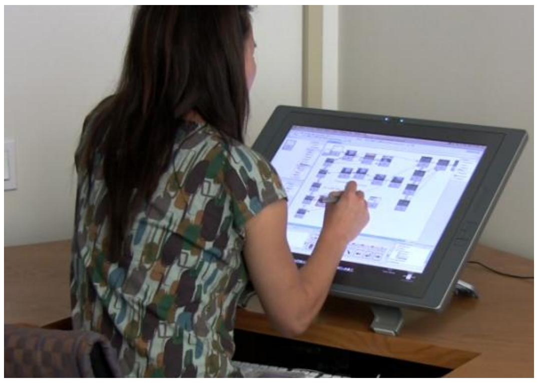


Figure 1.7: d.note introduces stylus-driven revision of interaction diagrams.

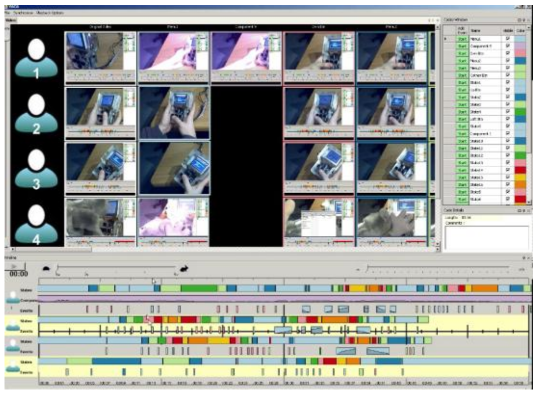


Figure 1.8: The d.tools test & analysis functions link video clips of test sessions to event traces of the tested prototype.

		d.tools	d.note	Exemplar	Juxtapose
HOW	Visual Authoring Environment	✓	✓	✓	
(Techniques)	Programming by Demonstration			✓	
	Parallel Editing & Execution				✓
WHY	Rapid Construction & Iteration	✓		✓	
(Place in design)	User Feedback: Design-Test-Analyze	✓			
	Team Feedback: Design-Review-Annotate		✓		
	Exploring Design Alternatives	✓	✓		✓
WHAT	Custom, sensor-based interactions	✓	✓	✓	✓
(Artifacts built)	Mobile device interfaces	✓			✓
	Desktop GUIs				✓
	Tentative Sketches; Simulated Functionality		✓		

Table 1.1: An overview how research concerns map onto the concrete systems presented in this dissertation.

support for testing prototypes with users and rapidly analyzing test videos to inform subsequent iteration (Figure 1.8). d.tools logs all user interactions with a prototype and records an event-synchronized video stream of the user's interactions. The video is automatically structured through state transitions and input events. After a test, d.tools enables designers to view video and storyboard in parallel as a multiple view interface [41] into the test data. Two novel query techniques shorten the time to find relevant segments in the video recordings: query by state selection, where users access video segments by selecting states in the storyboard; and query by input demonstration, where designers demonstrate the kind of input that should occur in the video.

1.2.6 CONCLUSIONS & FUTURE WORK (CHAPTER 7)

The final chapter provides a review of the contributions, and offers an outlook to future work by reconsidering the fundamental assumptions made in this dissertation. The chapter discusses opportunities for different types of authoring tools that result if some of these assumptions are modified.

1.2.7 OVERVIEW: RESEARCH CONCERNS & PROJECTS

This dissertation explores the space of novel prototyping tools through multiple projects. To aid the reader, Table 1.1 shows how research concerns map onto the different concrete projects discussed in this dissertation.

1.3 STATEMENT ON MULTIPLE AUTHORSHIP AND PRIOR PUBLICATIONS

The research presented in this dissertation was not undertaken by me alone. While I initiated and led all of the projects described here, the contributions of a talented group of collaborators must be acknowledged — without their efforts, the research could not have been realized in its current scope. In particular, the d.tools project benefited from UI implementation contributions by Michael Bernstein, Leith Abdulla, and Jennifer Gee; and video editor programming and integration by Brandon Burr and Avi Robinson-Mosher. In the Exemplar project, Manas Mittal contributed to the signal processing routines; Leith Abdulla contributed to the Exemplar user interface implementation. Sean Follmer, Timothy Cardenas, and Anthony Ricciardi contributed to gesture recognition, revision management, and the graphical user interface editor in d.tools and d.note. Meredith Ringel Morris, Sean Follmer, Haiyan Zhang, and Jesse Cirimele collaborated on various demonstration applications for d.tools and Exemplar. Loren Yu worked with me on the source code editor and runtime user interface of Juxtapose; Abel Allison and Yeonsoo Yang helped with the implementation of Juxtapose functionality for microcontrollers.

This dissertation is partially based on papers published in previous ACM conference proceedings; I am primary author on all publications. In particular, the d.tools system was published at UIST 2006 [109]; Exemplar at CHI 2007 [107]; and Juxtapose at UIST 2008 [114]. A paper describing d.mix is still in submission at the time of publication of this dissertation.