Rafael Ballagas

Bringing Iterative Design to Ubiquitous Computing

Cuvillier Verlag Göttingen

media computing group

RWTH AACHEN UNIVERSITY

Bringing Iterative Design to Ubiquitous Computing:

Interaction Techniques, Toolkits, and Evaluation Methods

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Rafael A. Ballagas, M. Sc.
aus Atlanta, Georgia, USA

Berichter: Prof. Dr. Jan Borchers
Prof. Dr. Hans Werner Gellersen

Tag der mündlichen Prüfung: 23. August, 2007

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.
1. Aufl. - Göttingen : Cuvillier, 2008
Zugl.: (TH) Aachen, Univ., Diss., 2007

978-3-86727-531-6

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage, 2008
Gedruckt auf säurefreiem Papier

978-3-86727-531-6

Contents

List of Figures

List of Tables

Abstract

An iterative human-centered design process is required to create interfaces that are useful, intuitive, efficient, and enjoyable for users in the ubiquitous computing domain. Currently, only experts can design, prototype, and deploy ubiquitous computing applications; others lack the tools and conceptual frameworks. This work starts to fill the gap by providing contributions that support each phase of the iterative human-centered design process and address the complexity of ubiquitous computing application scenarios.

- To support the design phase, the range of ubiquitous mobile input techniques are organized into a design space, which helps identify the relationships between input techniques, and select the most appropriate input technique for an interaction scenario.
- To support the prototyping phase, the iStuff Toolkit architecture simplifies construction of functional prototypes for ubiquitous computing application scenarios. The architecture has been used to create two separate toolkits: iStuff to simplify prototyping physical user interfaces for ubiquitous computing, and iStuff Mobile to simplify prototyping new sensor-based interactions for mobile phones in ubiquitous computing.
- To support the evaluation phase, a new conceptual framework based on expressiveness is used to demonstrate how to evaluate input devices in prototype form (suffering from reduced resolution or sampling rates) and still make conclusions about future performance if further time and money were invested in improvements.

To illustrate how this iterative design process can be used from drawing board to deployment, experiences developing REXplorer are shared. REXplorer is one of the first permanently installed pervasive games and helps tourists explore the historical UNESCO World Heritage city of Regensburg, Germany. Players use a special "paranormal activity detector" (a device composed of a mobile phone and a GPS receiver) to interact with location-based and site-specific spirits. "Casting a spell" by waving the wand-like detector lets players awaken and communicate with the spirits to receive and solve quests. The game is designed to make learning history fun and influence tourists' path through the city.

Zusammenfassung

Um nützliche, intuitive, effiziente und unterhaltsame Schnittstellen für die Benutzer im ubiquitären IT (ubiquitous computing) zu entwickeln, ist ein iterativer und benutzerorienter Designprozess erforderlich. Zurzeit sind es nur Experten, die unter diesen Voraussetzungen Prototypen entwickeln können. Nicht-Experten fehlt es an den notwendigen Werkzeugen und den konzeptuellen Bezugssystemen. Die vorliegende Arbeit verringert diese bestehende Lücke, wobei sie einen Beitrag leistet, der jede Phase des iterativen und benutzerorientierten Designprozesses unterstützt und die Realisierung solcher ubiquitären IT-Szenarien vereinfacht.

- Zur Unterstützung der Designphase sind die gegenwärtigen mobilen Eingabetechniken in einem Gestaltungsraum (design space) organisiert. Dieser Gestaltungsraum hilft, die Beziehung zwischen den Eingabetechniken zu identifizieren und die passende für das jeweilige Interaktionsszenario zu wählen.

- Zur Unterstützung der Prototypenphase vereinfacht die iStuff Architektur mit ihrer „Analogie eines Werkzeugkastens“ die Konstruktion der Funktionsprototypen für das ubiquitäre IT. Basierend auf dieser Architektur wurden zwei separate „Werkzeugsätze“ entwickelt: iStuff vereinfacht die Herstellung von Prototypen für physikalisch basierte Benutzeroberflächen während iStuff Mobile die Herstellung von Prototypen für neuen sensorbasierte Interaktionen mit Mobiltelefonen unterstützt.

- Zur Unterstützung der Auswertungsphase wird ein neues Konzept angewendet, welches auf Ausdrucksfähigkeit basiert. Es ist für die Auswertungen von prototypischen Eingabeprototypgeräten (mit niedriger Auflösung und geringen Abtastrate) geeignet und zeigt anhand der erreichten Leistungen, ob zukünftige Zeit- und Geldinvestitionen erforderlich sind.

Um die Anwendung dieses iterativen Designprozesses von Zeichenbrett bis zum richtigen Einsatz darzustellen, werde ich die von mir gesammelten Erfahrungen während der Entwicklung von REXplorer, eines der ersten fest installierten pervasive Spiels, behandeln. REXplorer hilft Touristen bei der Erforschung der von der UNESCO zum Weltkulturerbe erklärten historischen Stadt Regensburg in Deutschland. Die Spieler benutzen einen speziellen „paranormalen Aktivitätsdetektor“ (ein Gerät, bestehend aus einem mobilen Telefon und einen GPS-Empfänger), mit dem sie mit standortbezogenen und ortspezifischen fiktiven Mitspielern zusammenspielen. Durch sogenanntes „Verzaubern“ (winkende Bewegungen mit einem Zauberstab) erweckt der Spieler die fiktiven Mitspieler zum Leben, unterhält sich mit ihnen und bekommt Fragen gestellt, deren Lösungen er in der Stadt suchen muss. Das Spiel wurde entwickelt, um das Lernen von Geschichte unterhaltsam zu machen und um Touristen durch die Stadt zu führen.

Acknowledgements

First, I would like to thank my advisor, Jan Borchers, for his continued support of my work. He is ultimately the reason I came to Germany to work on my Doctoral degree at RWTH Aachen University, and I am glad that I followed him. He was there to meet with me to discuss ideas, and always willing to provide comments on my papers and chapters. He kept challenging me to think differently throughout our time together.

I would also like to thank my co-advisor, Hans Gellersen, who has supported me over the past years through collaborations with his research group. His influence and support have helped shape this work.

I would also like to acknowledge all of the contributors to the REXplorer project. REXplorer was jointly developed between RWTH Aachen University and ETH Zurich with over 50 individual contributors. Primarily, I would like to thank Steffen P. Walz, my counterpart at ETH Zurich who took the lead for content development and co-designed the game with me. Alexander Möhnle significantly contributed to character development and script creation for the game content, and he is the photographer for the picture on the cover of this work. Also, I would like to thank my students at RWTH Aachen University who helped realize the game implementation, especially Sven Kratz, Joel Mendoza, Eugen Yu, André Kuntze, and Johannes Fundalewicz. REXplorer would not be a product today without the help of these individuals and many others that are not listed here.

My colleagues have also played an important role in this process, especially Eric Lee, David Holman, Daniel Spelmezan, Thorsten Karrer and Elaine Huang. They were always available to provide comments, help refine ideas, and most importantly laugh with me to maintain my sanity. I only hope to have the same quality of people to work with throughout the remainder of my career.

I definitely would not have gotten this far without the love and companionship of Snezhana Dimitrova. She has helped me stay grounded and remember what is important in life. The ability to finish this thesis was largely due to her encouragement and support.

Finally, I'd like to dedicate this thesis to my parents, Rafael and Linda Ballagas, and my sister, Corina Ballagas. They have always encouraged me to do my best, and their love and support has been unwavering.

Conventions

The following conventions will be used throughout this thesis:

Technical terms or jargon that appear for the first time will be set in *italics*. Definitions of technical terms or other jargon will be enclosed in shaded boxes, and will be referenced in the index.

Jargon:
Special words or expressions that are used by a particular profession or group and are difficult for others to understand.

Definition:
Jargon

Source code and implementation symbols will be typeset using a `monospace font`.

Margin notes will be used to improve readability and assist in browsing the text.

Chapter 1

Introduction

"You don't have to please everyone–you have to please the user."

—Brenda Laurel

Mark Weiser envisioned ubiquitous computing as a world where computation and communication "*blend into the fabric of our everyday lives*" (Weiser, 1991). To realize Weiser's vision, we must find interfaces that are useful, intuitive, efficient, and enjoyable for users in the ubiquitous computing domain. An iterative human-centered design process (Nielsen, 1993) is required to find these interfaces. Currently, only experts can design, prototype, and deploy ubiquitous computing applications; others are lacking the tools and conceptual frameworks to fully support an iterative human-centered design process for ubiquitous computing. This work starts to fill the gap by providing contributions that support each phase of the iterative human-centered design process that addresses the complexity of ubiquitous computing application scenarios.

Ubicomp needs iterative human-centered design

1.1 Iterative Human-Centered Design

The field of Human–Computer Interaction (HCI) has long recognized that user interfaces should be designed iteratively (Nielsen, 1993; Buxton and Sniderman, 1980; Gould and Lewis, 1985), because the requirements for an interactive system cannot be completely specified at the beginning of the lifecycle (Dix *et al.*, 2004). Instead, the road to success in interaction design is to *fail early and often*. The design needs to be prototyped and tested with real users to reveal any false assumptions or unforeseen design problems. These problems can then be corrected in the next iteration of the prototype, which should then again be tested to ensure the problems are resolved. Each prototype is more detailed and functional than the last, thus converging towards the final system (see figure 1.1). The main phases of iterative design are:

the road to success in interaction design is to fail early and often

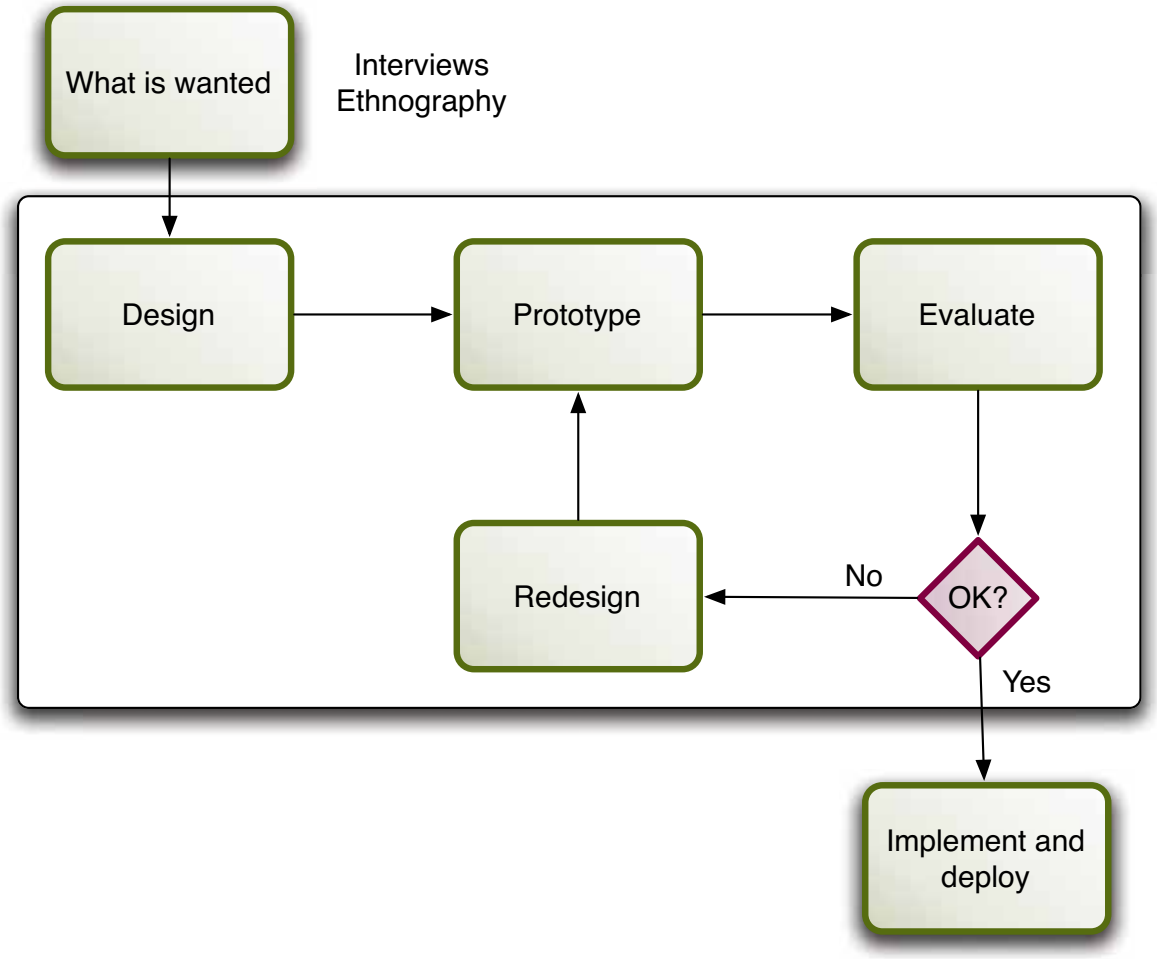


Figure 1.1: A high-level diagram illustrating the iterative design process (adapted from Dix *et al.* (2004))

know the user

- **Requirements (what is wanted)**– In human-centered design, the primary task is to 'know the user' (Hansen, 1971). When beginning to develop any interactive system, it is important to clearly identify who the system is intended to support, and for which tasks. After the user group is identified, exploratory techniques such as contextual inquiry or ethnography can be used to derive user needs and system requirements.

existing design knowledge can help conceptualize an interface

- **Design** – This is the stage of the design process where the system requirements are translated into a design solution. This stage can be informed through design knowledge captured by abstract design guidelines (Mayhew, 1991), platform specific design guidelines (Apple Computer Inc., 1992), heuristics (such as Shneiderman's golden rules (Shneiderman, 1992)), and HCI design patterns (Borchers, 2001). Other tools that can assist in making informed design decisions are design spaces, such as Card *et al.*'s design space of input devices (Card *et al.*, 1991), which helps designers reason about design alternatives and identify the most appropriate design for the given

task.

- **Prototyping** – Early in the development of a product, prototypes are typically conceptual in the form of scenarios, sketches, and storyboards that illustrate the basic usage in context. Later after evaluation, more detailed prototypes flush out concrete design ideas. With each cycle in the iterative design process, the ideas are further refined with a combination of functional (works like) and form (looks like) prototyping strategies.

Prototype refinement advances with each iteration

- **Evaluation** – The essence of iterative design is to evaluate the prototypes early and often to identify problems and design flaws early in the design process. Delaying meaningful testing increases the cost of correcting fundamental design problems. Analysis can either be done without users, such as employing an expert to perform a heuristic analysis, or they can be tested by observing real users interacting with the product either in controlled experiments or in real context of use. All of these forms of evaluation should be used together throughout the design process to identify potential difficulties users may have with a product. After the problems are identified, they can be translated into design changes for the next iteration of the prototype (Dix *et al.*, 2004).

Evaluate early and often with real users

- **Implementation and Deployment** – Once the design is of acceptable quality, the creation of production quality code, the manufacturing of robust and integrated hardware, and the creation of documentation and manuals can begin. Commonly, the output of the iterative design process is a full design specification and reference prototype, not the final implementation. Evaluations should continue throughout the implementation stage to ensure that the implementation meets the quality required by the design. If the quality cannot be met, further design iterations may be necessary.

Design before implementation

Prototyping structures innovation, collaboration, and creativity in the most successful design studios (Kelley and Littman, 2001). Designers use prototypes as physical representations of ideas, effectively externalizing cognition and facilitating a "conversation with materials" to uncover surprising problems or generate suggestions for new designs (Schön and Bennett, 1996). Prototypes also serve as artifacts that represent tacit knowledge of developers as a communication tool to clients or other members of a design team (Schrage, 1999). Most importantly, prototypes provide an artifact to test with real users as a part of a human-centered iterative design process.

Prototypes externalize cognition

Research has shown that, generally speaking, the more iterations in the design process, the better the user interface (Nielsen, 1993). Figure 1.2 illustrates how usability improves with each iteration in the design process. At the surface, it appears that each iteration is a time consuming and expensive process, but studies have shown that iterative design has economic value (Karat, 1990). Additionally, the cost of performing design iterations can be dramatically decreased with rapid prototyping strategies, but the

More iterations lead to better designs

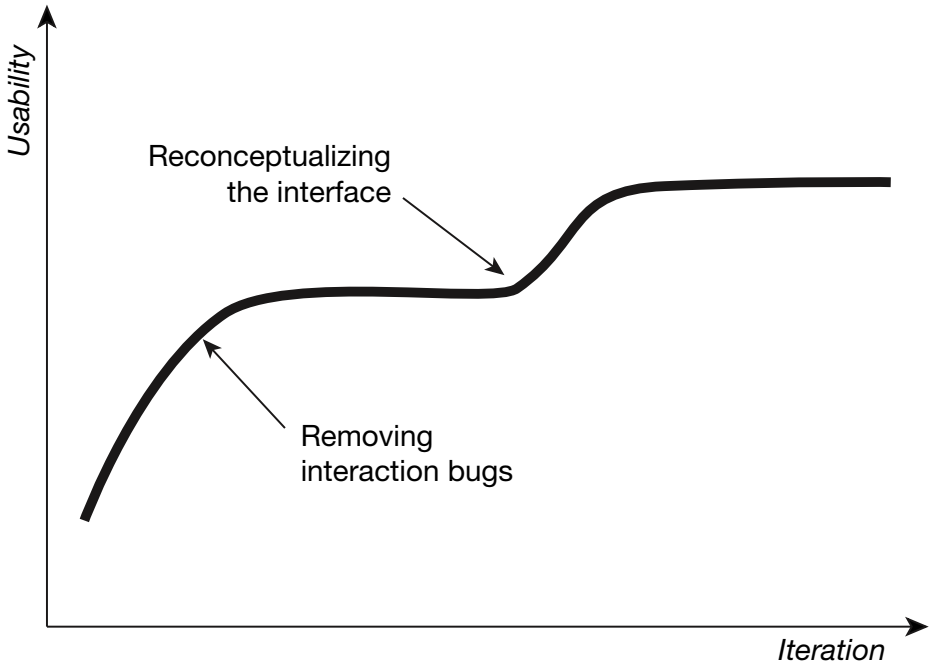


Figure 1.2: An illustration of interface quality as a function of the number of design iterations. Each additional iteration increases the usability of the design until a potential "usability plateau" is reached. (Nielsen, 1993)

nature of the prototype influences the nature of the problems that can be identified.

Prototype both form and function

Prototypes can generally be characterized as one of two variants: functional (*works like*) prototypes try to match the interactive experience as closely as possible (Buchenau and Suri, 2000), and form (*looks like*) prototypes are passive and try to match the appearance and affordances of the final design. These two characterizations can be seen as two extremes of a prototype continuum where, in most practical situations, prototypes have aspects of both. For example, humans can simulate the interactive functionality of form prototypes by updating the state of the form prototype manually. This practice, known as Wizard of Oz prototyping (Dahlbäck *et al.*, 1993), was originally developed in the context of natural language interfaces where a hidden human stenographer typed in spoken text to simulate high-performance natural language processing (Kelley, 1984). This technique allows the interactions to be tested before significant effort is placed in the implementation.

Beware of false starts

One of the pitfalls of iterative human-centered design is that if you pick a poor starting point, you may reach a peak in the usability of a particular design without reaching the desired usability goals. In this case, it may be necessary to throw the design away and start over. False starts are relatively painless early in the design process if low-fidelity prototyping techniques are used, but can be extremely expensive if determined late in the design process. In order to minimize the risk of false starts, a parallel design

strategy (Nielsen and Faber, 1996) can be used, where multiple designs can be explored independently early in the design process. As the designs mature, the best design becomes clear, or the strengths of the top designs can be merged to a unified design. Parallel design is more practical early in the design process when rapid prototyping techniques are used.

1.2 Applying Iterative Design to Ubicomp

Today, strategies for applying an iterative design process to desktop graphical user interfaces are generally well-defined. Figure 1.3 exemplifies how these strategies might be applied over the course of a design process for a desktop application. However, attempting to apply an identical design process to ubiquitous computing is problematic, often because ubiquitous computing application scenarios require a functional prototypes to convey the intended experience. Currently, functional prototypes for ubiquitous computing are costly, time-consuming, and require technical expertise to construct. For example, Heiner *et al.* (1999) report spending about one person-year developing a ubiquitous peripheral display. If meaningful testing is delayed until too late in the design process, monetary constraints and resource commitments prohibit fundamental design changes (Ulrich and Eppinger, 1995).

Ubicomp requires functional prototypes

1.2.1 Fieldwork

Fieldwork that has examined current design practices indicates that there are many issues obstructing iterative design in ubiquitous computing applications.

Hartmann *et al.* (2006) conducted fieldwork interviewing product designers. They found that most product designers have had exposure to programming, but few were proficient. Although access to programmers and engineers was available, there were not enough to complete large prototyping projects. This resulted in a perception that interdisciplinary teams slow the interaction design process and increase costs. Thus, prototypes that combined form and function were not built until late in the design process. These prototypes were typically expensive, one-off presentation tools instead of artifacts for human-centered reflective practice.

Functional prototypes are often delayed

Klemmer (2004) conducted structured interviews with tangible user interface developers. For these developers, dealing with physical input was the primary challenge requiring a high level of technical expertise and extensive development effort. One developer commented "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." Developers reported that often extensive system redesigns were required to perform straightforward changes to input technologies (e.g., exchanging a camera and barcode reader). Addi-

Effort to build functional prototypes is too high

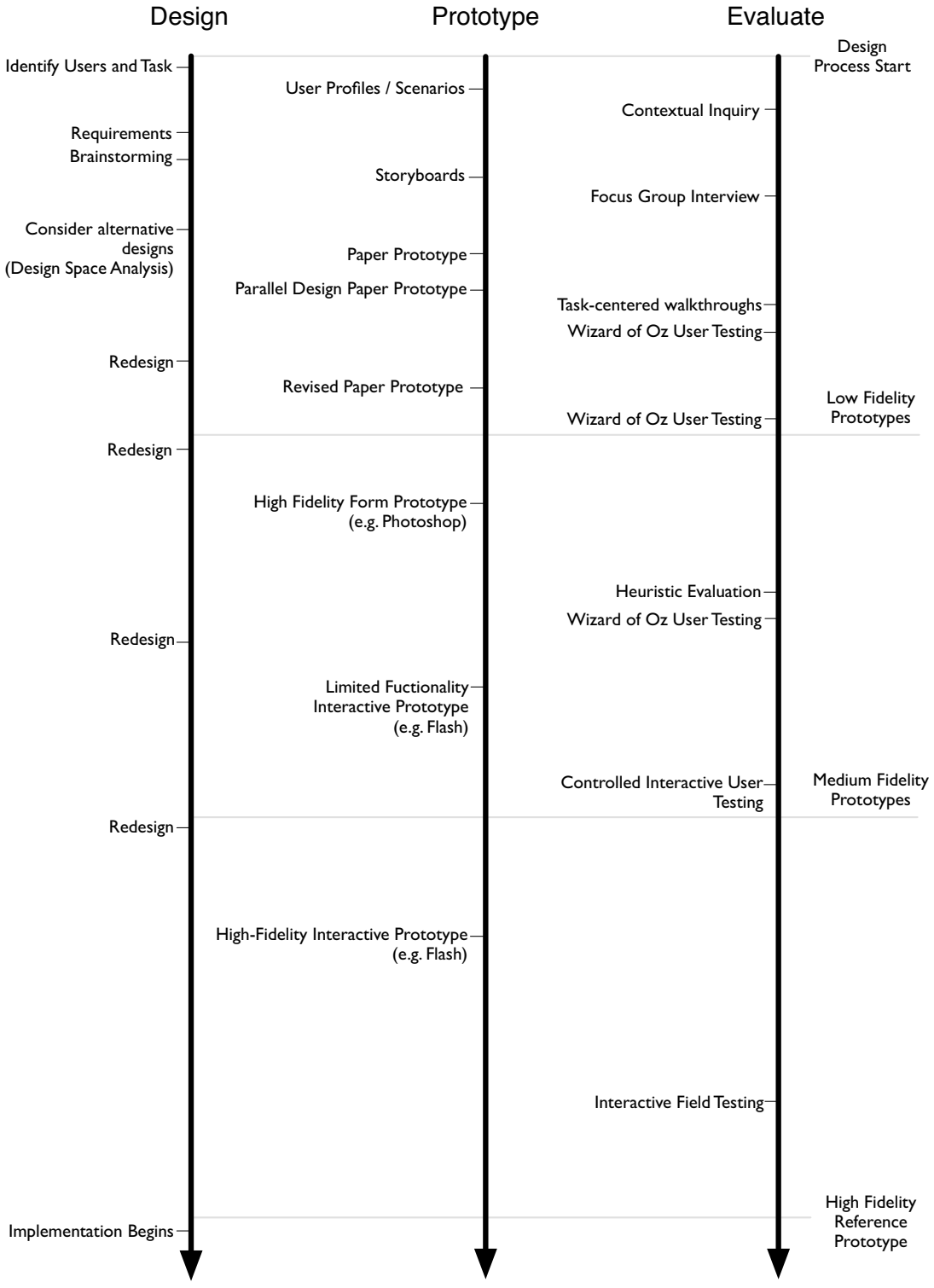


Figure 1.3: An abstract timeline illustrating a sample desktop iterative design process. Low-fidelity prototypes can be prototyped and evaluated much quicker than high fidelity prototypes. Identifying design flaws earlier in the iterative design process saves time and money.

tionally, each development team was creating their own software architectures based on basic event-based software design patterns from the ground up because no tool existed that could save developers time and effort.

Carter *et al.* (2007b) conducted fieldwork examining current design practices of applications for mobile devices (personal digital assistants or mobile phones). The biggest challenge in this domain was developing prototypes robust enough for use "in the wild". Similarly, Kjeldskov and Graham (2003) reviewed many mobile HCI projects and concluded that many mobile developers rarely used lightweight prototypes because they strongly believed it was important to test their tools in a realistic and ecologically valid setting. Developers found that lightweight prototypes were insufficient to perform these types of evaluations.

Functional prototypes are needed for ecologically valid evaluations

Matthews (2005) conducted fieldwork of peripheral display developers. One developer interviewed commented "I would say the hardest part about implementing these displays is the mechanics of doing it...". Participants were interested in building and deploying functional prototypes as rapidly as possible because the "real value in many of these systems is only apparent longitudinally." Developers interviewed also expressed a need for tools that support building applications that combined distributed input and output over multiple modalities (physical, graphical, or audio).

Functional prototypes are needed for longitudinal studies

1.2.2 Lightweight Prototypes

The most prevalent low-fidelity prototyping technique for graphical user interfaces is paper prototyping (Snyder, 2003) (see Figure 1.4). Paper prototypes are valuable because of the speed and low cost with which they can be constructed, evaluated, and thrown away or modified. Paper prototypes can be tested using a Wizard of Oz technique (Dahlbäck *et al.*, 1993), where the designer plays the role of the computer to update the paper "display" to respond to user input. Paper prototypes by themselves are low-fidelity form (*looks-like*) prototypes; when combined with Wizard of Oz, they are low-fidelity prototypes of the form and function of the proposed design. The unfinished nature and rough form of these early prototypes can be particularly valuable; end-users often see them as unfinished and provide richer design suggestions (Landay, 1996). A more polished prototype, on the other hand, implies effort and may discourage comments from testers that imply drastic design changes.

Unpolished prototypes can be valuable

This form of low-fidelity prototyping is well suited for the desktop paradigm as the constrained 2D nature of paper is a good match to the experience of using the standard 2D display of desktop environments. However, paper prototyping does not translate well to the ubiquitous computing domain, because it falls short of capturing the ubiquitous computing user experience convincingly (Liu and Khooshabeh, 2003). For example, Rudström *et al.* (2003) used paper prototypes to evaluate a mobile social application, but reported that users had difficulty reflecting upon how their use

Paper prototypes don't capture ubiquitous computing user experiences

Figure 1.4: Paper prototypes provide a low-fidelity representation of a graphical user interface to enable user testing early in the design process. (Photo by Kris Kables, reprinted under the Creative Commons License.)

would change if they were mobile and the system were interactive. In a different study, Mankoff and Schilit (1997) successfully applied paper prototypes to test a ubicomp terminal application that supported activities tied to a particular space such as requesting supplies, or making reservations. The prototype required wizards to respond once per day during evaluation. These contrasting examples serve to illustrate that paper prototypes can be well-suited when "interactivity" is limited, but are generally ineffective in ubiquitous computing – especially when real-time feedback is required.

Wizard of Oz prototyping techniques have proven successful for many ubiquitous computing interaction scenarios early in the design process. For example, many have successfully simulated sensor-based interactions using Wizard of Oz techniques (Mynatt *et al.*, 2001; Hudson *et al.*, 2003; Consolvo *et al.*, 2004). Other researchers have had success using Wizard of Oz for location detection (Li *et al.*, 2004; Benford *et al.*, 2004), gesture recognizers (Akers, 2006), speech interfaces (Klemmer *et al.*, 2000), multimodal interactions (Oviatt *et al.*, 2000), augmented reality (MacIntyre *et al.*, 2004), and input techniques (Klemmer *et al.*, 2004).

1.2.3 High-fidelity Prototypes

In the desktop domain, high-fidelity comprehensive prototypes that demonstrate both form and function of the user interface can be implemented

fairly quickly using development tools such as Adobe Flash[1] (Moggridge, 2006). In the ubicomp domain, support for functional prototyping is still emerging. Generally speaking, the only practical high-fidelity prototyping approach is to develop a working product that can be tested. Significant time, resources, and expertise are required to create these high-fidelity ubiquitous computing systems (Abowd, 1999).

1.3 Thesis Structure

This thesis is organized to emphasize contributions in each phase (design, prototyping, and evaluation) of the iterative human-centered design process.

Chapter 2 supports the design phase. It organizes the range of ubiquitous mobile input techniques into a design space. This design space is an important tool to help designers of ubiquitous computing applications identify the relationships between input techniques, and select the most appropriate input technique for their interaction scenarios. Included in this survey are two camera-based input techniques that I have developed: "Sweep" and "Point & Shoot".

Chapter 3 supports the prototyping phase. The iStuff toolkit architecture simplifies construction of functional prototypes for ubiquitous computing. This architecture has been used to create two separate toolkits: *iStuff* to simplify prototyping physical user interfaces for ubiquitous computing, and *iStuff Mobile* to simplify prototyping new sensor-based interactions for mobile phones in ubiquitous computing. The toolkit architecture is supported by the *Patch Panel* infrastructure that uses intermediation to deal with heterogeneity and allow for incremental integration.

Chapter 4 supports the evaluation phase. A new technique has been developed to evaluate prototype input devices. These devices may have a reduced sampling rate or resolution due to their technical immaturity, making it difficult to predict their efficiency if time and money were spent on improving the technology. The notion of device expressiveness is introduced to structure the evaluation such that conclusions can be made about future improvements of the input device.

Chapter 5 illustrates how an iterative design process can be used from drawing board to deployment, by discussing the experiences developing REXplorer. REXplorer is a permanently installed pervasive game; it helps tourists explore the history of Regensburg, Germany. In the game, historically-based spirits are stationed at points of interest throughout the city. Players use a special "paranormal activity detector" (a device composed of a mobile phone and a GPS receiver in a protective shell) to interact with location-based and site-specific spirits. A novel mobile interaction

[1] http://www.adobe.com/products/flash/

mechanism of "casting a spell" (making specific gestures by waving the wand-like detector through the air) allows players to awaken and communicate with spirits to receive and resolve quests. The game is designed to make learning history fun for tourists and influence their path through the city.

1.4 Thesis Contributions

This thesis provides contributions that support each phase of the iterative human-centered design process for ubiquitous computing.

Design spaces help reason about design alternatives

1. Organizes mobile phone input techniques into a design space that:
   (a) Organizes mobile phone input techniques into families to help reason about their relationships,
   (b) Aids designers in considering alternative parallel designs and selecting the most appropriate mobile phone input technique for a particular interaction scenario,
   (c) Allows future mobile phone input techniques to be predicted.

Mobile phones are ubiquitous input devices

2. New interaction techniques for employing mobile phones as input devices in ubiquitous computing application scenarios.
   (a) The *Sweep* and *Point & Shoot* input techniques use the camera on the mobile phone as a sensor to enable interactions with large public displays.
   (b) The Elope project shows how to fluidly combine the storage and processing of mobile phones with the input and output capabilities of an interactive workspace.
   (c) The REXplorer pervasive and mobile game for tourists employs mobile phones as a platform where users can interact with spirits (historical characters) distributed throughout an urban environment by casting spells (gestures created from waving the mobile phone through the air).

iStuff architecture radically simplifies construction of functional prototypes

3. Architecture support for physical user interface input and mobile phone input techniques in ubiquitous computing application scenarios that:
   (a) Lowers the threshold for prototyping ubiquitous computing applications that employ physical or mobile phone input,
   (b) Supports incremental integration, extensibility, and rapid configuration of input using the Patch Panel infrastructure,
   (c) Introduces several new programming interfaces for rapid prototyping ubiquitous computing interactions including a light scripting language and a visual programming environment.

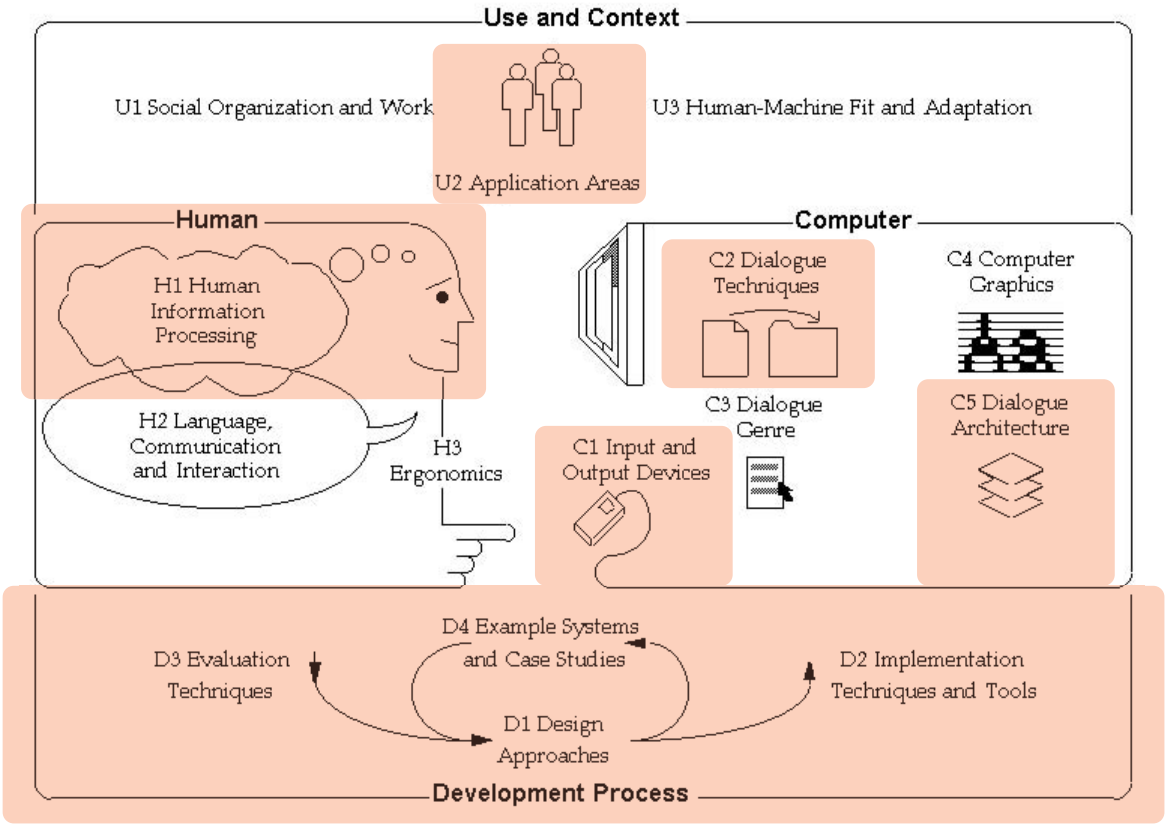


Figure 1.5: This thesis makes contributions across the field of Human–Computer Interaction in the highlighted areas. (Hewett *et al.*, 1992)

4. Redefines the expressiveness of input devices so that it can be calculated using physical properties of the device instead of empirical thresholds.

   Expressiveness helps evaluate input devices

   (a) Demonstrates how to use expressiveness to structure the evaluation of prototype pointing devices to be able to make conclusions about how the device will function after further refinement.

5. Demonstrates how to apply player-centered iterative design to pervasive game development.

   Iterative design from concept to deployment for ubicomp

   (a) Illustrates a range of low-fidelity, and limited functionality prototypes that can be used to evaluate parts of the game earlier in the design process.

Many of these contributions relate directly to the field of Human–Computer Interaction. As a preview, the corresponding areas are highlighted in the ACM's map of the field (see Figure 1.5).

Chapter 2

Supporting Design: The Design Space of Ubiquitous Mobile Phone Input Techniques

"Basically, an input device is a transducer from the physical properties of the world into logical parameters of an application."

—R. Baecker and W. Buxton

Today, mobile phones are used not just to keep in touch with others but also to manage everyday tasks, to share files, and to create personal content. Consequently, our mobile phones are always at hand. Technological trends result in ever more features packed into this small, convenient form factor. Smart phones can already see, hear, and sense their environment. But, as Weiser (1991) pointed out: "Prototype tabs, pads and boards are just the beginning of ubiquitous computing. The real power of the concept comes not from any one of these devices; it emerges from the interaction of all of them." Therefore, this chapter demonstrates how modern mobile phones (Weiser's tabs) can interact with their environment – especially large situated displays (Weiser's boards).

Broad I/O capabilities are an opportunity

The range of input and output (I/O) capabilities for modern mobile phones is broad. Keypad, joystick, microphone, display, touch-screen, loudspeaker, short range wireless connectivity over Bluetooth, WiFi, and infrared, long range wireless connectivity via GSM/GPRS and UMTS all provide multiple ways of interacting with our phones. These multiple I/O capabilities have increased our ability to use mobile phones to control resources available in our environment, such as public displays, vending machines, and home appliances.

Figure 2.1: A large public display used for advertisments and announcements in a subway stop in Vienna, Austria.

Potential default ubiquitous interface

The ubiquity of mobile phones gives them great potential to be the default physical interface for ubiquitous computing applications. This would provide the foundation for new interaction paradigms, similar to the way the mouse and keyboard on desktop systems enabled the WIMP (windows, icons, menus, pointers) paradigm of the graphical user interface to emerge. However, before this potential is realized, we must find mobile phone interaction techniques that are intuitive, efficient, and enjoyable for applications in the ubiquitous computing domain.

2.1 Examining the Design Space of Input Devices

Design spaces help reason about design alternatives

Recent research demonstrates a broad array of mobile phone input techniques for ubiquitous computing application scenarios. To make sense of the cumulative knowledge, we systematically organize the input techniques to give insights into the design space. The design space is an important tool for helping designers of ubiquitous computing applications to identify the relationships between input techniques, and to select the most appropriate input technique for their interaction scenarios. Design spaces can also be used to identify gaps in the current body of knowledge and suggest new designs (Zwicky, 1967).

The new ubiquitous design space draws inspiration from seminal work

Looking to Foley, Wallace and Chan's classic paper (Foley *et al.*, 1984), we find a taxonomy of desktop input devices that are structured around the graphics subtasks that they are capable of performing (Position, Orient, Select, Path, Quantify, and Text Entry). These subtasks are the elementary operators that are combined to perform higher level interface tasks and will be elaborated upon in later sections. In this chapter, we structure our analysis of smart phones as ubiquitous input devices using this taxonomy. This analysis builds on classic design spaces (Buxton, 1983; Card *et al.*, 1991) and extends our own previous work (Ballagas *et al.*, 2003,

2006, 2008) on the design space of input techniques. In our analysis, we blur the line between smart phones and personal digital assistants (PDAs) because their feature sets continue to converge.

Graphics subtasks also apply to ubicomp

Although Foley *et al.*'s analysis was completed with the desktop computing paradigm in mind, the subtasks in their analysis are still applicable to ubiquitous computing today. They naturally apply to situated display interactions; however, their applicability is not limited to graphical interactions.

Foley *et al.*'s taxonomy uses the following input characteristics to further classify input techniques:

Feedback

Feedback can be continuous or discrete

Continuous interactions describe a closed-loop feedback, where the user continuously gets informed of the interaction progress as the subtask is being performed. For example, when using a mouse, the current cursor position is continually fed back to the user. Discrete interactions describe an open-loop feedback, where the user is only informed of the interaction progress after the subtask is complete. For example, when selecting an object on a touch panel, the progress of the selection is not displayed until after the finger meets the surface to complete the selection of the desired item.

Interaction Style

Interaction can be direct or indirect

In direct interactions, input actions are physically coupled with the user-perceivable entity being manipulated (such as an image on a display). Physical coupling can be achieved when the feedback spatially coincides with the input action, or can be achieved at a distance if the user is manipulating a 3D ray (such as with a laser pointer) that intersects directly with the entity being manipulated. To the user, this appears as if there is no mediation, translation, or adaptation between input and output.

In indirect interactions, user activity and feedback occur in disjoint spaces (e.g., using a mouse to control an on-screen cursor). Scaling and abstraction between input actions and feedback are often necessary in indirect interactions.

In the following sections, each of Foley *et al.*'s subtasks will be examined in the context of mobile phone interactions.

2.1.1 The Position Subtask

Specify position in application coordinates

During a positioning task, the user specifies a position in application coordinates, often as part of a command to place an entity at a particular position. Positioning techniques can either be *continuous* where the object position is continually fed back to the user, or *discrete* where the position is changed at the end of the positioning task. Positioning tasks can further be differentiated using the directness of the interaction. In *direct* interactions, input actions are physically coupled with the object being positioned; in *indirect* interactions, user activity and feedback occur in disjoint spaces. We note that position could refer to screen position, or physical position in the real world. For example, the height of motorized window blinds can be adjusted using the position subtask.

The mobile phone has been used for positioning tasks in a variety of ways:

Continuous Indirect Interactions

Example: Remote Commander

1. *Trackpad.* A trackpad is a touch sensitive surface that is used as a relative pointing device, standard in modern laptops. Remote Commander (Myers *et al.*, 1998) enables individuals to use the touch screen on a PDA as a trackpad to control the relative position of a cursor on a remote situated display. In this interaction, the user's attention is concentrated on the situated display and no application-level feedback is provided on the PDA, thus the functionality of the PDA is essentially reduced to an input device.

Example:Joystick

2. *Velocity-controlled joystick.* A return-to-zero joystick controls the velocity of an object (such as a cursor) that is continuously repositioned on the display. Zero displacement of the joystick corresponds to no motion (zero velocity). Positioning with a velocity-controlled joystick (a temporally- and spatially-constrained task) has been shown to be inferior to positioning with a mouse (a spatially-constrained task) for desktop pointing scenarios (Card *et al.*, 1978). Silfverberg *et al.* (2001) have done an in-depth study of isometric joysticks on handheld devices to control the cursor on a situated public display. Many of today's mobile phones are shipping with simple joysticks with a push button for menu navigation.

Example: Tilt-to-scroll

3. *Accelerometers.* Accelerometers are beginning to emerge in handheld devices. For example, Samsung's SCH-S310 mobile phone comes with an integrated 3D accelerometer. Several researchers (Bartlett, 2000; Harrison *et al.*, 1998; Hinckley and Horvitz, 2001) have proposed interactions that allow users to scroll (e.g., through an electronic photo album) by tilting the handheld device. The scrolling is typically activated through a clutch mechanism, such as squeezing the sides of the device (Harrison *et al.*, 1998). The degree of tilting controls the speed of scrolling, making this a temporally-constrained positioning task similar to the velocity-controlled joystick. Although these tech-

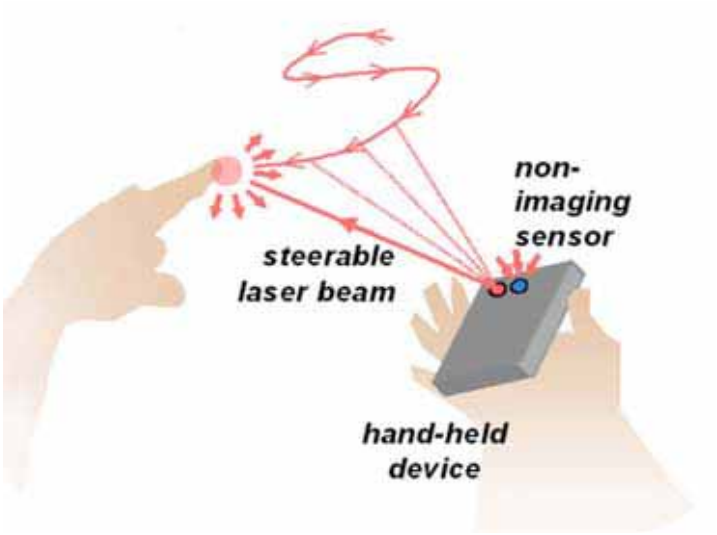


Figure 2.2: The Smart Laser Scanner: a 3D input technique for mobile devices using laser tracking (Cassinelli *et al.*, 2005).

niques were used to interact with an application directly on the device, they could clearly be extended to positioning tasks in ubiquitous computing environments.

4. *Camera tracking.* The Smart Laser Scanner uses a laser combined with a wide-angle photo detector (see Figure 2.2) to detect relative finger motion in 3-dimensional space (Cassinelli *et al.*, 2005). The laser beam is steered with a two-axis micro-mirror. The tracking principle is based on the backscatter of a laser beam. When the backscatter is disrupted the motion is deduced from the angle of the backscatter, and the laser is repositioned for the next measurement. Like other tracking techniques, it is possible for the device to lose track if the finger moves too fast, but input can easily be resumed by repositioning the finger to the laser. The research prototype of the tracker is fast enough to track the motion of a bouncing ping-pong ball.

   Example: Smart Laser Scanner

   C-Blink (Miyaoku *et al.*, 2004) rapidly changes the hue of a color phone screen to allow an external camera system to track the phone's absolute motion for cursor control on a large public display (see Figure 2.3). The hue sequence encodes an ID to allow multiple users to interact simulaneously and control independent cursors.

   Example:C-Blink

5. *Motion detection.* The Sweep (Ballagas *et al.*, 2005) interaction technique the phone is waved in the air to control relative cursor motion on a remote screen (see Figure 2.4). This is accomplished using motion detection – an image processing technique involving rapidly sampling successive images from the phone's camera and sequentially comparing them to determine relative motion in the (x, y, θ) dimensions. No visual tags are required. The screen on the phone can be ignored, and the camera doesn't even need to be pointed at the display. A clutch

   Example: Sweep is like an optical mouse

Figure 2.3: In the C-Blink system, the user waves the phone screen in front of a camera to control cursor position (Miyaoku *et al.*, 2004).

Figure 2.4: The Sweep technique uses camera input and optical flow image processing to control a cursor (Ballagas *et al.*, 2005).

mechanism, such as a button press, is used to activate the Sweep interaction. The clutch can be used to reposition the arm, similar to the way a mouse can be lifted to be repositioned without additional cursor motion.

6. *Location detection.* Location of the phone can also be used as input, where the user moves through physical space. Mogi (Licoppe and Inada, 2006), for instance, is a phone-based persistent item collection and trading game where the absolute geo-position of a subscriber correlates to the position in the game world. Mogi combines GPS (global positioning system) technology built into the phone with information from different mobile infrastructure towers from the network service provider to determine the player's position.

Example: Mogi

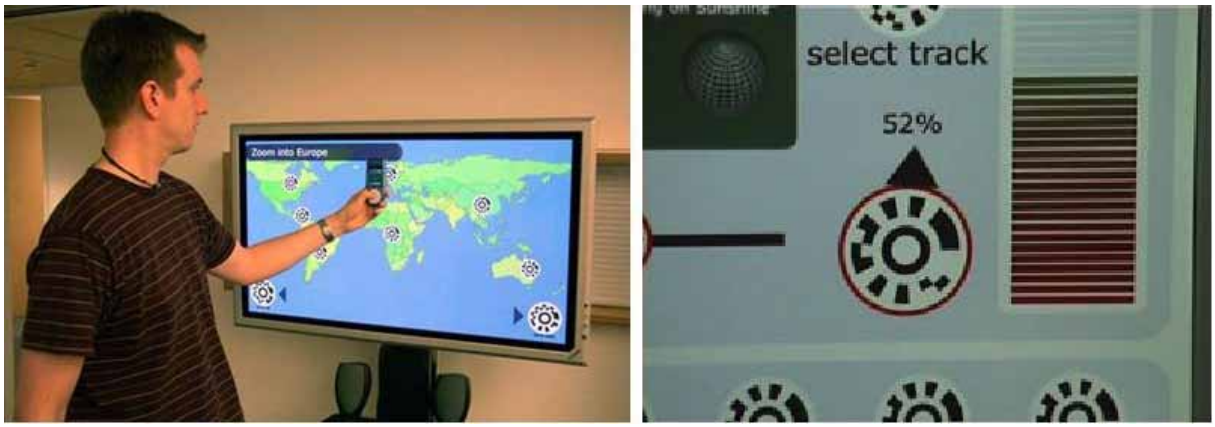


Figure 2.5: Using the phone to manipulate tagged widgets such as buttons, dials, and sliders (Madhavapeddy *et al.*, 2004).

Continuous Direct Interactions

7. *Camera tracking.* Madhavapeddy *et al.* (2004) present camera-based interactions involving tagging interactive GUI elements such as sliders and dials (see Figure 2.5). In manipulating the position and orientation of the phone camera, the user can position a graphical slider, or orient a graphical dial. Similarly, Direct Pointer (Jiang *et al.*, 2006) uses a handheld camera to track the standard cursor on the display (see Figure 2.6). An analogy can be drawn to the classic light pen with a tracking cross. As the light pen moves to a new position, the cross follows the motions of the pen. Tracking may be lost if the pen is moved too fast, but can be easily resumed by repositioning the pen back to the tracking cross. Madhavapeddy *et al.*'s interactions rely on the tagged GUI widget instead of a cross for tracking; in Direct Pointer, the mouse cursor is the modern equivalent of the tracking cross.

   Example: tracked GUI elements

   In the tracking examples above, the handheld device is responsible for tracking. An alternative is to use a tracker in the environment to track the output from a handheld device. For example, smart phones have been augmented with laser pointers, as in (Patel and Abowd, 2003), making them suitable for positioning tasks described by Olsen and Nielsen (2001) that use a camera in the environment to track the laser.

   Example: laser pointer

   The mobile phone can also be passively tracked using a camera in the environment such as in VisionWand (Cao and Balakrishnan, 2003). The user holds a passive handheld device that is augmented with distinctive markings (such as colored balls) at each end. Using two fixed cameras to perform stereo tracking, a 3D ray can be deduced from the orientation of the markings in the stereo view, assuming the distance of the markings on the device is known *a priori*. This allows using a projection of the ray as a pointing device for a fixed remote screen. The result is an interaction that is very similar to pointing using a laser pointer, except the ray is not a visible beam of light.

   Example: VisionWand

Figure 2.6: Direct Pointer uses a handheld camera to track a cursor displayed on the remote screen without relying on visual tags. (Jiang *et al.*, 2006)

This technique has an advantage over the standard laser pointer in that it provides an extra dimension of information: the distance to the display. The disadvantage of this interaction is that it is vulnerable to occlusion (e.g., by the users' own body) bringing into question the robustness of tracking in practical scenarios, although different camera configurations (such as from overhead facing downward) may solve these issues for certain interaction scenarios.

Discrete Indirect Interactions

8. *Directional step keys.* The location of an object is controlled using up, down, left, and right step keys for 2D applications, plus in and out for 3D. In the Blinkenlights project (Chaos Computer Club, 2002), users played the arcade classic "Pong" using the side of a building as a large public display. Each window equalled one pixel on the 18x8 pixel display (shown in Figure 2.7). Players connected to the display by making a standard voice call to a phone number. Pressing the number 5 on the phone keypad moved the paddle up, and the number 8 moved it down. The server controlling the "Pong" application would decode the tones generated from the key activity during the phone call and use them as application input. One of the notable things about this interaction is that it used the lowest common denominator of phone technologies. The communications channel was the standard voice channel, and the input was the numeric keypad, requiring no additional hardware or software besides what standard phones provide.

Example: Blinkenlights

Figure 2.7: The view of the façade of a building used to play the classic game "Pong" with buttons on the mobile phone controlling the paddle. (Chaos Computer Club, 2002)

Figure 2.8: Point & Shoot technique: (Left) The phone display is used to aim at a puzzle piece on a large display. (Middle) Pressing the joystick indicates *selection* and a visual code grid flashes on the large display to compute the target coordinates. (Right) The grid disappears and the targeted piece highlights to indicate successful selection. (Ballagas *et al.*, 2005)

Discrete Direct Interactions

9. *Camera image.* Using the Point & Shoot (Ballagas *et al.*, 2005) interaction technique, the user can specify an absolute position on a public display using a cross hair drawn over a live camera image on the mobile phone. To make a selection, the user presses a button while aiming at the desired target.[1] The button press triggers a brief

Example: Point & Shoot

[1] An alternative implementation of the Point & Shoot technique could use pen input instead of the cross-hair image so that the user repositions the cursor by selecting the desired position directly on the live camera image displayed on the phone screen.

overlay of a grid of 2D tags over the large display contents, as can be seen in the middle of Figure 2.8. The grid allows the phone to derive a perspective-independent coordinate system on the large display, which is enabled by the special properties of the Visual Code tags (Rohs, 2005a). Only one visual tag is required to establish a coordinate system, but a grid is used to increase the probability of having one tag entirely in the camera view. The drawback of the current implementation is that the tag grid is disruptive in multi-user scenarios, but future implementations could, for example, display the tags in infrared so that they are visible to the camera but not to other users.

Point & Shoot is related to the classic light pen, where position is discretely determined by displaying a raster scan when the user clicks a button on the light pen. When the raster scan is temporally sensed by the pen, the position of the pen is known because of a tight coupling between the pen clock and display clock. In Point & Shoot, a visual tag grid replaces the functionality of the raster scan except its mechanics are spatial rather than temporal. The lack of temporal dependencies makes Point & Shoot robust to different display technologies and the loose coupling between camera and display.

Table 2.1 summarizes the range of mobile position techniques at a glance. The breadth of positioning techniques is relatively large, making it difficult to choose which technique is most appropriate for a particular application scenario. To help with this selection, it is important to examine different figures of merit for each device.

Evaluating Positioning Techniques

There have been only a handful of thorough evaluations of the different ubiquitous mobile input techniques (Ballagas *et al.*, 2005; Myers *et al.*, 2002; Silfverberg *et al.*, 2001; Wang *et al.*, 2006) as the field is still relatively new. These studies are difficult to compare directly since they each used different experimental parameters, and some evaluations were not done in the context of ubiquitous computing interaction scenarios. Therefore, rough estimates for a variety of ergonomic measures are used to create a high-level comparison table for the positioning task presented in Table 2.2. These rough estimates are derived using our knowledge of the interaction techniques for mobile phones and the collective knowledge of their desktop computing counterparts. The ergonomic parameters are mostly borrowed from Foley *et al.*'s survey of interaction techniques.

Evaluations are rare, and difficult to compare

The evaluation measures are grounded in psychological and physiological foundations. Card *et al.* (1983) provide an integrated survey of the various fundamental theories in a way that makes them more accessible and easier to use during analysis. Central to this work is the human processor model, which brings knowledge of the perceptual, cognitive, and motor processes of

Psychology provides a good basis for comparison

	Foley	Mobile Phone Interactions	
	In Environment	On Phone	In Environment
Direct Pick	**Search for Light Pen** (Raster Scan)	**Camera + On-Screen Cursor** [Point & Shoot]	**Visual Tags**
		Laser Pointer + [Olsen]	**Camera**
	Light Pen Tracking	**Camera Tracking** + [Madhavapeddy]	**Visual Tags**
		Camera Tracking + [Direct Pointer]	**On-Screen Cursor**
		Vision Markers + [VisionWand]	**Camera**
		Camera + Touch-Screen	
Direct with Locator Device	**Touch Panel**		
Indirect with Locator Device	**Mouse**	**Camera** [Sweep]	
		Display + [C-blink]	**Stationary Camera**
	Joystick (Velocity)	**Joystick** (Velocity) [Silfverberg]	
	Tablet	**Trackpad** [Remote Commander]	
	Trackball		
	Cursor Control Keys with Auto-Repeat	**Cursor Control Keys with Auto-Repeat**	
	Joystick (absolute)	**Accelerometer** [Harrison + Rock 'n' Scroll]	
		Steerable Laser + Wide Angle Photodetector [Cassinelli]	
Indirect with DIrectional Commands/ Button Push	**Up-Down-Left-Right Arrow Keys**	**Directional Step Keys** [Blinkenlights]	
	(See Selection)	(See Selection)	
Location Detection		**GPS** + [Mogi]	**Cell Network Towers**
Numerical Value/ Numerical Coordinates/ Character String Name	(See Text Input)	(See Text Input)	

Table 2.1: Summary of POSITION techniques using a smart phone as an input device.

a human together under a single model. Ideally, a user interface minimizes the work required for each of these basic psychological processes.

The comparison table also incorporates various ergonomic measures designed to capture the efficiency of users executing the subtask, the accuracy they can achieve, and the pleasure the user derives from the process. The individual measures used in our comparison table are as follows:

- Perceptual Load refers to the difficulty for the user to recognize with their own senses the physical stimuli and feedback of the interaction. For example, in the Point & Shoot interaction, users need to shift their perceptual attention between a large display and the phone screen to isolate a target in the phone camera view, leading to a comparatively high perceptual load. Sweep has a low perceptual load because the user can focus their attention on the remote screen, similar to using a mouse.

- Cognitive Load refers to the difficulty for the users to organize and retrieve information related to the interaction technique. For example, the joystick has a relatively low cognitive load because of the simplicity of the interaction sequence.

- Motor Load refers to the physical movement required to execute the action after the appropriate action has been determined in the cognitive process. For example, Mogi is classified as a high motor load technique because the user needs to physically move at the city scale to specify the necessary position.

- Motor Acquisition Time characterizes the amount of time for the processes involved in the interaction technique (e.g., reaching for an object, moving to a certain target area, rotating to a certain orientation, etc.). For example, Point & Shoot has a high motor acquisition time because of the aiming required. Remote Commander has a relatively low motor acquisition time, similar to the trackpad on a laptop.

- Visual Acquisition Time characterizes the amount of time it takes to perceive the physical stimuli of the interaction technique. For example, Point & Shoot has a high visual acquisition time because the target needs to be visually acquired on two separate displays. The laser pointer has a low visual acquisition time because the selected item and the pointer visually coincide on the same display surface.

- Ease of Learning characterizes the level of skill that is required to use the device. The Sweep technique is relatively difficult to learn because one must understand the clutch concept to interact fluidly with the system. The joystick, on the other hand, is easy to learn because of the simplicity of the interaction steps.

- Fatigue characterizes how tiring the interaction technique is to perform. Rock 'n' Scroll is a relatively low fatigue interaction since tilting the arm is not a very muscle intensive activity. Sweep, on the other

hand, requires larger arm motions that could tire the user after extended use.

- Error Proneness characterizes the susceptibility for errors of the input technique, the degree to which the interaction technique by its design allows/avoids errors, e.g., if possible movement trajectories match the degrees of freedom of the required input then certain errors can be avoided. Laser pointers, for example, are susceptible to natural jitters in the arm. Remote Commander, on the other hand, allows for more fine-grained control since the pen can utilize the friction against the touch sensitive surface to stabilize the interaction.
- Sensitivity to Distance. Users in ubiquitous computing scenarios typically have freedom of motion, making the amount of separation between the user and the target in the environment (such as a large display or other device) dynamic and unpredictable. Thus, the range of distances the interaction will support is an important design consideration. Interactions that are based on aiming, such as laser pointers, become more difficult to perform when further away, where targets are perspectively smaller. Other techniques, such as the Sweep technique are not significantly affected by distance of interaction.

2.1.2 The ORIENT Subtask

The ORIENT subtask involves specifying a heading or direction instead of a position. Like POSITION, ORIENT is also not limited to graphics subtasks as it can relate to physical orientation in the real world, such as a security camera, a spot light, or a steerable projector. Some of Foley *et al.*'s original graphics interactions carry over directly to ubiquitous computing including *indirect continuous orientation with velocity-controlled joystick* and *discrete orientation with angle type-in*. The remaining techniques observed in our survey include:

Specify a heading or direction

Continuous Indirect Interactions

1. *Locator device.* The user can specify the angle of orientation by using a continuous quantifier or one axis of a positioning device. The Sweep technique supports detection of rotation around the Z-axis (perpendicular to the display) allowing interactions like rotating a puzzle piece in a jigsaw puzzle application, where the phone is used like a ratchet to adjust orientation. The image processing used by Sweep also detects rotation around the X and Y-axis. However, for better performance as a positioning device, rotation around the Y-axis is mapped to translation along the X-axis and rotation around the X-axis is mapped to translation along the Y-axis.

   Example: Sweep

2. *Camera tracking.* VisionWand (Cao and Balakrishnan, 2003) uses a set of cameras in the environment to track the absolute orientation of

Project (Author)	Interaction Type	Ergonomic Measures								
		Cognitive Load	Perceptual Load	Motor Load	Visual Acquisition	Motor Acquisition	Ease of Learning	Fatigue	Error Proneness	Distance Sensitivity
Remote Commander (Myers et al.)	Continuous Indirect	●	●	.	●	.	.	.	.	.
Isometric Joystick (Silfverberg et al.)		●	●	.	.	●	.	.	●	.
Tilt to Scroll (Harrison et al.)		●	●	●	●	⬤	●	●	●	.
Smart Laser Scanner (Cassinelli et al.)		●	●	●	●	●	●	●	⬤	.
C-blink (Miyaoku et al.)		●	●	⬤	●	●	●	⬤	●	●
Sweep (Ballagas et al.)		●	●	⬤	●	⬤	⬤	⬤	●	.
Mogi (Licoppe et al.)		⬤	.	⬤	.	⬤	.	⬤	⬤	.
Visual Tag Widgets (Madhavapeddy et al.)	Continuous Direct	.	●	●	●	●	●	●	●	⬤
Direct Pointer (Jiang et al.)		.	●	●	●	●	.	●	●	⬤
Laser Pointer Interaction (Olsen et al.)		.	●	●	●	●	.	●	⬤	⬤
VisionWand (Cao et al.)		●	●	●	●	●	.	●	⬤	⬤
Blinkenlights (Chaos Computer Club)	Discrete Indirect	.	●	.	⬤	●	●	.	●	.
Point & Shoot (Ballagas et al.)	Discrete Direct	●	⬤	●	⬤	⬤	●	●	⬤	⬤

Table 2.2: Rough estimates of ergonomic measures to compare mobile phone-based POSITION techniques (small circle = low, medium circle = medium, large circle = high).

Example: VisionWand

a marked handheld device. The technique requires that at least two markers are visible in at least two camera viewpoints to determine the orientation in 3-dimensional space.

Continuous Direct Interactions

Example: tagged GUI dials

3. *Camera tracking.* Madhavapeddy's tagged GUI dials (Madhavapeddy *et al.*, 2004) can be oriented using the phone camera to track rotation movement. Similar to the Sweep technique, the phone is used like a rachet to adjust orientation.

Example: compass

4. *Compass.* Electronic compasses such as the Honeywell HMC1052 magnetometer can be used to detect the physical orientation of the phone with a +/-3° error, enabling a continuous and direct ORIENT task. These or similar sensors could be easily incorporated into future mobile phone applications.

	Foley	Mobile Phone Interactions	
	In Environment	On Phone	In Environment
Direct Pick		**Camera** [Point & Shoot]	
		Camera Tag Tracking + **Visual Tags** [Madhavapeddy]	
		Vision Markers + **Camera** [VisionWand]	
Indirect with Cursor Match/ Locator Device		**Camera** [Sweep]	
	Joystick (Velocity)	**Joystick** (Velocity)	
	Joystick (absolute)		
Numerical Value/ Numerical Coordinates/ with Character String Name	(See Text Input)	(See Text Input)	

Table 2.3: Summary of ORIENT techniques using a smart phone as an input device.

Project (Author)	**Inter-action Type**	**Ergonomic Measures**								
		Cognitive Load	**Perceptual Load**	**Motor Load**	**Visual Acqui-sition**	**Motor Acqui-sition**	**Ease of Learning**	**Fatigue**	**Error Prone-ness**	**Distance Sensi-tivity**
Sweep (Ballagas et al.)	**Continuous Indirect**	●	●	⬤	●	⬤	⬤	⬤	●	.
Isometric Joystick (Silfverberg et al.)		.	●	.	.	●	.	.	●	.
Visual Tag Widgets (Madhavapeddy et al.)	**Continuous Direct**	.	●	●	●	●	●	⬤	●	⬤
VisionWand (Cao et al.)		●	●	●	●	●	.	●	⬤	⬤
Point & Shoot (Ballagas et al.)	**Discrete Direct**	⬤	⬤	●	⬤	⬤	●	●	⬤	⬤

Table 2.4: Rough estimates of ergonomic measures to compare mobile phone-based ORIENT techniques (small circle = low, medium circle = medium, large circle = high).

Discrete Direct Interactions

Example: Point & Shoot

5. *Camera image.* The Point & Shoot technique supports discrete orientation along the Z-axis. As the user aims at a target, they rotate the phone to specify the desired Z-orientation using the aiming cross-hair as an axis of rotation.

The range of ORIENT techniques are summarized in Table 2.3. To help guide designers in selecting the most appropriate ORIENT technique, rough estimates for performance measures are given in Table 2.4.

2.1.3 The SELECT Subtask

Choose from a set of alternatives

In many interaction scenarios, the user must choose from a set of alternatives, such as a menu of icons. The SELECT subtask addresses this style of interaction. The SELECT subtask is commonly accomplished by arranging the items spatially in a graphical user interface, allowing the user to complete the selection using a cursor controlled through the POSITION subtask. Instead of icons, the set of alternatives might be a list of commands. However, selection is not limited to graphical interactions as a user may select a physical object to operate upon, such as selecting a lamp to adjust its setting. Many selection techniques carry over directly from Foley *et al.*'s earlier analysis such as *character string name type-in* common for command prompts, or *button push–soft keys* where buttons are located on the edge of the display area with their labels displayed on screen. The remaining selection techniques are as follows:

Continuous Indirect Interactions

Example: VisionWand

1. *Gesture recognition.* The user makes a sequence of movements with a continuous positioning device such as the joystick, camera, trackpad, or accelerometers. For example, VisionWand (Cao and Balakrishnan, 2003) demonstrates a rich gesture vocabulary using stereovision to track a passive wand. For example, a tapping gesture is used to allow selection of the current cursor position specified by the orientation of the wand. This gesture interaction is indirect, but with other technologies gestures can also occur directly on the feedback surface, such as circling a group of objects on a touch screen.

Continuous Direct Interactions

Example: RFIG

2. *Tagged objects.* RFIG Lamps (Raskar *et al.*, 2004) allows a handheld projector to be used to select objects with photosensitive RFID tags in the physical world (See Figure 2.9). The handheld projector emits a gray-code pattern that allows the tags to determine their relative position in the projected view. Waving the handheld projector

Figure 2.9: A warehouse scenario employing RFIG. The users can highlight and select objects of interest combining a handheld projector and embedded light sensitive RFID tags (Raskar *et al.*, 2004).

around, you can navigate a cursor in the center of the projected view to select individual physical objects.

Discrete Indirect Interactions

3. *Voice recognition.* The user speaks the name of the selected command, and a speech recognizer determines which command was spoken. The Personal Universal Controller (Nichols and Myers, 2006) supports automatic generation of speech interfaces (as well as graphical interfaces) to issue commands to objects in the real world.

Example: Personal Universal Controller

Discrete Direct Interactions

4. *Tagged objects.* Tagged objects can be used to present information on a wireless mobile computer equipped with an electronic tag reader, as demonstrated by the early E-tag project (Want *et al.*, 1999). For example, selecting a book by scanning its embedded RFID tag would activate a virtual representation of the object on the screen, such as a web-reference to the book allowing it to be purchased. Similar interactions have also been proposed for visual tags in the environment (Rohs, 2005a) and tagged GUI elements (Madhavapeddy *et al.*, 2004; Toye *et al.*, 2007; Rohs, 2005b) where a camera is used to acquire an image to decode the selected tag. Patel and Abowd (2003) present a physical world selection method for mobile phones in which a modulated laser pointer signal triggers a photosensitive tag placed in the environment, allowing users to bring up a menu to control the object on their handheld device.

Example: RFID and visual tags

Table 2.5: Summary of SELECT techniques using a smart phone as an input device (Continued in Table 2.6).

	Foley	Mobile Phone Interaction	
	In Environment	On Phone	In Environment
Direct Pick		**Camera + On-Screen Cursor** [Point & Shoot]	
		Laser Pointer +	**Light Sensor** (e.g. camera) [Olsen, Semantic Snarfing, Patel]
	Light Pen Tracking	**Camera +**	**Visual Tags** [Madhavapeddy]
		Camera Tracking +	**On-Screen Cursor** [Direct Pointer]
		Camera +	**Pen Input**
		Vision Markers +	**Camera + Tapping Gesture Recognition** [VisionWand]
		Handheld Projector + RFID Reader	**Light Sensitive RFID Tags** [RFIG]
		RFID Reader +	**RFID Tags** [Want]
	Touch Panel		
Indirect with Cursor Match/ Locator Device	**Mouse**	**Camera** [Sweep]	
		Display +	**Stationary Camera** [C-blink]
	Joystick (Velocity)	**Joystick** (Velocity) [Silfverberg]	
	Tablet	**Trackpad** [Remote Commander]	
	Trackball		
	Cursor Control Keys	**Cursor Control Keys**	
	Joystick (absolute)	**Accelerometer** [Harrison, Rock 'n' Scroll]	
		Steerable Laser + Wide Angle Photodetector + Button Push [Cassinelli]	

Table 2.6: Summary of SELECT techniques using a smart phone as an input device (Continued from Table 2.5)

	Foley	Mobile Phone Interaction	
	In Environment	On Phone	In Environment
Indirect with Directional Commands/ Button Push/ Time Scan	**Programmed Function Keyboard**	**Programmed Function Keyboard**	
	Soft Keys	**Soft Keys**	
	Alphanumeric Keyboard		
Gesture Recognition/ Sketch Recognition		**Camera** + **Button Push Clutch** [TinyMotion]	
		Display +	**Camera**
		Accelerometer [Patel]	
	Light Pen	**Pen Input**	
		Steerable Laser + **Wide Angle Photodetector** + **Button Push for Clutch** [Cassinelli]	
		Laser Pointer +	**Light Sensor** (e.g. camera) [Olsen, Semantic Snarfing, Patel]
		Vision Markers +	**Camera** [VisionWand]
	Tablet + Stylus		
Location Detection		**GPS** +	**Cell Networks Towers** [Mogi]
Voice Input		**Microphone** +	**Voice Recognizer** [PUC]
Numerical Value/ Numerical Coordinates/ with Character String Name	(See Text Input)	(See Text Input)	

Project (Author)	Inter-action Type	Ergonomic Measures								
		Cognitive Load	Perceptual Load	Motor Load	Visual Acquisition	Motor Acquisition	Ease of Learning	Fatigue	Error Proneness	Distance Sensitivity
VisionWand (Cao et al.)	Continuous Indirect	•	•	•	•	•	.	•	●	●
RFIG Lamps (Raskar et al.)	Continuous Direct	.	•	•	•	•	.	•	●	●
Personal Unversal Controller (Nichols et al.)	Discrete Indirect	●	•	.	.	.	●	.	•	.
E-tag (Want et al.)	Discrete Direct	.	.	•	●	•	•	•	.	●
Visual tags (Rohs et al.)		.	.	•	•	•	•	•	.	●
Photosensitive tags + lasers (Patel et al.)		.	•	•	●	●	•	•	●	●
Semantic Snarfing (Myers et al.)		●	•	•	•	●	•	•	●	●

Table 2.7: Rough estimates of ergonomic measures to compare mobile phone-based SELECT techniques (small circle = low, medium circle = medium, large circle = high).

Example: Semantic Snarfing

5. *Laser pointer.* Myers *et al.* (2002) proposed a multi-layer selection technique called "semantic snarfing" that combines multiple devices in consecutive actions. First, a laser pointer integrated with a handheld computer is used to make a coarse-grained selection of a screen region on a display in the environment. A camera, also in the environment, detects laser activity on the display. The system then transmits the details of the selected screen region to the handheld device, which composes a GUI on the handheld screen to make the fine-grained selection with a stylus.

The range of SELECT techniques are summarized in Tables 2.5, and 2.6. To help guide designers in selecting the most appropriate SELECT technique, rough estimates for pefromance measures are given in Table 2.7.

2.1.4 The PATH Subtask

The PATH subtask involves specifying a series of positions and orientations over time. The PATH subtask has different requirements than POSITION and ORIENT because the movement is governed by the speed-accuracy trade-off (Schmidt *et al.*, 1979). Despite this, PATH adheres to the same taxonomy as the corresponding POSITION and ORIENT techniques, because a PATH task can be specified using the more primitive subtasks.

Specify a series of positions over time

2.1.5 The QUANTIFY Subtask

The QUANTIFY task involves specifying a value or number within a range of numbers. This technique is used to specify numeric parameters such as time or speaker volume. In ubiquitous applications, QUANTIFY tasks using phone input are typically accomplished through the GUI using 1D POSITION or ORIENT subtasks.

Specify a value

2.1.6 The TEXT ENTRY Subtask

TEXT ENTRY for mobile phones is a well-studied area (MacKenzie and Soukoreff, 2002) as it is central to text-based mobile communications like SMS (short messaging service) and personal information management functionality. Text entry also has many applications for ubiquitous applications, e.g., the Digital Graffiti (Carter *et al.*, 2004) project seeks to annotate public content on large public displays. This section is not intended to be a comprehensive survey of mobile text entry techniques, but we have selected a few examples to illustrate the design space. All of the techniques listed were originally designed for text input directly on the mobile phone, but could clearly be used for text entry for a ubiquitous computing application.

Text entry is central to mobile communications

Keyboard

Although some mobile phones and handheld devices feature a full QWERTY keyboard (albeit much smaller than their desktop counterparts), miniaturization trends make this type of keyboard impractical for the majority of mobile phone form factors. The most well known text entry techniques for mobile phones use a standard numeric keypad. For text entry from a 26 character alphabet using this keyboard, a mapping with more than one character per button is required. Following the classification by Wigdor and Balakrishnan (2004), there are two fundamental types of disambiguation: *consecutive*, where the user first selects a letter grouping and then an individual letter, or *concurrent*, where the user simultaneously selects the letter grouping and the individual letter.

Disambiguation can be consecutive or concurrent

Consecutive approaches are most common today

Consecutive approaches are the most common today. One approach to disambiguate text entry is MultiTap, which requires users to make multiple presses to select a single letter from the characters associated with a certain key. Another solution is to use a two-key disambiguation where the first key selects the letter group, and the second key specifies the letter in the group. Dictionary-based techniques predict the word being typed based on the different possibilities for combining the groups of characters assigned to each key. When multiple words match the key sequence, the user selects the intended word from a list (typically ordered by probability or frequency of use). Examples of dictionary-based text entry include T9[2] with a standard numeric keypad, or more recently, SureType[3] with an extended numeric keypad commercialized with the Blackberry Pearl (Kao *et al.*, 2007).

Example: TiltText

Concurrent approaches, however, demonstrate a lot of promise. For example, TiltText (Wigdor and Balakrishnan, 2003) combines the standard 12-key keypad with an accelerometer. To disambiguate which character is intended when a key is pressed, TiltText uses the tilt orientation of the handset. A keypress with the phone tilted to the left enters the first character on the key, forward tilt enters the second character, right tilt enters the third character, tilting towards the user enters the fourth character (if one exists for the key), and no tilt enters the numeric character.

Example:ChordTap

ChordTap (Wigdor and Balakrishnan, 2004) combines the standard numeric keyboard with additional "chording" buttons on the back of the phone. A user selects an individual letter by selecting the key group on the numeric keyboard while pressing the appropriate "chord" key on the back of the phone.

Example:TiltType

If miniaturization trends continue, TiltType (Partridge *et al.*, 2002) represents an interesting point in the design space that combines chord button presses to specify a letter grouping and tilting to allow the user to specify a particular character within that grouping. Using only 4 buttons and a 2-axis accelerometer, the technique supports an alphabet of 55 characters in a watch-sized form factor. Expert users can memorize the character positions, allowing the letter grouping and individual character within the grouping to be specified concurrently.

Speech Recognition

Text entry by speech recognition is not yet technically viable on mobile platforms, but we list it here for completeness. Technology is making rapid advances in the realm of speech processing. For example, system on a chip designs for speech processing (Ravindran *et al.*, 2005) have the potential to bring speech input to interactive text entry on mobile phones. Karpov *et al.* (2006) have developed a short message (SMS) dictation system for Symbian phones with a vocabulary of 23000 words. The language model is

[2] http://www.tegic.com

[3] http://www.blackberry.com/products/suretype/

adapted to words typically used in SMS messages.

Speech recognition could also be achieved in a compound architecture where the speech is recognized through an external computer (i.e., connected through a voice call) and sent back to the mobile phone.

Stroked Character Recognition

Pen-based techniques, such as Graffiti, are very common in the PDA form factor and are also available on a small portion of the handsets on today's market. However, any of the continuous positioning tasks discussed earlier are capable of generating stroke information necessary for stroked-character recognition. For example, TinyMotion (Wang *et al.*, 2006) demonstrates both English and Chinese stroked character recognition using camera-based motion estimation (similar to the Sweep technique).

Example: TinyMotion

Menu Selection

On-screen keyboards are common for touch sensitive displays, where the letters of the alphabet are displayed as a menu of buttons, commonly in a spatial layout similar to the QWERTY keyboard. If the screen size of the mobile phone is not large enough to depict a keyboard layout, items in the environment could be used to display the menu, where users select the characters using the selection subtask previously discussed.

The range of TEXT ENTRY techniques are summarized in Table 2.8. To help guide designers in selecting the most appropriate TEXT ENTRY technique, rough estimates for pefromance measures are given in Table 2.9.

2.2 Spatial Layout of the Design Space

Our interaction taxonomy is summarized in Tables 2.1, 2.3, 2.5, 2.6, and 2.8. Card *et al.* (1991) point out that this ad hoc format lacks a notion of completeness. Card then builds on the work of Buxton (1983) to create a systematic spatial layout of the design space of input devices that captures the physical properties of manual devices very well. However, it does not capture many aspects that are relevant to ubicomp interactions such as modality or feedback (Ballagas *et al.*, 2003).

The subtask summary tables lack a notion of completeness

Using Foley *et al.*'s taxonomy, we propose a 5-part spatial layout, shown in Figure 2.10, for mobile phone interaction tasks discussed in our survey including supported subtasks (POSITION, ORIENT, and SELECTION), dimensionality, relative vs. absolute, interaction style (direct vs. indirect), and feedback from the environment (continuous vs. discrete). Feedback

New 5-dimension spatial layout

	Foley	Mobile Phone Interactions	
	In Environment	On Phone	In Environment
Keyboard	**Alphanumeric**	**Alphanumeric** [Multitap, T9]	
		Alphanumeric + Accelerometer [TiltText]	
	Chord	**Alphanumeric + Chord** [ChordTap]	
		Chord + Accelerometer [TiltType]	
Stroked Character Recognition	**Tablet with Stylus**	(See Continuous Positioning)	
Voice/Speech Recognition	**Voice Recognizer**	**Speech Recognizer**	
		Microphone +	**Speech Recognizer**
Direct Pick from Menu with Locator Device	**Light Pen**	(See Selection)	
	Touch Panel	(See Selection)	
Indirect Pick from Menu with Locator Device	(See Positioning)	(See Positioning)	

Table 2.8: Summary of TEXT ENTRY techniques using a smart phone as an input device.

and interaction style have been previously defined in the introduction to Foley *et al.*'s taxonomy. We describe the remaining dimensions in more detail in the remainder of this section.

2.2.1 Supported Subtasks

Uses Foley *et al.*'s subtasks

When choosing the most appropriate input device for a particular interaction scenario, the subtasks an interaction supports are the primary consideration. By including the subtask directly in the design space, it becomes more useful as a design tool.

Project (Author)	Ergonomic Measures								
	Cognitive Load	Perceptual Load	Motor Load	Visual Acquisition	Motor Acquisition	Ease of Learning	Fatigue	Error Proneness	Distance Sensitivity
QWERTY	.	●	.	●	.	.	.	.	.
Multitap	.	●	.	.	●	.	.	●	.
T9	.	●	.	●	●	.	.	●	.
SureType	.	●	.	●	●	.	.	.	.
TiltText (Wigdor et al.)	●	●	●	●	●	●	●	●	.
ChordTap (Wigdor et al.)	●	●	●	●	●	●	●	●	.
TiltType (Partridge et al.)	⬤	●	●	●	●	⬤	●	●	.
Speech Recognition (Karpov et al.)	⬤	.	.	.	.	.	.	⬤	.
Stroked Character Recognition - Pen	.	●	●	.	●	.	.	⬤	.
Stroked Character Recognition – TinyMotion (Wang et al.)	●	⬤	⬤	.	⬤	.	⬤	⬤	●

Table 2.9: Rough estimates of ergonomic measures to compare mobile phone-based TEXT ENTRY techniques (small circle = low, medium circle = medium, large circle = high).

2.2.2 Dimensionality

Dimensionality refers to the number of dimensions the interaction supports. Dimensionality can indicate spatial dimensions (X, Y, Z) or rotational dimensions (rX, rY, rZ). This distinction is visible in our design space by observing the subtask of the dimension. Following Card *et al.* (1991), if a particular interaction uses a combination of dimensions across different points in the design space, the relationship is indicated using a merge composition operator (a solid line). In contrast to Card's notation, our merge composition operators are connecting subtasks, not spatial sensor dimensions.

Number of dimensions the interaction supports

2.2.3 Relative vs. Absolute

Relative input is specified with respect to interaction history: the input technique provides information about the amount of change from the previous state. Relative input can be specified regardless of the current physical properties, such as position and orientation. For example, standard desktop mouse input is specified through motion across the desktop regardless of the physical position of the mouse on the desktop.

Absolute input is specified with respect to current physical properties, and can be specified indepedently of any interaction history. For example, stylus input can be used to provide absolute positional information on a screen space.

2.2.4 Other Relevant Attributes of Interaction Devices

Other dimensions may be appropriate for further insights

It should be noted that this set of dimensions is not comprehensive, and other dimensions such as resolution, direction (input vs. output), and modality may provide further insights into the design space. However, the design space depicted in Figure 2.10 does provide an interesting overview of the interaction techniques covered in this work. Using this graphical layout, we are able to pinpoint gaps in the breadth of the interaction techniques surveyed, and can anticipate opportunities for future work. For example, our space shows no interaction that supports 3-dimensional relative direct orientation. An alternative layout might include direction and modality, which would demonstrate the sparse usage of auditory and haptic feedback in these techniques.

Designing for Serendipity

Establishment of data connection must be considered

One key design consideration is the ease and speed of setting up a data connection between the phone and the environment or the device it is controlling. Olsen *et al.* (2001) refer to the ad hoc assembly of input and output resources as *opportunistic assemblies*. In some of the interactions surveyed, the data connection is inherent in the physical properties of the device. For example, VisionWand (Cao and Balakrishnan, 2003) is a completely passive system and requires no additional action on the user's part to start the interaction.

The C-Blink interaction is classified as highly serendipitous as the users merely launch an application on their mobile phone to interact with a display; no network connection or handshaking is required. The RFIG Lamps project also falls into this category because RFID tags are so simple in terms of communications protocol that no connection needs to be established before data can be transferred.

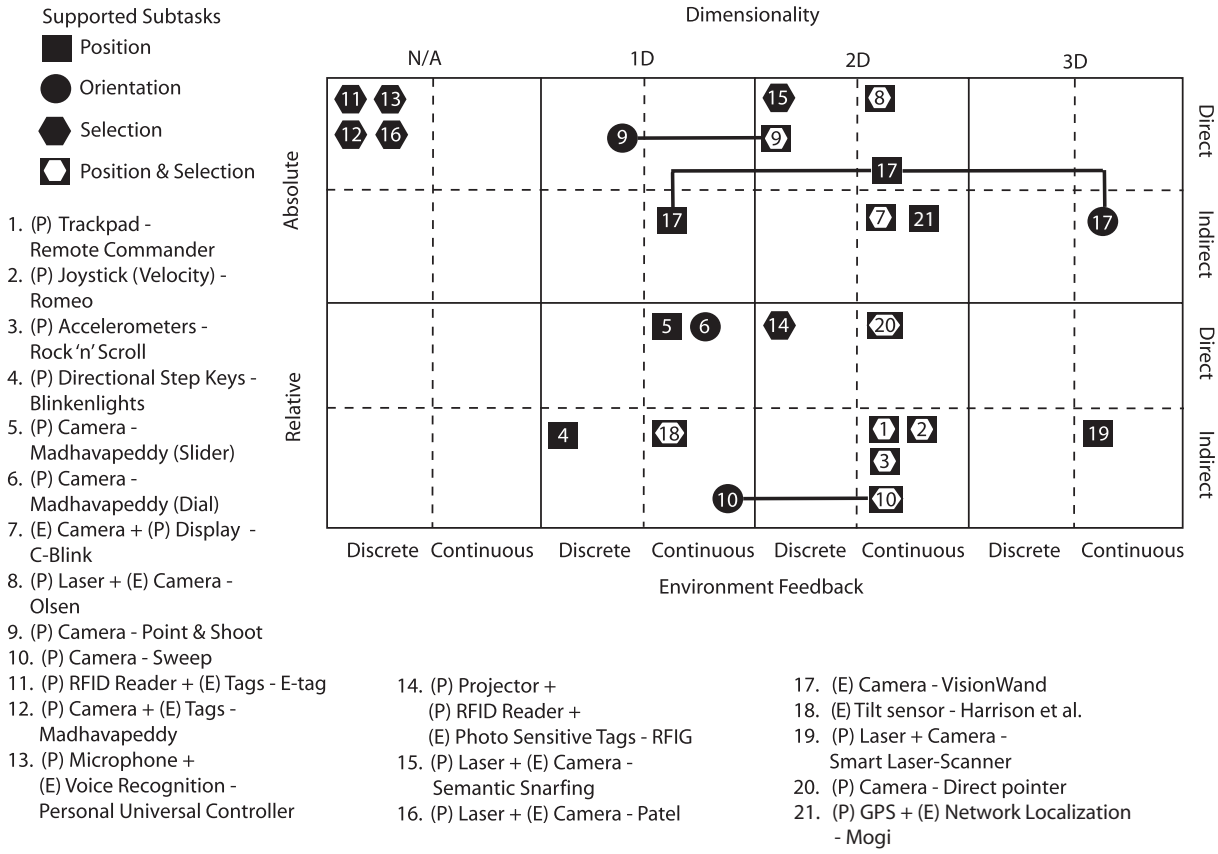


Figure 2.10: Classification of different mobile phone interactions that have been implemented in the projects surveyed. Inspection of the diagram reveals opportunities for future work – for instance, developing interaction techniques that support 3D relative direct orientation. In the listing of techniques, (P) indicates capabilities of the phone, and (E) indicates capabilities of the environment.

For projects that use short range wireless communications models such as Bluetooth, visual or RFID tags can be used to encode the connection information for the environment, creating a very low threshold of use.

Social Acceptance

Smart phones today are social devices. While smart phone ubiquity seems inevitable, social acceptance will influence the success of these new interactions. Remind yourself, for example, of the first time you came across a person using a wireless headset to communicate via their mobile phone. For many people, this communication technique is still awkward and strange, particularly in public places. Smart phone interaction will require users to perform particular actions and behaviors which might feel unintuitive and awkward to them. Furthermore they will perform these actions in the presence of passive or active others, both familiars and strangers. On one hand, outside observers might find these interactions disturbing or embarrassing, but on the other hand these kinds of interaction have the potential to raise

Some interactions may be awkward; others may raise social status

your social status, similar to the way phones themselves are status symbols for part of our society.

2.3 Design Spaces in the Design Process

Design spaces are particularly useful design tools as a part of a human-centered iterative design process (Nielsen, 1993). One of the pitfalls of iterative human-centered design is that if you pick a poor starting point, you may reach a peak in the usability of a particular design without reaching your desired usability goals. In this case, it may be necessary to throw the design away and start over. False starts are relatively painless early in the design process, but can be extremely expensive if determined late in the design process. In order to minimize the risk of false starts, a parallel design strategy (Nielsen and Faber, 1996) can be used, where multiple designs can be explored independently early in the design process. As the designs mature, the best design becomes clear, or the strengths of the top designs can be merged to a unified design. Using the design space, designers can more easily reason about alternative input techniques in a parallel design process.

Parallel design helps prevent false starts

As a concrete example, REXplorer (Ballagas *et al.*, 2007b) is a pervasive spell-casting game that allows tourists to explore the history of the medieval buildings in Regensburg, Germany. The game premise is that historical spirits are trapped inside of medieval buildings. Players need to interact with the spirits to learn their stories and perform quests on their behalf to earn points in the game. The game design called for spell-casting as the primary interaction metaphor; in order to awaken a spirit, one of four spells must be cast.

Example: spell-casting interaction

Choosing one spell out of four can be characterized as a Select subtask. The design space was used to identify a set of design alternatives that we initially considered:

Alternatives for spell-casting input

1. Four dedicated spell buttons,
2. Selecting one of four spells from on-screen menu,
3. Recognition of spell gestures. We noted that gestures are actually specified using the path subtask. Then we came up with gesture input alternatives including:
   (a) Pen trace across a touch screen,
   (b) Path using camera-based motion detection to allow the phone to be used like a magic wand.

Engagement over efficiency

After preliminary analysis with our target group (students aged 15-25), we decided to go with the camera-based motion detection solution (see

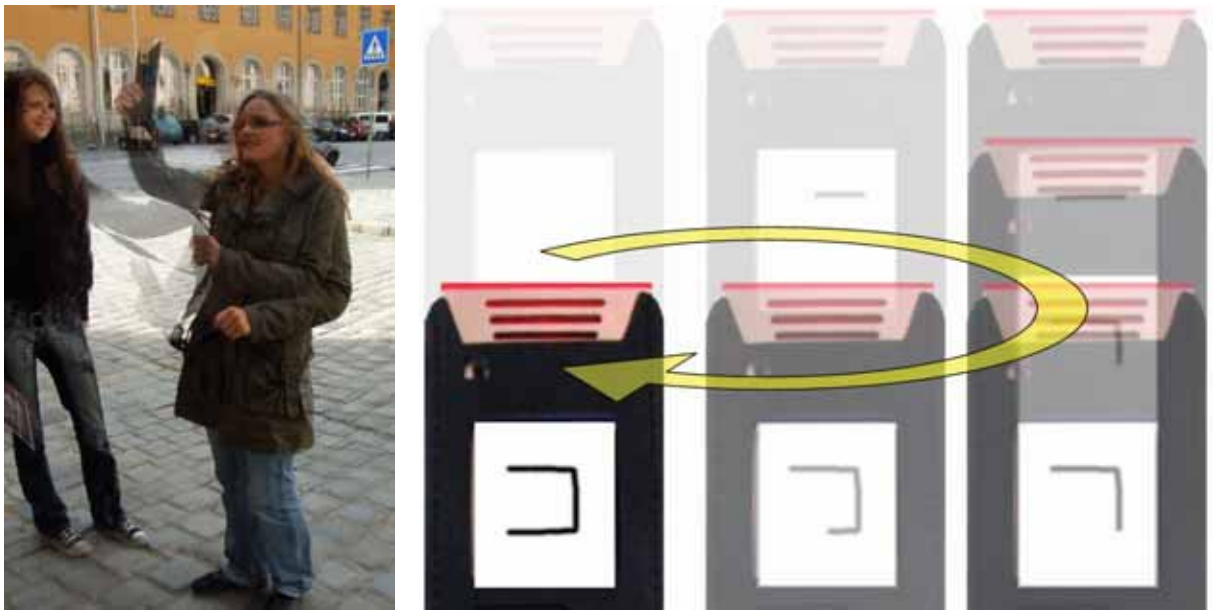


Figure 2.11: REXplorer uses camera-based motion estimation to allow players to cast spells using the path subtask (Ballagas *et al.*, 2007b).

Figure 2.11). Waving the phone through the air is not the most efficient technique, but is the most similar to the spell-casting metaphor. Also, this physical style of gesture was more likely to create an engaging experience (Hummels, 2000).

Mixed response was resolved through unified design

Later in the design process, after a working gesture recognition system was created, we did a full playability test. Most of the test players found the gestures to be an important element of gameplay. They found it heightened the sense of magic and mysteriousness. However, we also discovered during the playability tests that a few of our players (especially our older participants) found the gestures awkward. As a compromise, we created a unified design where an alternative gesture selection mechanism through an on-screen menu can be used anytime an invalid gesture is performed, effectively allowing people to avoid gestures altogether if desired. This final design encouraged the use of gestures for spell selection to promote engagement, but allowed an alternative selection mechanism to those who preferred to avoid gestures.

2.4 Chapter Summary

Our structured tour illustrates the state of the art in using smart phones to interact with and control our environments. The taxonomy organizes the range of techniques into families that help make functional relations between the mobile phone techniques and their desktop counterparts. The design space addresses the lack of a sense of completeness in the taxonomy, and structures the range of interactions in a way that helps visually identify

gaps and predict future interaction techniques. The design space can be used as a part of a human-centered iterative design process to help generate parallel or alternative designs. These methods of thought are intended to inspire new applications that use the mobile phone for interaction with the environment, as well as inform the design of future smart phone interaction techniques.

This also concludes the Chapter 2 of this work, in which we support the design phase of the human-centered iterative design process. The next chapter will discuss how to rapidly create functional prototypes of user interfaces and interaction techniques for ubiquitous computing, including those discussed in this chapter.

Chapter 3

Supporting Prototyping: Toolkit Support for Ubiquitous Computing Applications

"Our Age of Anxiety is, in great part, the result of trying to do today's jobs with yesterday's tools."

—Marshall McLuhan

After a design is conceptualized, it needs to be translated into a prototype that captures the interactive experience. Currently, only experts can effectively prototype, and deploy ubiquitous computing applications. Much of the effort in ubiquitous computing application development is still focused on low-level systems aspects, making it a long road before a system is functional enough for end-user testing. Once user testing can be peformed, it is often too expensive to make any significant changes. Toolkit support is required to simplify the design process, but recently, fieldwork on prototyping ubiquitous computing systems led Carter *et al.* (2007a) to conclude: "heterogeneity of ubicomp's input technologies may require different support architectures than GUI toolkits provide."

New tools are needed for prototyping in ubicomp

This chapter introduces the iStuff toolkit architecture to radically simplify design, prototyping, evaluation and deployment of ubiquitous computing systems. The architecture consists of several layers of abstraction to promote flexibility and provide a *high ceiling* for prototyping activities: it places few limits on prototyping behavior. At the same time, systems can be built and configured using rapid prototyping environments, such as Apple's Quartz Composer visual programming environment, to provide a *low threshold* for prototyping activities: the toolkit is not difficult to learn.

The iStuff architecture provides a high ceiling and low threshold of use

3.1 Requirements

Ubiquitous computing applications have different challenges and requirements than traditional desktop applications.

Ubicomp has more dimensions of multiplicity

M^5 The first, and possibly biggest, challenge is architectural: Software architectures need to move away from the scenario of a single user interacting with a single computer. Distributed Computing has already tackled this challenge on a technical level, so that multiple systems interacting with each other have become a commodity. On a user interface level, CSCW has looked at groups of co-located people collaborating using a single system (Single-Display Groupware, or SDG). Bier and Freeman (1991), for example, introduced one of the earliest user interface architectures for such systems. Its name, MMM, stood for multiple (input) devices, users, and editors. To implement the visions of ubiquitous and pervasive computing for an interactive environment, however, requires extending this concept to also include multiple systems (computers) and output devices (especially screens)—hence M^5.

Components must communicate without *a priori* knowledge of their counterparts

Extensibility and Incremental Integration. Our homes (Kidd *et al.*, 1999) and offices (Johanson *et al.*, 2002a) are becoming augmented with technologies to improve the way we live and work, but it is clear that this transformation will not happen overnight. New technologies are brought piecemeal into these environments (Edwards *et al.*, 2003). Physical spaces evolve slowly, and ubicomp technologies will be incrementally deployed (Rodden and Benford, 2003). Ubiquitous entities such as physical devices, applications, and services require a mechanism to coherently communicate without *a priori* knowledge of the other system components they may encounter. Moreover, the semantics of the interactions must be meaningful to both the individual components and the users of the environment. Therefore, systems must be designed such that they may be augmented with future devices and services whose feature set cannot be predicted in advance. Integration strategies are needed for situations where devices and applications have no direct knowledge of each other's function or even existence. We refer to this challenge as *incremental integration*, and note that solving these issues will also provide solutions for getting *current* devices and services to interoperate that were not designed to do so.

Interoperability relationships should adapt

Heterogeneity and Dynamic Composition Weiser's vision of ubiquitous computing (Weiser, 1991) consists of heterogeneous components seamlessly working together. People bring with them personal devices such as mobile phones (Weiser's tabs) and laptop computers (Weiser's pads), and these need to fluidly interoperate with resources embedded in the environment, such as wall sized displays (Weiser's boards). Related to this, different usage scenarios require different compositions of hardware and software components. Users of these

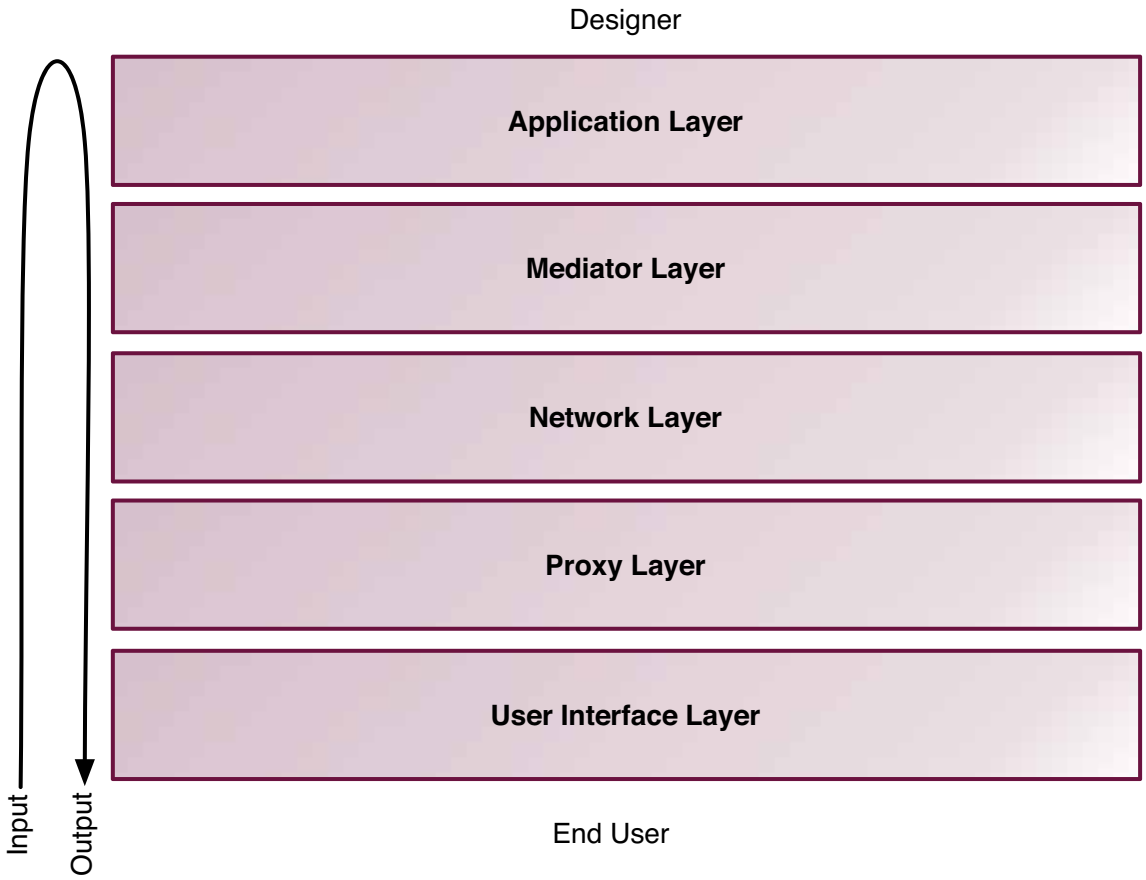


Figure 3.1: A layered model of the iStuff Toolkit Architecture.

environments also need a mechanism to fluidly task switch, which may require changes the interoperability relationships at run-time. They need to dynamically redirect their input or the system output to different computational resources in the environment, similar to the way we use the concept of *focus* to redirect our input to different applications in desktop environments.

3.2 iStuff Toolkit Architecture

The iStuff Toolkit architecture is designed to address these requirements. The five layer architectural model (see Figure 3.1) provides appropriate flexibility required to rapidly prototype and refine new user interfaces for ubiquitous computing, promoting a human-centered iterative design process. This architectural model is intended to be an abstract tool to reason about the layers of functionality, not a strict prescription for how the functionality should be implemented. An analogy can be drawn to the classic four-layered model of window system architectures (Gosling *et al.*, 1989), which was a useful tool to compare various window system implementations. In the following sections, we will examine each of these layers in

This model helps reason about the layers of functionality

Figure 3.2: Interactive Workspaces are ubiquitous computing environments that combine an array of input devices and displays to provide a coordinated user experience.

turn, starting from the User Interface Layer moving up to the Application Layer.

3.2.1 User Interface Layer

The UI Layer includes both physical and graphical UIs

The user interface layer is how the designer, through affordances and feedback, communicates to the end user. This layer encompasses all of the properties of what Norman (2002) refers to as the "system image". The user interface layer is not limited to graphical user interfaces; it also incorporates physical embodiment and form factor of the user interface. Hardware and form factor are important considerations for ubiquitous application designers because, as we have seen in earlier chapters, there is no default set of hardware, like a keyboard and mouse, for designers to depend on when constructing their applications.

UI components are distributed around a space

The user interface layer in ubiquitous computing applications is typically not spatially concentrated; instead, different elements of the user interface may be distributed throughout space. Consider, for example, an interactive workspace (Johanson *et al.*, 2002a) (see Figure 3.2) where multiple displays and an array of input devices work in concert to support tasks. Here the goal is to give the user the experience that they are interacting with a coordinated and unified space instead of a loose collection of independent components.

To simplify creation of ubiquitous user interfaces, the iStuff toolkit encourages reuse of both hardware and software components. For example, in the beginning of the iStuff project, several custom input and output devices were built to serve as physical widgets, which are reusable building blocks to prototype novel user interfaces (see Figure 3.3).

Hardware and software reuse are promoted

To examine the completeness of this early building block set, we can use the design space concept presented in Chapter 2. For this design space, the following dimensions were used:

- *Direction* - This attribute indicates whether a device is used to provide input, output, or both.
- *Sense Addressed / Modality Used* - For input devices, this attribute describes the modalities used to operate an input device – manual (e.g., mouse or stylus), visual (e.g., eye-tracking input), acoustic (e.g., sound or speech input), thermal (heat sensors), etc. For output components, this describes the sense(s) which perceive the output – visual (LEDs, displays), auditory (noise or speech output), haptic (force, temperature changes), etc.
- *Resolution* - For an input device, resolution is analogous to Card *et al.*'s property that classifies the domain provided by the device as ranging from a single, binary value to an infinite range of values. For output devices, the interpretation of resolution varies depending on the sense addressed. For visual output, resolution means number of pixels, levels of brightness, and/or number of colors. Resolution of auditory devices can range from one-bit (as in a buzzer) to near-infinite (as in a speaker). For haptic feedback, resolution describes whether a binary value (presence/absence of feedback) or a range of values can be provided.
- *Dimensions* - For manual input and visual output devices, the familiar spatial concepts of 1, 2, and 3D are applicable. Upon inspection, such concepts apply to other modalities as well – for instance, sound output could provide 3D information if high-quality "surround sound" speakers were used to provide a sense of location to the sound. Similarly, vocal input could carry with it dimensional information if triangulation techniques were used to pinpoint the location of the speaker. 0D interactions represent non-spatial interactions, such as simple audio.
- *Relative vs. Absolute* - This concept applies not only in the familiar domain of manual input (with a stylus providing absolute positional information while a mouse provides the relative variety), but to other domains/directions as well. For instance, an audio output device could be absolute, conveying the presence or absence of a sound, or it could be relative, conveying a change in pitch.

This design space helps identify gaps of coverage of our original hardware set. Inspection of the design space in Figure 3.3 reveals that the preliminary

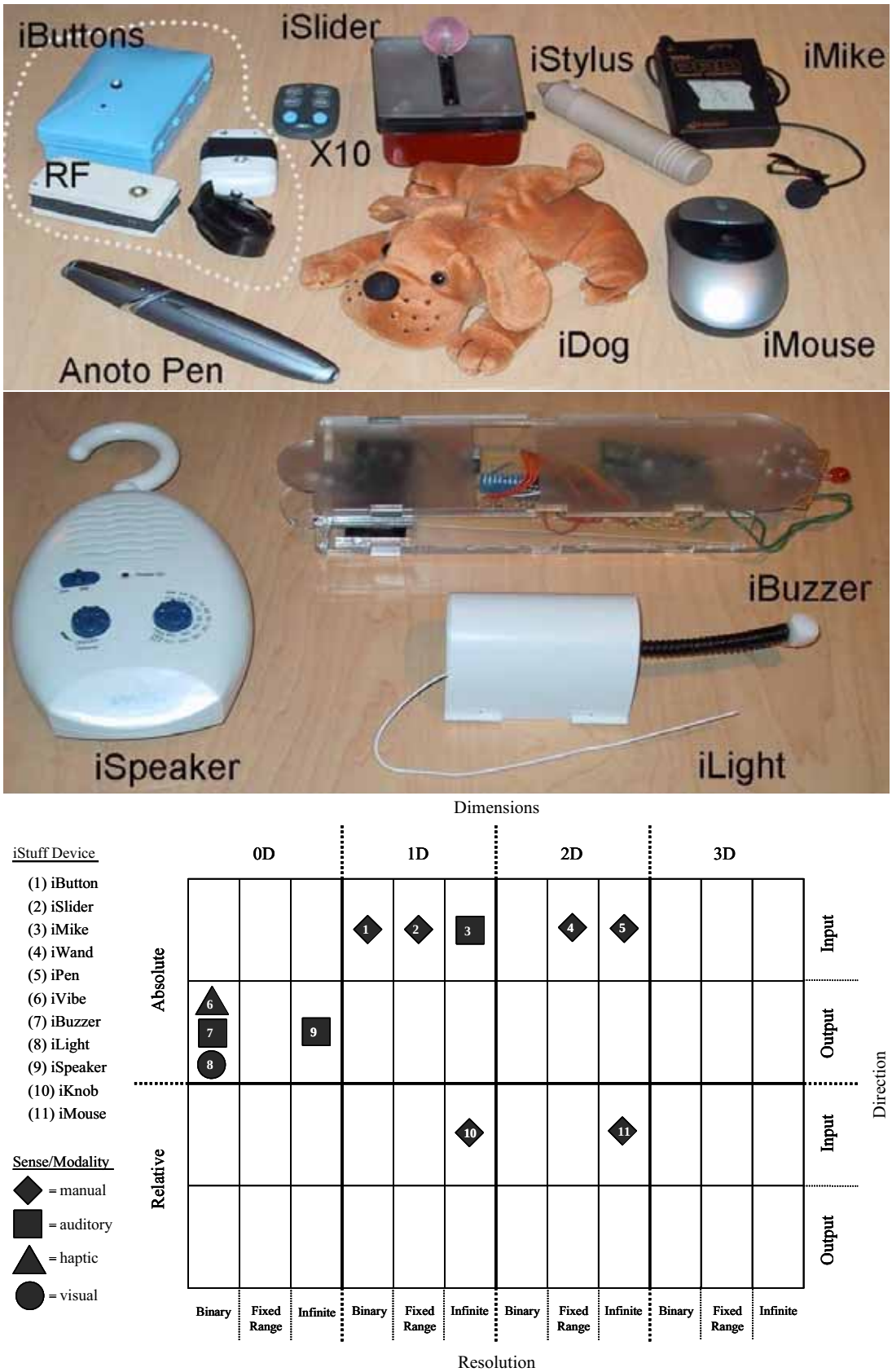


Figure 3.3: Custom built iStuff components are reusable modules that can combined to build a physical user interface prototype. (Top) contains input devices, (Middle) shows output devices, (Bottom) A design space can be used to illustrate the coverage of these devices.

set of iStuff hardware components were lacking 3D interactions, non-manual input devices, and higher resolution output components.

Other Physical Hardware Toolkits

A variety of hardware toolkits have emerged in recent years to help prototype physical interactions. BOXES (Hudson and Mankoff, 2006) provides a prototyping solution using common household items, to simplify early stage exploration of form factor. Toolkits like Phidgets[1] (Greenberg and Fitchett, 2001), Teleo[2], Calder (Lee *et al.*, 2004a), VoodooIO (Villar *et al.*, 2006) and Smart-Its[3] (Gellersen *et al.*, 2004) provide a set of reusable hardware components with accessible APIs to reduce the barriers of physical device prototyping. d.tools (Hartmann *et al.*, 2006) provides a set of software tools in addition to hardware components to support the full range of design, testing, and analysis activities in an iterative design process.

Many hardware toolkits have emerged

The custom built hardware components also have the disadvantage that for non-experts they are difficult to recreate or produce more instances. Therefore, the iStuff toolkit architecture also supports a wide range of commercially available hardware toolkits that have emerged since the project has begun including Phidgets, Teleo, and Smart-its.

Many components are commercially available

Software Components

The user interface layer also includes graphical user interfaces. iStuff encourages reuse of existing software components. Perhaps a designer wants to prototype a ubiquitous computing presentation application, but doesn't want to build a new presentation system (such as Microsoft PowerPoint) from the ground up to explore her ideas. Using the iStuff architecture, the designer can take an existing presentation application and adapt it to suit her needs.

We note that although our architecture does include graphical user interfaces, the classic window system layers are not represented directly by our model and are instead subsumed into this layer. This is because the affordances and feedback that make up the system image can either be represented physically through a custom hardware construction, or virtually through a graphical user interface. Often it is desirable to switch between virtual and physical representations of the same interface. For example in d.tools (Hartmann *et al.*, 2006), virtual representations of objects are used to allow a designer to simulate an interaction before the effort is made to build the physical prototype. Alternatively, the interactive virtual representation facilitates Wizard of Oz prototyping at test time if the physical prototype is not fully functional. Essentially this means that in the iStuff

Switching between virtual and physical

[1] `http://www.phidgets.com`
[2] `http://www.makingthings.com/teleo.htm`
[3] For Smart-Its hardware, see `http://www.particle-computer.net`

architecture layer model, clicking a virtual button with a mouse is functionally equivalent to pressing a physical button.

Layer Summary

Diverse components need to interoperate

The User Interface Layer comprises the "system image" for the user, including both physical and graphical user interface components. This layer is responsible for both enabling user input, and providing user feedback. The problem with using this diverse array of user interface components is interoperability. How does the designer get this range of components, produced by different (often competing) companies with different standards, to work together to provided a unified user experience? The remaining layers of the architecture show how this can be achieved.

3.2.2 Proxy Layer

Enables Network Layer access to all UI components

The Proxy Layer provides the first level of interoperability by enabling the heterogeneous UI components to communicate over the same network substrate. Gamma *et al.* (1995) describes a proxy as "a surrogate or placeholder for another object to control access to it". The role of the Proxy Layer in the iStuff Toolkit Architecture is to accommodate user interface components that are unable to interact with the Network Layer directly (perhaps because of lacking computational resources, or because they are legacy). The proxy communicates (e.g., through serial ports or application hooks) with a service, device, or application to send and receive Network Layer packets on its behalf.

Example: low computational resources

- *Case Study – original iStuff hardware*: The original iStuff hardware consisted of low-cost circuitry and pre-built electronic chips with custom wireless transmission capabilities (see Appendix A for full construction details). This made the hardware relatively simple to build for someone with basic knowledge of soldering. However the construction included no programmable components, making it impossible to implement the necessary protocols to communicate with the Network Layer. To resolve this, a proxy strategy was used to allow a desktop computer in the room to communicate with the Network Layer on their behalf. A simple transceiver was built, using compatible wireless components, that was connected to the desktop computer through a USB port. A Java native interface (JNI) extension was used to expose the USB device in Java which then encapsulated the input data from the custom-built wireless devices into a Network Layer packet.

Example: legacy software

- *Case Study – PowerPoint*: Existing desktop applications such as presentation applications perform their tasks well, but are not designed to adapt to new interaction scenarios of interactive workspaces. To expose presentation control functions for ubiquitous computing prototyping scenarios, we created a proxy that simulates key presses,

essentially enabling any keyboard shortcut (e.g., Next Slide, Previous Slide, Goto Slide Number, Full Screen, etc.) to be accessed from the network.

- *Case Study – iTunes*: Several operating systems provide high-level mechanisms or hooks for end-users to programmatically control applications. For example, in the Windows operating system, Visual Basic can be used to create macros to simplify execution of repetitive tasks. Similarly, in Mac OS X, AppleScript provides an accessible programming interface to control applications. For example, we have created a proxy for the iTunes music player on Mac OS X that uses AppleScript hooks to expose functionality to the Network layer including: play, pause, next track, previous track, volume, and retrieve track information, playlist information, album art, etc.[4]

Example: application hooks

- *Case Study – Smart-Its*: Smart-Its are sensor network modules that combines a miniature array of sensors with a wireless radio for networking. The communication with the Smart-Its is facilitated through the XBridge[5] device, which allows the sensor network modules to be accessed and configured through a local area network. Our Smart-Its proxy interprets the sensor data packets from the XBridge, canonicalizes the format and posts them to the Network Layer. For this proxy, we also allow the developer to configure the sensor boards, activate particular sensors, and set sampling rates through a basic GUI (see Figure 3.4).

Example: commercially available wireless sensor modules

- *Case Study – iSpeaker*: The iSpeaker is an off-the-shelf shower radio that we turned into a mobile speaker. The wireless communication is handled through a commodity FM transmitter that is plugged into the headphone jack of a desktop computer in the room. The proxy for the iSpeaker uses standard audio libraries to play sounds on the mobile speaker. This example highlights the flexibility of the proxy approach to expose the functionality of low-cost commodity components for ubiquitous computing application scenarios.

Example: low-cost commodity hardware

Proxy Manager

To simplify working with iStuff proxies, we have developed the proof of concept Proxy Manager GUI (see Figure 3.5), which provides an interface to organize and manage which proxies are running on a particular machine. The left side of the interface displays automatically discovered Network Layers that can be used. A hierarchical tree structure is used to organize the available proxies on the local machine. This tree is automatically generated by traversing user-specified proxy directories allowing the list to be easily extended. Proxies of interest can brought into the current workspace, where they can be started or stopped. The workspace represents a prototype configuration that can be saved for later use.

Manage active proxies through a GUI

[4] For an extensive list of examples of iTunes scripting support, see the following website: `http://www.dougscripts.com/itunes/index.php`

[5] `http://www.particle-computer.net/index.php?article_id=74`

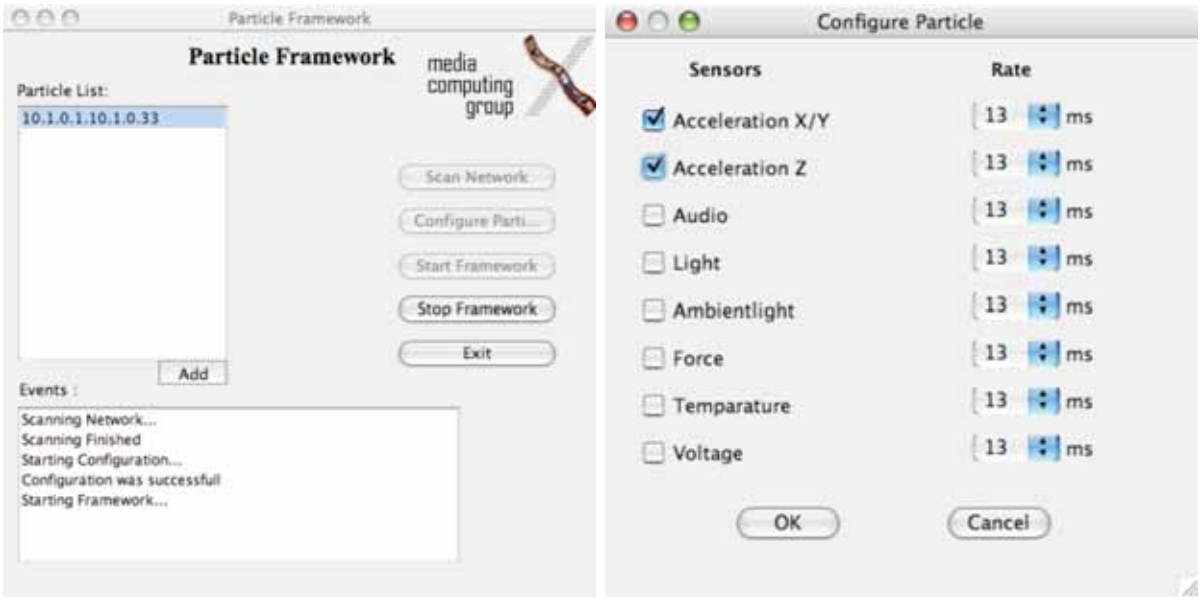


Figure 3.4: (Left) The Smart-Its proxy GUI allows developers to discover Smart-Its sensor boards, configure them, and activate communication with the Network Layer through this basic interface. (Right) The configuration menu allows the developer to activate sensors and set sampling rates for the sensor network module.

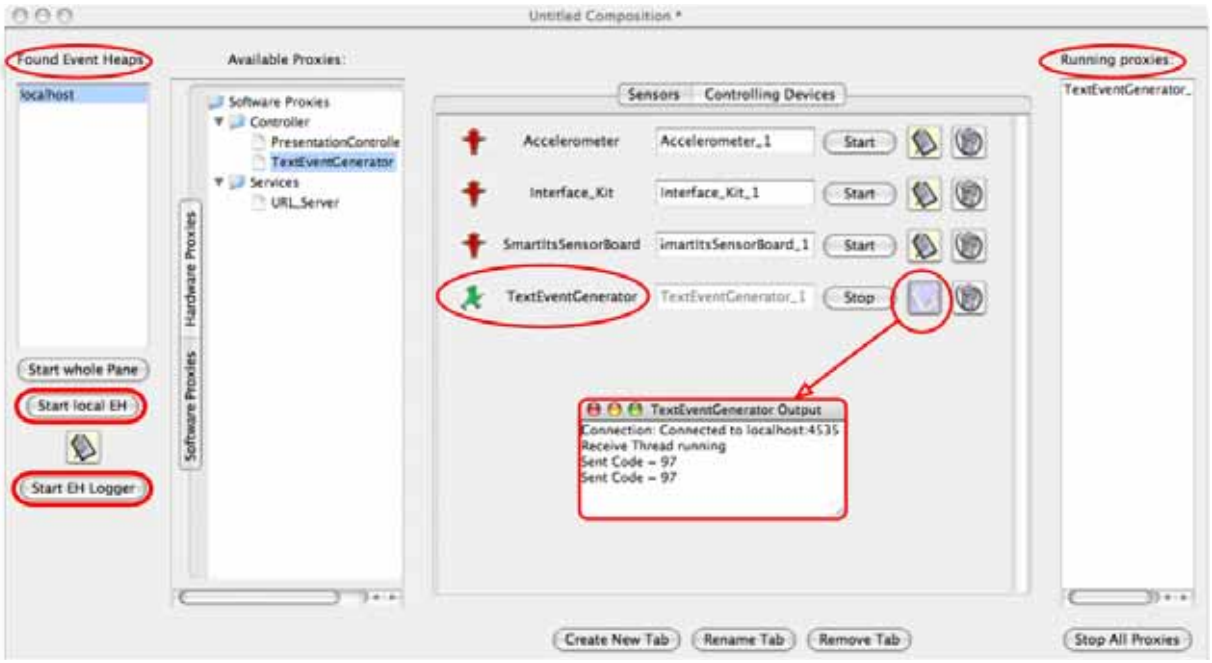


Figure 3.5: A screenshot of the ProxyManager application. Proxies can be arranged on different tabs (middle). On the left the discovered Event Heaps are displayed as well as buttons for launching a local Event Heap and the Event Logger, respectively. In the middle, a hierarchical tree is used to browse available proxies. A workspace shows the current proxies of interest selected by the user and their status. On the right side, the currently running proxies are shown. In the displayed situation, a "TextEventGenerator" proxy is running, indicated by the "walking man" icon.

Layer Summary

The Proxy Layer provides the glue that allows a diverse set of hardware and software components to communicate over the same network substrate, discussed in the subsequent Network Layer section. Examples demonstrate how the proxy strategy can be applied to support a range of different physical and graphical user interface elements. The Proxy Manager helps designers administer the active proxies running on specific machines that communicate over a distributed setting through the Network Layer.

Proxy Layer is the glue for the Network Layer

3.2.3 Network Layer

The network layer serves as a medium for the distributed components to communicate. It is abstracted as its own layer because it serves to simplify network communications, providing a high-level message-passing abstraction that makes networking accessible to designers in rapid prototoyping scenerios.

The Event Heap

The Event Heap (Johanson and Fox, 2002) is the central coordination mechanism used in the iROS ubicomp software framework. The Event Heap is partly modeled after the Linda tuplespace (Leichter and Whiteside, 1989), which provides a repository where tuples can be placed and then later retrieved by others. The tuplespace model is easy to grasp, and serves as a useful abstraction layer as it decouples readers and writers in both time and space. This communication model enables inter-application coordination in the iRoom interactive workspace (Johanson *et al.*, 2002a).

Tuplespace decouples communication in both time and space

Event Heap clients communicate over TCP/IP networks through the Event Heap's tuplespace abstraction using event publish/subscribe APIs that hide the underlying networking details. Similar to tuples, events consist of an unordered set of human-readable attributed-value pairs. The set of fields include mandatory fields, such as `EventType` and `TimeToLive`, and an arbitrary number of optional application-defined fields. The basic operations allowed of the client are `put`, to post an event, and `get`, which queries (either blocking or non-blocking) for the existence of an event based on a template that specifies required fields and constraints on their values. Clients may alternatively choose to receive notification of events through a callback function. It is possible to remove events from the Event Heap upon retrieval, but the removal is executed after all registered event subscriptions have been notified, preventing any race conditions for existing subscriptions. Events that are not marked as deleted are placed in an event store and are available for retrieval until their `TimeToLive` expires. Sequence numbers and time stamps are used to prevent duplicate delivery of events to the same client.

Simplified APIs hide underlying networking details

```
import iwork.eheap2.*;
class SpeakTextSender {                    // Connects to event heap in
    static void main(String []args)        // arg[0], and sends AudioEvent
    {                                      // with the text in arg[1].
        try{
            EventHeap theHeap = new EventHeap(args[0]);   // Connect to the Event Heap
            Event myEvent = new Event("AudioEvent");      // Create an event
            myEvent.setPostValue("AudioCommand", "Read"); // Set its fields
            myEvent.setPostValue("Text", args[1]);
            theHeap.putEvent(myEvent);                    // Put event into Event Heap
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figure 3.6: Sample Java code for posting an event on the Event Heap. The event is of type *AudioEvent* and has fields `AudioCommand` and `Text`, which are parameters for the SpeakText receiver. This program can be executed with the following command:
`$> java SpeakTextSender eh1.informatik.rwth-aachen.de "Hello World"`

```
import iwork.eheap2.*;
class SpeakTextReceiver{
  public static void main( String argv[]){
      try{
          // Connect to the Event Heap
          EventHeap theHeap = new EventHeap(argv[0]);

          // Create an event template
          Event template=new Event("AudioEvent");
          // specifying exact field values requires the field match both
          // field name and value of the template
          template.setPostValue("AudioCommand", "Read");
          // wildcard fields can be specified as follows,
          // where only the field name and type must be matched, not the value
          template.addField("Text", String.class,
                            FieldValueTypes.FORMAL,
                            FieldValueTypes.FORMAL);

          // get event from Event Heap
          retEvent = theHeap.waitForEvent(templates, this);

          // handle the text to speech conversion here
          System.out.println("Speaking text " + retEvent.getPostValueString("Text"))
        } catch(Exception e) {
           e.printStackTrace();
        }
  }
}
```

Figure 3.7: Sample Java code for receiving an event from the Event Heap using a blocking method. This program can be executed with the following command:
`$> java SpeakTextReceiver eh1.informatik.rwth-aachen.de`

```
import iwork.eheap2.*;
class SpeakTextReceiver implements EventCallback{
  public SpeakTextReceiver(String ServerName){
      try{
          // Connect to the Event Heap
          EventHeap theHeap = new EventHeap(ServerName);

          // Create the template
          template=new Event("AudioEvent");
          // specifying exact field values requires the field match both
          // field name and value of the template
          template.setPostValue("AudioCommand", "Read");
          // wildcard fields can be specified as follows,
          // where only the field name and type must be matched, not the value
          template.addField("Text", String.class,
                            FieldValueTypes.FORMAL,
                            FieldValueTypes.FORMAL);

          // create a list of event templates
          Event templateList[]=new Event[1];
          templateList[0] = template

          // register the return event method of this class as the callback
          theHeap.registerForEvents(templateList, this);
        } catch(Exception e) {
           e.printStackTrace();
        }
  }

  public boolean returnEvent(Event[] retEvents){
    try{
          // handle the speech to text conversion here
          System.out.println("Speaking text " + retEvents[0].getPostValueString("Text"))
    } catch(Exception e){
           e.printStackTrace();
    }
    return true;
  }

  public static void main( String argv[]){
        // Connects to event heap in argv[0]
        SpeakTextReceiver s = new SpeakTextReceiver(argv[0]);
  }
}
```

Figure 3.8: Sample Java code for receiving an event from the Event Heap using a non-blocking callback. This program can be executed with the following command: `$> java SpeakTextReceiver eh1.informatik.rwth-aachen.de`

Decoupling promotes robustness

The Event Heap promotes an anonymous communication model where as long as two applications understand the same event types, they are able to communicate without an explicit rendezvous. Failure isolation and robustness are promoted under this model since senders and receivers are decoupled over both time and space. The level of indirection between senders and receivers also provides the foundation for intermediation, but the Event Heap does not directly provide facilities for expressing these intermediations. A drawback of the event-driven anonymous coordination is that it does not support end-to-end delivery semantics of messages, since the sender does not know in advance who the receiver(s) will be or whether there will be any at all.

Infrastructure should not exceed physical boundaries of space

The Boundary Principle (Kindberg and Fox, 2002) states that the scope of ubiquitous computing infrastructure should not exceed the perceived physical boundaries of a space. Thus, there is a single Event Heap serving as the locus of interaction for each ubiquitous computing environment. A service or device must be able to communicate with the Event Heap to participate in the environment. Multi-cast DNS, a network discovery mechanism, is used to allow mobile clients to automatically connect to the appropriate Event Heap as they move from space to space.

Many other languages supported

Various libraries and other software components also allow Event Heap clients to be written in Java, C/C++, AppleScript, Visual Basic, Perl, Python, and other languages, creating support for heterogeneous machines and operating systems; servlets allow Web-based clients to perform limited Event Heap operations as well.

Example: Java

Case Study – Java: The Event Heap implementation in Java allows the senders and receivers to `put` and `get` events with only a few lines of code (see Figures 3.6, 3.7, and 3.8).

Debugging the Event Heap

Human readable events simplify debugging

One strength of using a tuplespace abstraction is that the fields and values are human readable, simplifying debugging. The Event Logger GUI[6] (see Figures 3.9, and 3.10) allows a developer to monitor and influence Event Heap activity.

Layer Summary

Common substrate, but different formats

The Network Layer abstracts low-level networking details to simplify prototyping distributed application scenarios. In the iStuff Architecture, the Event Heap's tuplespace model provides a simplified API to promote rapid prototyping, decouples communication to provide more robustness, and

[6] The Event Logger GUI was created by Andy Szybalski as a part of the open source iROS project (`http://iros.sourceforge.net`).

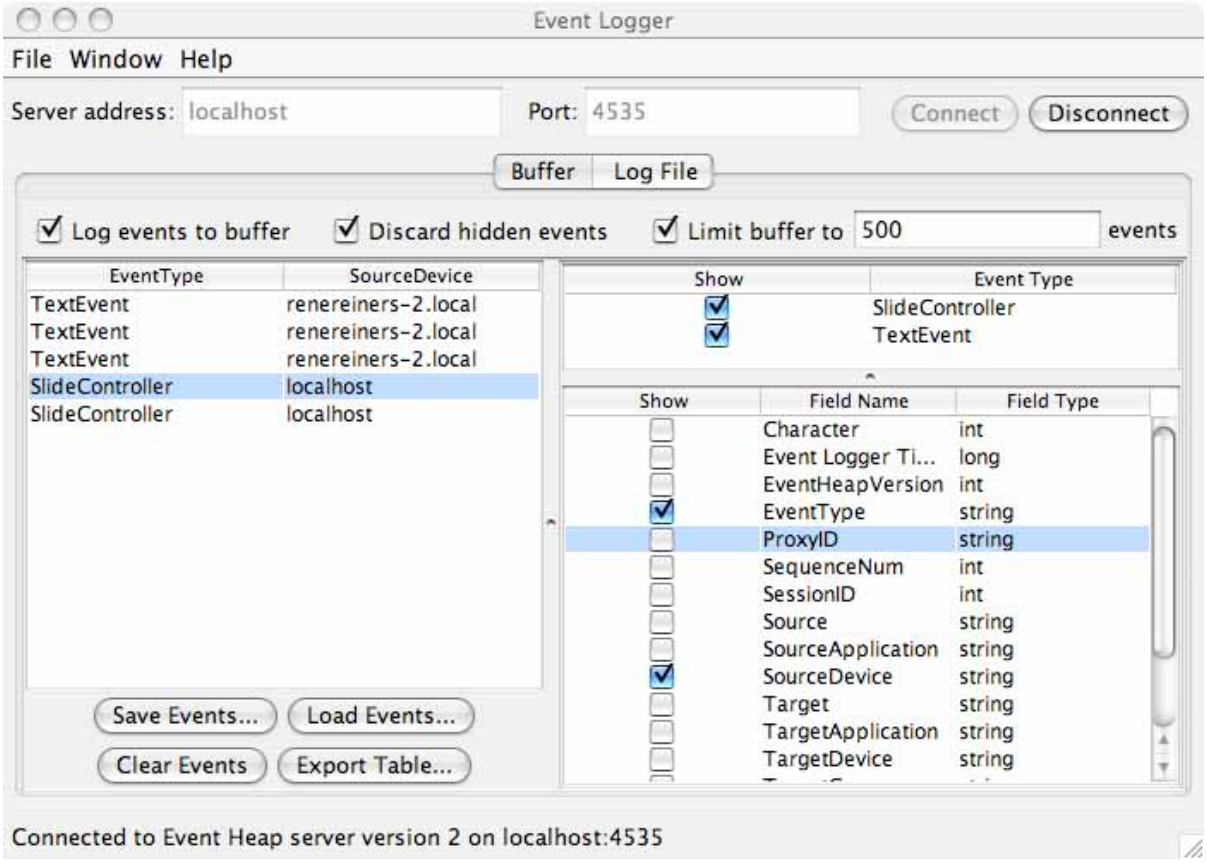


Figure 3.9: The Event Logger GUI simplifies debugging and monitoring Event Heap Activity. The top panel controls the connectivity to the Event Heap, the right panels control the event-level and field-level filtering desired for the log view (left panel).

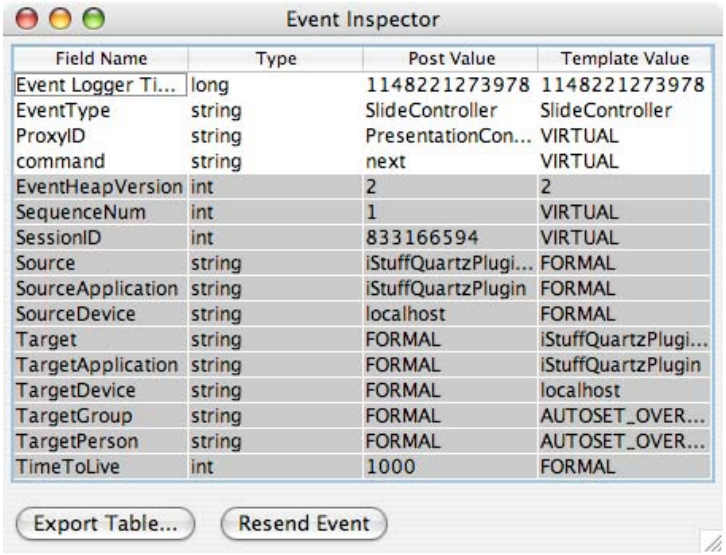


Figure 3.10: Selecting a particular event from the Event Logger displays all the fields and values of the event. The mandatory fields required by the Event Heap are visually separated using grey to assist the user in identifying the custom fields specified by the client application.

simplifies debugging through its human readable events. Although the UI components are all communicating over the same substrate, they are sending events in unique and incompatible formats. This last interoperability challenge is addressed in the next level, the Mediator Layer.

3.2.4 Mediator Layer

Mediators encapsulate how objects interact

The primary role of the Mediator Layer is to support interoperation between heterogeneous components. Gamma *et al.* (1995) describe a mediator as "an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently." In the iStuff architecture, this role is fulfilled by the Patch Panel infrastructure (Ballagas *et al.*, 2004).

However, to support incremental integration and dynamic composition, the interoperability relationships must support late binding (i.e., be run-time configurable).

Other System Approaches to Interoperability

A variety of approaches to interoperability exist

Many have argued for the use of service-oriented architectures, such as CORBA (Object Management Group, 2004), to address the interoperability challenges in pervasive computing. In these architectures, application-level interoperability is achieved through well-defined abstract interfaces to distributed software components that are accessed through RPC (remote procedure calls). Service-oriented architectures often support reflection allowing a service to reveal its service definition at run-time. An agent can use the service definition to support run-time decisions on how the services should interoperate. Message-oriented publish / subscribe systems such as Gryphon (Sturman *et al.*, 1998) offer brokering services to transform messages from one format to another. These systems also typically employ reflection to support run-time bindings between services. However, Johanson and Fox (2004) argue that a tuplespace communications model is better suited for room-sized environments, such as Interactive Workspaces, which is the domain the Patch Panel is designed to serve. Compared to RPC, the tuplespace model has better temporal decoupling, extensibility, expressiveness, a simple portable client API, easy debugging, fault tolerance and recovery mechanisms. Compared to message-oriented systems, the tuplespace model has better expressiveness, and limited temporal decoupling.

Automatic reasoning can be problematic for HCI

Agent-based architectures, such as the brokers in CORBA, automatically reason about interoperability relationships. Ontologies, such as SOUPA (Chen *et al.*, 2004), can facilitate automatic reasoning by defining a vocabulary to structure data shared between services or devices. However, automatic reasoning is not well suited to determine the appropriate mappings for human-computer interaction. Consider the exercise of mapping a

device to control the lights in the room. An agent will be able to identify a number of devices that can express the semantics of *on* and *off*, but it lacks human design insight and creativity to select the most appropriate device. Instead, ontologies and automatic reasoning can be used to support the human mapping process by automatically generating a list of mappings from which the designer can select the most appropriate mapping. This use of ontologies is an opportunity for future extensions of the Patch Panel.

One limitation of many service-oriented architectures, such as CORBA, and message-oriented publish/subscribe systems, such as Gryphon, is that they lack the ability to express *patterns of events*. Pietzuch *et al.* (2003) introduced a framework to express event composition at run-time in publish/subscribe systems using finite state automata. However, the relatively slow performance (approx. 10x slower than the Patch Panel) limits the system's usefulness for human-computer interaction. The Patch Panel extends this event composition work to show that finite state automata can also be used for modifying interoperability relationships (e.g., the input focus) of components in a room.

Patterns of events can be expressed using FSMs

Data-flow interoperation resolves mismatches in data type and has received much attention in the mobile computing literature (Kindberg and Fox, 2002). For example, a user may wish to connect a camera producing images in JPEG format to a printer accepting data in PostScript format. Speakeasy (Edwards *et al.*, 2002) provides support for datatype conversion by exploiting mobile code "proxies." However, datatype conversion is insufficient to enable interoperation because it neglects metadata and commands, which we refer to as *control-flow interoperability*. Continuing the printer example, the user may want to specify color vs. black-and-white printing, or the order in which pages will print. The communication of metadata and commands between the camera and the printer will vary depending on the the specific devices and their capabilities.

Control-flow interoperability addresses commands and metadata

A common approach to control-flow interoperability is interface standardization, as attempted by Sun's Jini (Waldo, 2000). Each *class* of service or device, such as printers, must adhere to a set of standard interfaces, enabling substitution by any other service or device in that class. The standardization process turns out to be surprisingly difficult, even for such a seemingly simple class of devices as printers. After more than 7 years in development, the Jini specification for printers is still (as of March 2007) labeled "draft"[7], meaning that "the implementation may necessitate changing the API before finalizing it as a full standard." One may infer that in fact the definition of a printer is a moving target since the feature set keeps changing. As new services or features are created, they cannot be exploited by applications until their integration into existing programmatic interfaces has passed the standards-approval process, potentially leading to unacceptable delay in adoption of the new service or feature.

Standardization is surprisingly difficult

Alternatively, mappings can be set through direct user intervention via GUIs as in Speakeasy (Edwards *et al.*, 2002) and iCrafter (Ponnekanti *et al.*,

[7] `http://www.jini.org/files/specs/print-api/index.html`

Direct user intervention is often not fluid enough

2001). Each time the user wishes to print a picture, a specialized GUI allows setting printing preferences for that specific printer model at print time. This approach limits interactions to "transactions" which are often not fluid enough for ubicomp. It also does not support making pre-defined rules for connections to reduce user intervention, for example, setting sensor data from a motion detector to turn on the lights.

Intermediation by rewriting event streams

The Patch Panel uses intermediation to achieve control-flow interoperability. This approach utilizes a decoupled communication model, such as event publish/subscribe, for inter-component communication. Implicit in the model is the ability to intercept and rewrite these event streams to convert the event (or interface) of the user input into another event (or interface) that clients producing user feedback expect (see the Adapter pattern from (Gamma *et al.*, 1995)). Other systems have proposed using intermediation for interoperation. Taylor's C2 architecture (Taylor *et al.*, 1996) uses connectors to perform intermediation on messages between heterogeneous components. Bates *et al.* (1998) proposes event-based rules to serve as integration glue to federate heterogeneous components. Munson (1998) also proposes using intermediation to support composition of distributed applications. The Patch Panel builds on these ideas by adding and exploiting dynamic reconfiguration capabilities.

Functional Details of the Patch Panel

The Patch Panel translates events to allow incompatible event publishers and subscribers to communicate. Simple intermediations are expressed as mappings that connect *triggers* (event templates that specify the conditions for intermediation to occur) with *outputs* (new events emitted as a result of the trigger, presumably for consumption by a different entity). The Patch Panel prototype is implemented as an Event Heap client that works by subscribing to all events. The Patch Panel internally manages a data structure to quickly compare each incoming event against all active triggers. When an event is received that matches one or more trigger conditions, the Patch Panel generates the corresponding superset of output event(s) and posts them to the Event Heap before processing the next incoming event.

A mapping represents the function of the Patch Panel

A *mapping* (trigger → output event(s)) is the basic abstraction provided by the Patch Panel. For example as in Fig. 3.11, suppose we have a wireless iStuff button (Ballagas *et al.*, 2003) that posts a *Button* event[8] each time it is pressed with string-valued field `id`, a light dimmer service that responds to *Lights* events containing an integer-valued field `brightness` between 0 and 10, and a projector-control service that responds to events *Projector* with boolean-valued field `powerOn`. We can configure the button to turn the lights and projector on by establishing the following Patch Panel mapping:

Button(`id=red`) → *Lights*(`brightness = 10`), *Projector*(`powerOn=true`)

[8] The stylized event names such as *Button* and *Lights* represent the mandatory `EventType` field of the events.

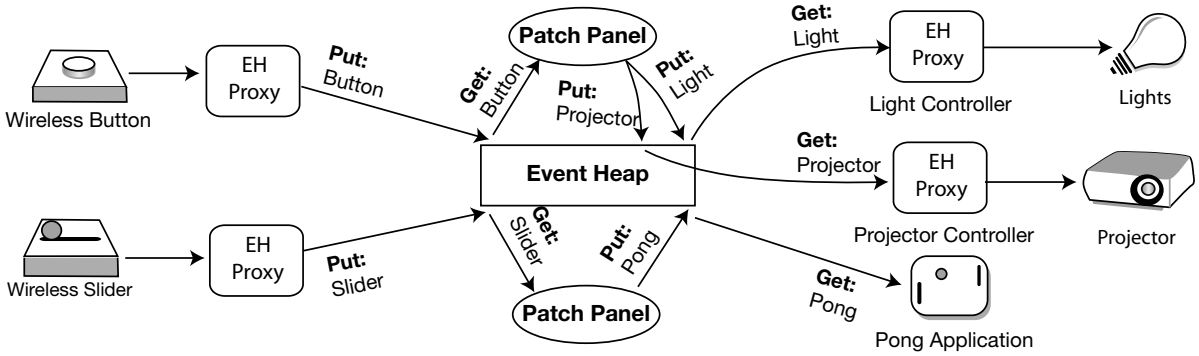


Figure 3.11: The Patch Panel adds a level of indirection to the communication channel between two components to perform event intermediation. The publish/subscribe semantics are also demonstrated. Note that there is actually only one Patch Panel process per Event Heap; two are shown above for visual clarity.

Now, when someone presses the button, the following sequence of operations occurs:

1. In response to being pressed, the physical button transmits a wireless signal that is received by the button's Event Heap proxy, which in turn posts a *Button*(`id=red`) event to the Event Heap.

   Input events rewritten

2. Although neither the light service nor the projector service recognize the *Button* event, the Patch Panel recognizes it as a trigger for the *Button* mapping.

3. The mapping fires, causing the Patch Panel to post the events *Lights*(`brightness=10`) and *Projector*(`powerOn=true`) to the Event Heap.

4. The light controller recognizes the *Lights* event and turns the lights on; the projector controller recognizes the *Projector* event and turns the projector on.

Event Heap clients can query, add, remove, and modify the mappings in the Patch Panel using event-based remote method invocation (RMI). The Patch Panel listens for an application-specific *PPMapping* event type. The event includes a `Method` field which describes the operation to perform, and a `Mapping` field which contains an XML-style description of the triggers and outputs. The Patch Panel handles the request and returns a *PPResponse* event indicating the result of the desired operation. This event also contains a field identifying the process that requested the operation, as well as the details of the method performed. By using the Event Heap as the communications substrate for RMI, the processes that affect mappings do not need to care about the "address" of the Patch Panel as they move from space to space. We have provided programmer-friendly client libraries that abstract from the details of the event-based RMI so that the programmer can modify mappings as if it were a local function call as shown in Fig. 3.12. This event-based interface design also has significant implications on the

RMI used to set mappings at run-time

```
// connect to a specific EH
EventHeap eh = new EventHeap(eheap.rwth-aachen.de);
PatchPanel pp = new PatchPanel(eh);

 // specify the template for the trigger
Event trigger = new Event("Button");
trigger.addPostValueString("id", "red");

// create the desired output events
Event[] outputs = new Event[2];
outputs[0] = new Event("Lights");
outputs[0].addPostValue("brightness", new Integer(10));
outputs[1] = new Event("Projector");
outputs[1].addPostValue("powerOn", new Boolean(true));

// commit the mapping to the Patch Panel
Mapping myMapping = new Mapping(trigger, outputs);
try{
     pp.setMapping(myMapping);
} catch(PatchPanelException ppe){
     System.out.println("RMI timeout");
}
                                                    // test the mapping
eh.putEvent(trigger);                               // put Button
Event received = eh.waitForEvent(outputs[0]);       // get Lights
```

Figure 3.12: Sample Java code for setting the *Button* → *Lights*, *Projector* mapping.

capabilities of the Patch Panel because event triggers can be mapped to generate *PPMapping* events as outputs, which in turn modify the active mappings. This idea of mappings that have the ability to change mappings when fired is the key feature that enables dynamic reconfiguration and differentiates the Patch Panel from other systems that use intermediation, as we will discuss further in later sections.

A range of interoperability problems

In order for the Patch Panel to serve as an viable strategy for control-flow interoperability, it must provide richer mapping capabilities than the basic button scenario shown in Fig. 3.11. Sheth (1999) classifies interoperability problems as System, Syntactic, Structural, and Semantic. System level interoperability corresponds to heterogeneous hardware and operating systems and is resolved by using the Event Heap. The remaining issues can be described as follows:

- **Syntactic differences between field values:** fields from different event formats may choose to represent the same information in incompatible formats (such as different data types).
- **Structural differences in the event formats:** the same data may be represented in a different event structure, for example, with different field names.
- **Semantic differences at the field level:** the intended meanings of fields do not directly correspond, but the mapping is such that the

value of one field affects the value of the other (such as absolute vs. relative values, or different units).

- **Semantic differences at the event level:** the desired outputs do not necessarily correspond to a single input trigger, and instead may be the result of many different input events, or if the correct mapping changes due to the context of interaction such as time passing.

Output events often depend on input events

We have built facilities into the Patch Panel to address each of these interoperability issues. First of all, the Patch Panel supports a field-level equation specification in order to support translation dependencies, where field values of the output events are not known at the time the mapping is specified and instead must be determined based on the trigger at run-time. Consider the following mapping from a simple slider whose value is a float that ranges from 0 to 1 that is intended to control a paddle in a multi-screen version of the classic Pong arcade game. The *Pong* event template expects a `Side` field which identifies which paddle to control (left or right) and a `Position` field that accepts integer values from 0 to 100:

Slider(`value=FORMAL`) → *Pong*(`Side=left, Pos=(int)(in.value*100)`)

A range of operators are supported for equation specification

The `FORMAL` value is used in the Event Heap to indicate that any value is acceptable for template matching as long as the field is present. In this instance, the equation directs the Patch Panel to perform an affine transformation of the `value` field of the input event (specified by the `in.` prefix) and then convert the result to an integer. The Patch Panel equations support all basic mathematical operators ($*, \%, +, -$, etc.), logical operators (AND, OR, NOT, $>, <, <=$, etc.), and string concatenation. Additionally, global variables (set via event RMI and referenced using a `global.` prefix) are also provided to preserve values across individual firings of a trigger. Global variables are especially useful when converting between absolute values (such as screen position) and relative values (such as delta X). Equations can be used to resolve field-level syntactic mismatches, structural mismatches, as well as perform operations to resolve field-level semantic differences.

Patch Panel mappings can be used to implement FSMs

In order to address the interoperability challenge of semantic differences at the event level, the Patch Panel employs finite state machines (FSMs). Going back to the wireless button example in Fig. 3.11, suppose instead we want the red button to *toggle* the lights and projector between on and off. Assuming (as is often the case) that the light and projector service cannot be "queried" about their current state (on or off), we could turn the button into a toggle switch using an FSM as in Fig. 3.13. Although the button is stateless, and the light and projector do not expose their internal state, the Patch Panel uses mappings to instantiate an FSM that "remembers" the current state of the interaction and responds accordingly to the *Button*(`id=red`) trigger event. The actual state transitions are performed using *PPMapping* events as a part of the mapping output to modify the active mappings.

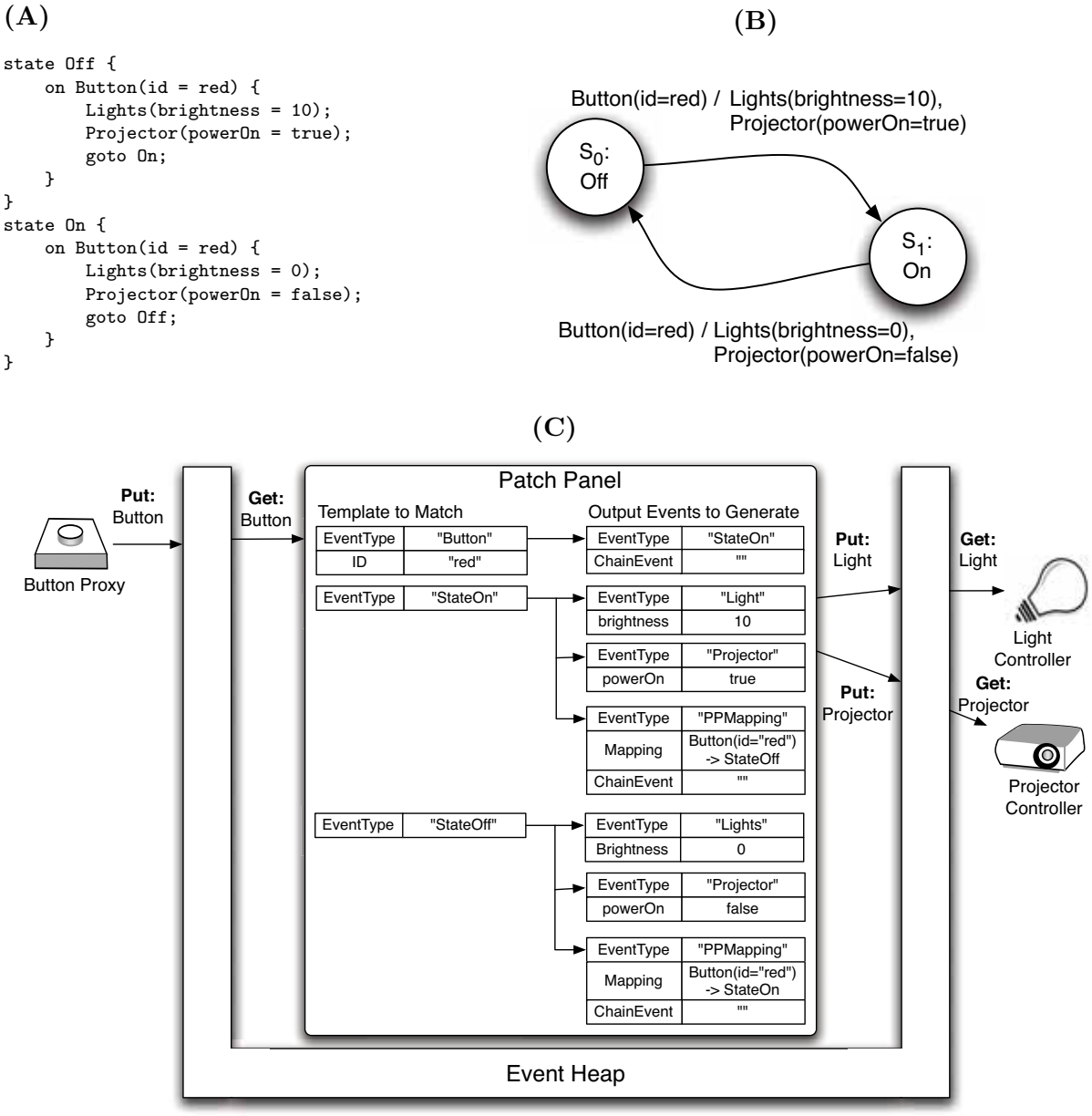


Figure 3.13: (A) A textual description of the Mealy state machine diagram for a light toggle in (B). Circles labeled Si are states. Each edge is labeled with "x / y" where x is the input and y is the output. (C) The Patch Panel mappings that implement the state machine.

To formally illustrate that this can be done for any FSM, consider a more complex example shown in Fig. 3.14, which adds support for a motion sensor that posts a *MotionSensor* event whenever it detects activity. If no motion is detected for `timeout` seconds, the lights and projector turn off automatically.

Patch Panel operation can be formalized

Using Fig. 3.14, we can formalize the Patch Panel's operation as follows. Let $S_0, \ldots, S_{n-1}$ be the n states of an FSM that expresses a Patch Panel-mediated interaction. Going back to Fig. 3.14, let S_0 be the Off state and S_1 the On state. Let m_i be a simple event mapping of the form $t_i \rightarrow$

(A)

```
state Off {
    on Button(id = red) {             // turn things on manually
        Lights(brightness = 10);
        Projector(powerOn = true);
        goto On;
    }
}
state On {
    on MotionSensor {                 // motion detected, so...
        set timer 1000*timeout;       //   ...reset idle timer
        goto On;
    }
    on timer {                        // no motion sensed, so...
        Lights(brightness = 0);       //    ...turn things off
        Projector(powerOn = false);
        goto Off;
    }
    on Button(id = red) {             // turn things off manually
        Lights(brightness = 0);
        Projector(powerOn = false);
        goto Off;
    }
}
```

(B)

m_0 = *Button*(`id=red`) → *Lights*(`brightness=10`), *Projector*(`powerOn=true`) NS(0) = S_1
m_1 = *MotionSensor* → set timer 1000*timeout NS(1) = S_1
m_2 = timer → *Lights*(`brightness=0`), *Projector*(`powerOn=false`) NS(2) = S_0
m_3 = *Button*(`id=red`) → *Lights*(`brightness=0`), *Projector*(`powerOn=false`) NS(3) = S_0

Figure 3.14: (A) FSM description with timers. (B) Formal notation for the mappings in A.

$o_{i1}, o_{i2}, \ldots o_{ik}$, where t_i is a trigger event and $o_{i1} \ldots o_{ik}$ are the output events triggered by t_i. Finally, let $NS(i)$ be the number of the next state to go to after emitting the output events of mapping m_i. Each of the "on" clauses generates one mapping m_i and one next-state $NS(i)$.

Let M_i be the set of mappings consistent with the FSM being in state S_i. For Fig. 3.14, we would have $M_0 = \{m_0\}$ and $M_1 = \{m_1, m_2, m_3\}$. The Patch Panel maintains a single set P of mappings that is currently active, i.e., against which incoming events will be checked for triggers. When a state transition occurs into state S_i, P must be set to M_i. With this notation, we can express in pseudocode the operation of the Patch Panel (by convention, S_0 is the initial state):

1. Compute all m_i, $NS(i)$, and sets $M_0 \ldots M_{n-1}$ from textual FSM description;
2. Set P to M_0;
3. do forever:

(a) wait for an event t_i that triggers some mapping m_i in P;
(b) emit the mapping's output events $o_{i1}, \ldots, o_{ik}$;
(c) set P to $M_{NS(i)}$;

In other words, the Patch Panel has an FSM "compiler" that takes the textual descriptions of the FSMs as input and produces the mappings to implement the FSM as output.

Chain events ensure atomic execution of state transitions

By design, the Event Heap does not guarantee ordering of events from different sources. Therefore, output events from the Patch Panel and trigger events from other sources may be interleaved. This is problematic since the FSM abstraction requires that emitting output events (step 3b above) and transitioning to the next state (step 3c) must occur atomically. To address this, the Patch Panel provides *event chains*, groupings of events that must be processed atomically. In our current implementation, any outgoing event containing a field `ChainEvent` is considered to be part of such a chain. The Patch Panel's event emitter handles event chains internally by directly passing them to its own event handling loop (bypassing the Event Heap), where they are given highest priority. These events are also passed to the Event Heap for other applications to use, but any incoming events from the Event Heap with the `ChainEvent` field are thrown out to prevent duplicate processing. For example, in Fig. 3.13, when the Button is pressed, a *StateOn* event is created. Because of its `ChainEvent` field, it bypasses the Event Heap and is processed directly by the Patch Panel as the next incoming event. Subsequently, the same happens for the following *PPMapping* event. Therefore, the state transition is complete before the next incoming event from the Event Heap is processed.

Layer Summary

The Mediator Layer supports interoperation by encapsulating how objects interact. In the iStuff Toolkit architecture, the Patch Panel peforms this functionality by rewriting events. The basic unit of functionality is the mapping, which can be leveraged to express patterns of events using finite state machines. The Patch Panel mappings can be dynamically set at run-time through remote method invocation allowing designers to specify mappings using higher-level Application Layer interfaces. This section has partly introduced one of these interfaces, the Patch Panel scripting language, which will be elaborated on further in the following section along with other graphical interfaces to configure mappings.

3.2.5 Application Layer

The application layer is where the interaction logic is expressed – defining how the user interface responds to certain inputs. This layer serves as

the interface to developers, allowing them to rapidly prototype different application designs.

Toolkit ceiling and threshold can be used for analysis

Myers *et al.* (2000) identify several characteristics, or "themes", for evaluating toolkits. In particular, the *threshold* describes how difficult it is to learn to use the system. The *ceiling* describes how much can be accomplished using the system.

Wide-range of UIs

We have developed several different user interfaces that allow end users to configure the active mappings of the Patch Panel. The range of UIs are geared towards different audiences and tasks, and each have different ceilings and thresholds for building applications. The first UI, partly introduced in the previous section, is a non-graphical text-based FSM scripting language.

Scripting Language

The scripting language enables programming patterns

The Patch Panel scripting language enables a repertoire of "programming patterns" needed to cover a broad variety of incremental-integration scenarios. In this section, we describe some of the patterns supported by the scripting language, motivating each one with a real-life example. As a preview, the mechanisms we will describe are as follows:

1. Allowing for *dependent translations*, where the field values of output events are not known at the time the mapping is specified. Instead, the output fields must be derived from field values of trigger events at run-time.
2. Programmatically exposing to applications the ability to change mappings on the fly, allowing the construction of GUIs and other applications whose function is to configure the Patch Panel itself.
3. Allowing the use of global variables (whose value persists across individual firings) to further support interactions requiring persistent state.
4. Providing an abstraction for time, allowing for time-based interactions

Set of examples based on the iClub

This set of Patch Panel scripting language examples is based on the iClub (Samberg *et al.*, 2002), an iRoom application that creates an interactive dance club environment using the large displays in the iRoom. The iClub was originally developed by a group of students in the Stanford iRoom as part of a student project.

Interactive Workspace turns night club

iClub is a distributed application that includes a playlist program to select songs and sound effects, an audio proxy application that plays the audio and publishes a "beat clock" event synchronized to each beat of the music,

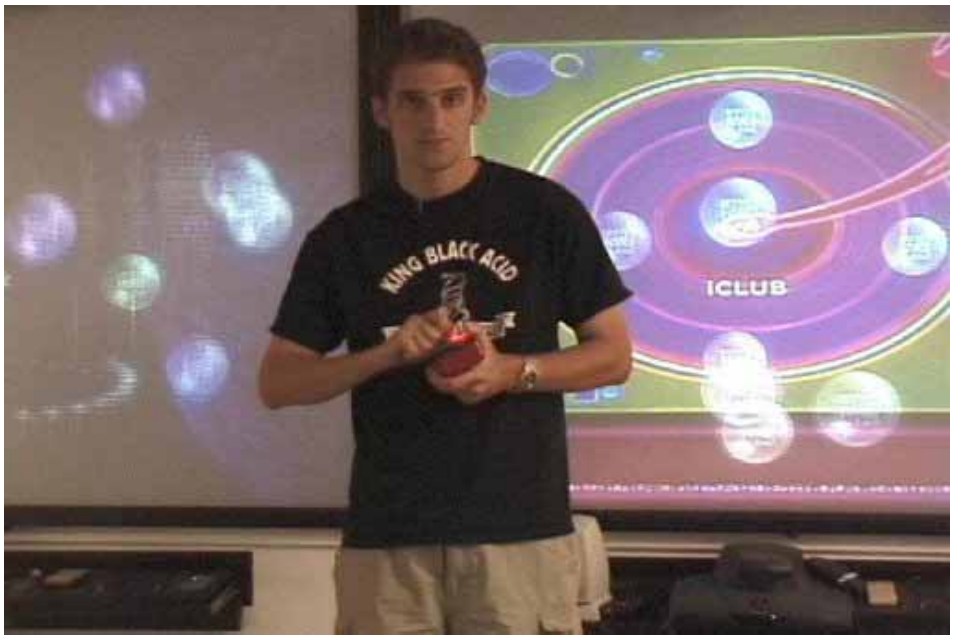


Figure 3.15: the iClub in Action with iSlider

several visualization applications that synchronize to the beat of the music by subscribing to the beat clock events, and a GUI that controls various aspects of the music as it plays.[9] We used the iStuff Toolkit architecture and Patch Panel scripting language to extend the iClub to achieve a range of new features as discussed in the following sections.

Connect a new physical UI to an existing behavior on-the-fly

Case Study – On-the-Fly Integration: The audio proxy recognizes events to adjust the volume and tempo of the music, inject sound effects, and apply high- and low-pass frequency filters (a common audio special-effect) to the music. In the original version of iClub, a human DJ would use a GUI to control these aspects of the music. However, an observer suggested that the clubgoers themselves should be able to participate in the music creation without leaving the dance floor. We arranged to give each clubgoer a wireless button with a unique ID as they entered the room. Each button was mapped through the Patch Panel to direct the audio proxy to play a sound, so each clubgoer had her own characteristic sound effect that could be injected by pressing her button. We refer to this common programming pattern—connecting a new physical or other UI to an existing behavior—as *on-the-fly integration.*

Perform affine transformations

Case Study – Range Normalization: The developers' next inspiration was to allow a DJ to use a wireless handheld slider to have mobile control the tempo of the current song. Conceptually, this is similar to the previous example in that a slider event must be used to trigger an iClubAudio output event, except that the *value* of the slider must also be reflected in the output event. Furthermore, the specific slider we used produces real numbers in the range 0.0 to 1.0, whereas the iClubAudio's `tempo` parameter must be an

[9] A video demonstrating the iClub is available at `http://iwork.stanford.edu/pubs/iclub-300mb.mov`

iSlider(`position = *`) → *iClubAudio*(`tempo = (int)in.position * 20 - 10`)

Figure 3.16: Range Normalization and Equation Specification

Button(`id=1`) → *PPMapping*[*iSlider*(`position=*`) → *iClubAudio*(`tempo=(int)in.position*20-10`)]
Button(`id=2`) → *PPMapping*[*iSlider*(`position=*`) → *iClubAudio*(`volume=(int)in.position*100`)]
Button(`id=3`) → *PPMapping*[*iSlider*(`position=*`) → *iClubAudio*(`highfreq=(int)in.position*100`)]
Button(`id=4`) → *PPMapping*[*iSlider*(`position=*`) → *iClubAudio*(`lowfreq=(int)in.position*100`)]

Figure 3.17: Patch Panel mappings that enable the multi-slider handheld music controller

integer from −10 to +10. We use the term *range normalization* to describe this pattern for incremental integration. The Patch Panel supports range normalization by allowing output event fields to reference input event fields and by providing a simple arithmetic expression evaluator. Figure 3.16 shows a mapping that connects the slider to the tempo control.

This mapping will fire when the Patch Panel receives an event of type *iSlider* that contains a field named `position`. The `tempo` field of the *iClubAudio* output event is computed from the `position` field of the input event (`in.`) each time the mapping fires. Similarly, the prefix `out.` could be used to reference other fields in the current output event; the Patch Panel automatically detects syntax errors or output field dependency loops and leaves such equations unresolved (in string form) for debugging purposes.

Physical actions that change the state of mappings

Case Study – Dynamic Reconfiguration: The wireless slider works well to control a single parameter, but in order for a DJ or clubgoer to abandon the desktop GUI completely, she must have the ability to change *any* of the music parameters, not just the tempo. We constructed a new device by physically attaching four wireless buttons (call them 1, 2, 3, 4) to the slider. Pressing a button determines which music parameter—tempo, volume, high pass filter, low pass filter—is controlled by the slider. This case is more subtle, because the effect of pressing a button is not to emit a new audio event, but rather to affect the handling of future iSlider events. In other words, pressing a button should change the currently-active iSlider mapping. We therefore refer to this pattern as *dynamic reconfiguration*.

As explained previously, the Patch Panel consumes events of type *PPMapping* to modify the set of currently active mappings. With this in mind, the mappings to implement the "multi-slider" handheld music controller are shown in Figure 3.17.

When the Patch Panel receives a *Button*`(id=1)` event, for example, it emits a *PPMapping* event that will set up a new mapping for future *iSlider* events to control the music's tempo. This powerful dynamic reconfiguration allows complex interactions to be synthesized without direct user interaction.

Since the mappings are somewhat difficult to interpret for a programmer unfamiliar with the Patch Panel, we express the multi-slider example using the FSM representation shown in Figure 3.18, and in fact, submitting this

```
state ControllingTempo {
  on iSlider(position=*) { send iClubAudio(tempo=int(in.position)*20-10); }
  on Button(id=2) { goto ControllingVolume; }
  on Button(id=3) { goto ControllingHighFreq; }
  on Button(id=4) { goto ControllingLowFreq; }
}
state ControllingVolume {
  on iSlider(position=*) { send iClubAudio(volume=int(in.position)*20-10); }
  on Button(id=1) { goto ControllingTempo; }
  on Button(id=3) { goto ControllingHighFreq; }
  on Button(id=4) { goto ControllingLowFreq; }
  }
}
...
```

Figure 3.18: FSM description of iClub multi-slider mappings. We have minimally stylized the code for space (Typically each event template must be described with necessary mandatory fields and values at the top of the script, instead of inline as shown above).

FSM description to the Patch Panel's FSM compiler would effectively result in the mappings in Figure 3.17.

Devices composed without *a priori* knowledge

Several points about this example should be emphasized. First, the handheld DJ device and the interaction with the iClub Audio Proxy were assembled from devices and services that had no *a priori* knowledge of each other. In other words, the slider has no concept of a button or vice versa, and the iClub Audio Proxy has no concept of any physical devices. Also in this example, the buttons and mappings, once initially set up by an administrator, represent a tangible UI for reconfiguring the Patch Panel that can be used by people who have no technical skill or knowledge of Event Heap operation.

Global variables store values that must persist across firings of mappings

Case Study – Semantic Mismatches and Globals: At one point, the wireless slider malfunctioned, and although we did not have another slider immediately available, we did have a joystick. However, while a slider's position naturally maps to the value being controlled, a joystick's position naturally maps to the rate of change of the value being controlled (since joysticks are self-zeroing). We refer to this circumstance as *semantic mismatch*, and in this case can be resolved by using an FSM with global variables as shown in Figure 3.19. Global variables store values that must persist across firings of mappings; variables can be set using "PPVariable" events and dereferenced on the output side of any mapping by using the prefix `global`. To mimic the slider's behavior with a joystick, we use a global variable to hold the current "slider position" and adjust the global variable's value each time the joystick is moved. To avoid wild fluctuation in the variable's value, we can use timers (as previously described) to control the interval at which the joystick is sampled; changing the timer value changes the sensitivity of the joystick as a controller. Although the joystick is an imperfect interaction modality for the iClub Audio Proxy, the Patch Panel made it possible to

```
state JoystickMoved {
   on Joystick(joystickX, joystickY) {
       global.currentX += in.joystickX * global.scaleFactor;
       global.currentY += in.joystickY * global.scaleFactor;
       send Position(global.currentX, global.currentY);
       set timer sampleRate;
       goto WaitingForSample;
   }
}
state WaitingForSample {
   on timer { goto JoystickMoved; }
}
```

Figure 3.19: Resolving semantic mismatch between relative-position and absolute-position devices. We have minimally stylized the code for space (Typically each event template must be described with necessary mandatory fields and values at the top of the script, instead of inline as shown above).

use it as an adequate substitute until the slider could be replaced.

Patch Panel Manager

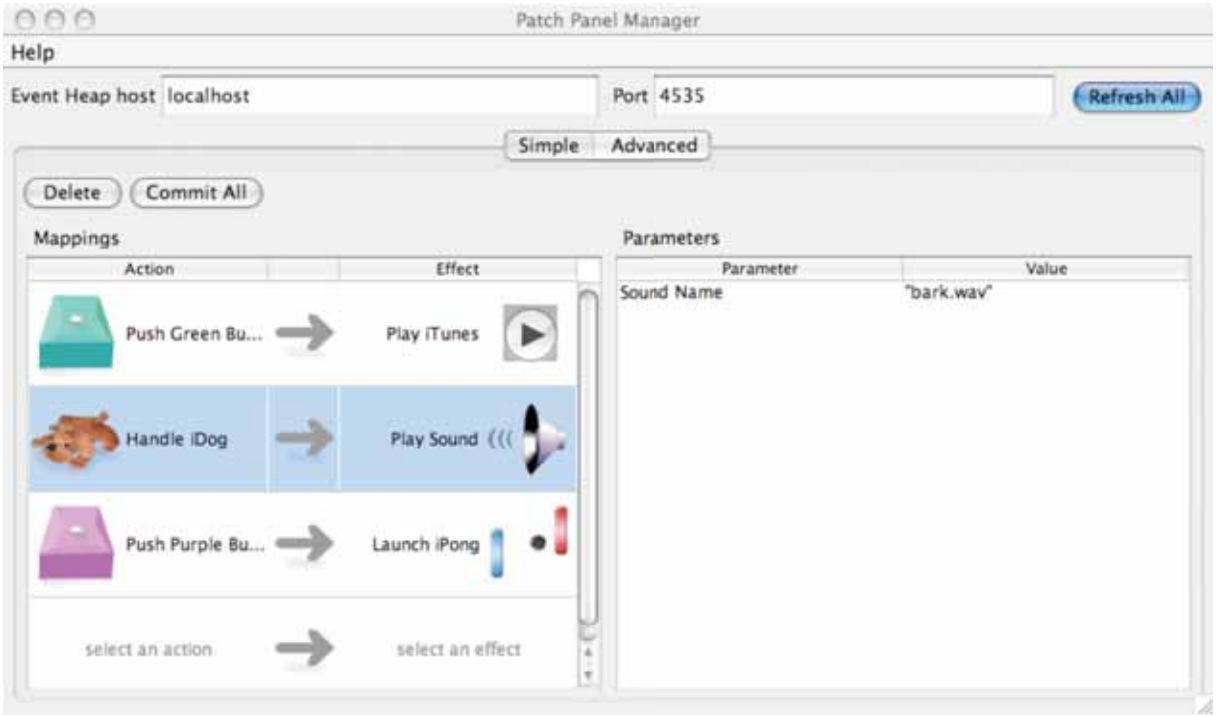
Transitioning from textual to graphical interfaces to control the mappings, the Patch Panel Manager is a general purpose GUI that exposes different granularities of Patch Panel functionality to the user. The Patch Panel Manager distinguishes between expert and casual users. Expert users include: developers who want to create or augment applications by connecting new behaviors via the Patch Panel; system administrators; and "power users" (akin to those who can write complex macros in spreadsheet programs) who want to modify the behavior of existing applications.

The simple UI has a low threshold, but also a low ceiling

For casual users, the "simple" pane of the Patch Panel Manager provides very basic interface provides access to a very limited set of simple mappings (see Figure 3.20 [Top]). The full details of the mapping are abstracted away through a configuration file that is modified by the room administrator. If a new device or service is added to the room, it can only be used through this interface after the room administrator has made the appropriate modifications to the configuration. This "simple" pane is considered to be a low threshold UI because of its simplicity, but also a low ceiling UI since the types of mappings that can be created are very limited.

The expert UI is high ceiling, but high threshold

Expert users can access the "advanced" pane of the Patch Panel Manager (see Figure 3.20 [Bottom]), a tree-based view of the Patch Panel mappings that allows graphically browsing of the current mappings and the creation of new mappings. Using this GUI requires some understanding of Event Heap semantics as well as the event interfaces of individual components. In everyday situations, the Patch Panel Manager is used to visualize and debug



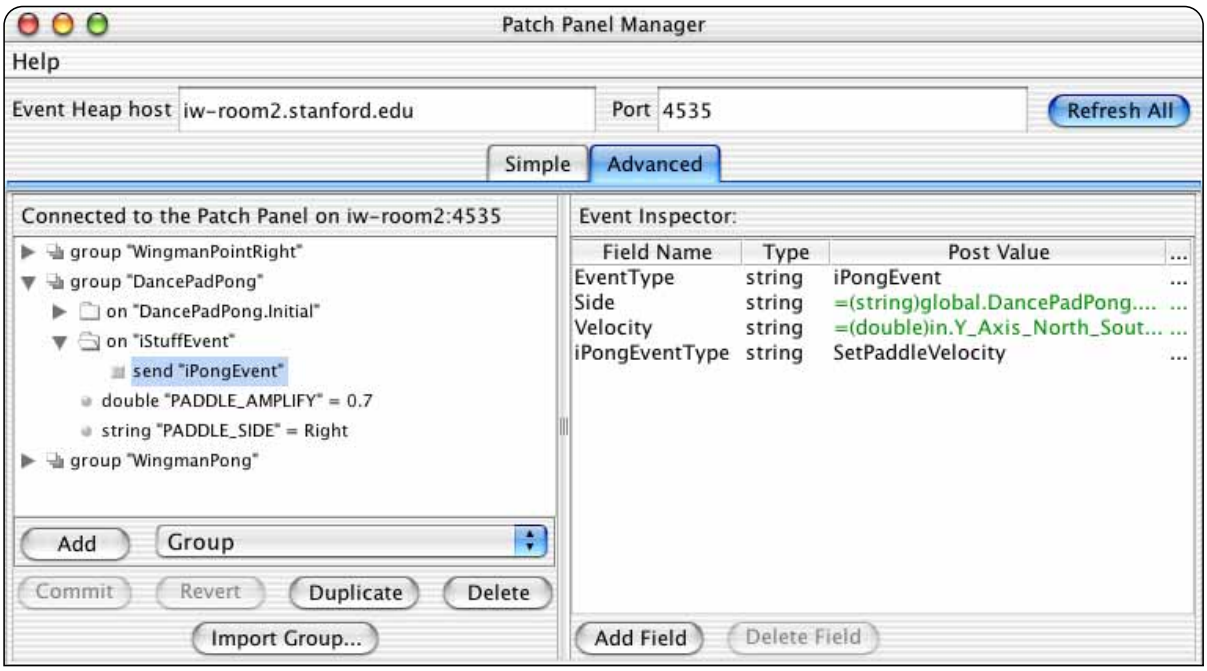


Figure 3.20: (Top) The simple panel of the patch panel GUI, intended for casual non-technical users, provides access to a very limited set of simple mappings. (Bottom) The advanced panel of the patch panel manager, intended for experts, supports direct browsing and editing of the Patch Panel mappings.

the state of the Patch Panel. It is also particularly useful for making on-the-fly modifications to mappings or for "debugging" in-progress mappings; for example, changing the amplification factor applied to an input device. This "advanced" pane is considered to be a very high ceiling interface since any mapping can be created, but is also a high threshold interface since it is very difficult to setup a series of interrelated mappings such as with a state machine.

Custom GUI - Workspace Navigator

Workspace Navigator (WSN) (Ionescu *et al.*, 2002) is an application that captures multi-person meetings in interactive rooms containing shared public displays. WSN's GUI provides a "bookmark" feature that allows a participant to flag an important moment in the meeting; WSN's meeting-replay tools can then be used later to reconstruct the state of the meeting (e.g., which documents were visible on each of the shared displays) at the time the bookmark was inserted. To implement the bookmark function, the WSN GUI console sends a *Bookmark* event to the WSN server application when the Bookmark GUI widget is clicked.

Meeting capture software

During user testing of WSN, one user complained that inserting a bookmark required disrupting the meeting to acquire the shared keyboard and mouse (in order to interact with the GUI), discouraging users from taking advantage of this feature. The researcher proposed giving each meeting participant a wireless button that could be discreetly pressed to add a bookmark during the meeting. This approach has the additional benefit that each bookmark could be associated with the participant who inserted it.

Need for efficient bookmarking

To implement this, an iRoom administrator created a simple Web-based "This Is My Button" wizard (see Figure 3.21) that configures an iButton (an iStuff wireless physical button) to send Bookmark events. On entry to the iRoom, a meeting participant, say Rachel, picks up an iButton from a bucket of buttons, enters her name into the Web form, and submits the form. The form submission runs a servlet that waits for the next button press from any iButton. Rachel now presses the (physical) button, causing the servlet to establish a Patch Panel mapping connecting her particular button to bookmark events with her name attached to them.

Simple web interface for mapping physical buttons as bookmark UI

This is another example of dynamic reconfiguration. The entire process of integrating the wireless button interaction into the WSN project only took about an hour and did not require any changes to the Workspace Navigator code or the wireless buttons.

Quartz Composer

Quartz Composer is a visual programming environment (see Figure 3.22) that is part of Apple's freely available Xcode development environment.

Button(`id=*`) → PPMapping[Button(`id=in.id`) → Bookmark(`username="Rachel"`)]

Figure 3.21: (Top) Sequence of screens that illustrate the functionality of the Button-to-Bookmark Configuration Servlet. (Bottom) The equivalent function of the GUI specified in the mapping notation.

Uses a pipe-and-filter metaphor

It was introduced with Mac OS X 10.4 "Tiger". It uses a *pipe-and-filter* metaphor (originally from software architecture, popularized by the UNIX operating system) to establish data and control flow between different components, establishing a composition. Advantages of the pipe-and-filter metaphor include (Bass *et al.*, 1998):

- Filters stand alone and can be treated as black boxes ensuring information hiding, high cohesion, modifiability, and reuse.
- Filters interact with other components in limited ways. This connection simplicity helps to ensure low coupling.
- Pipes and filters can be hierarchically composed allowing higher order filters to be created from any combination of lower order pipes and filters.
- The construction of the pipe and filter sequence can often be delayed until runtime (late binding).
- Parallelism is encouraged because the process performed by the filter is isolated from the other components in the system.

The editor is live, and changes made in the workspace are immediately functional without any compilation steps. We have extended the Quartz Composer environment to enable prototyping of physical interfaces. We added library components for each of the iStuff proxies and new data processing modules that are particularly useful in physical prototyping scenarios.[10]

[10] Update: In response to requests from the developer community, including our own initiative (`www.qcplugins.com`), Apple decided to officially open its Quartz Composer architecture to third-party plugin development.

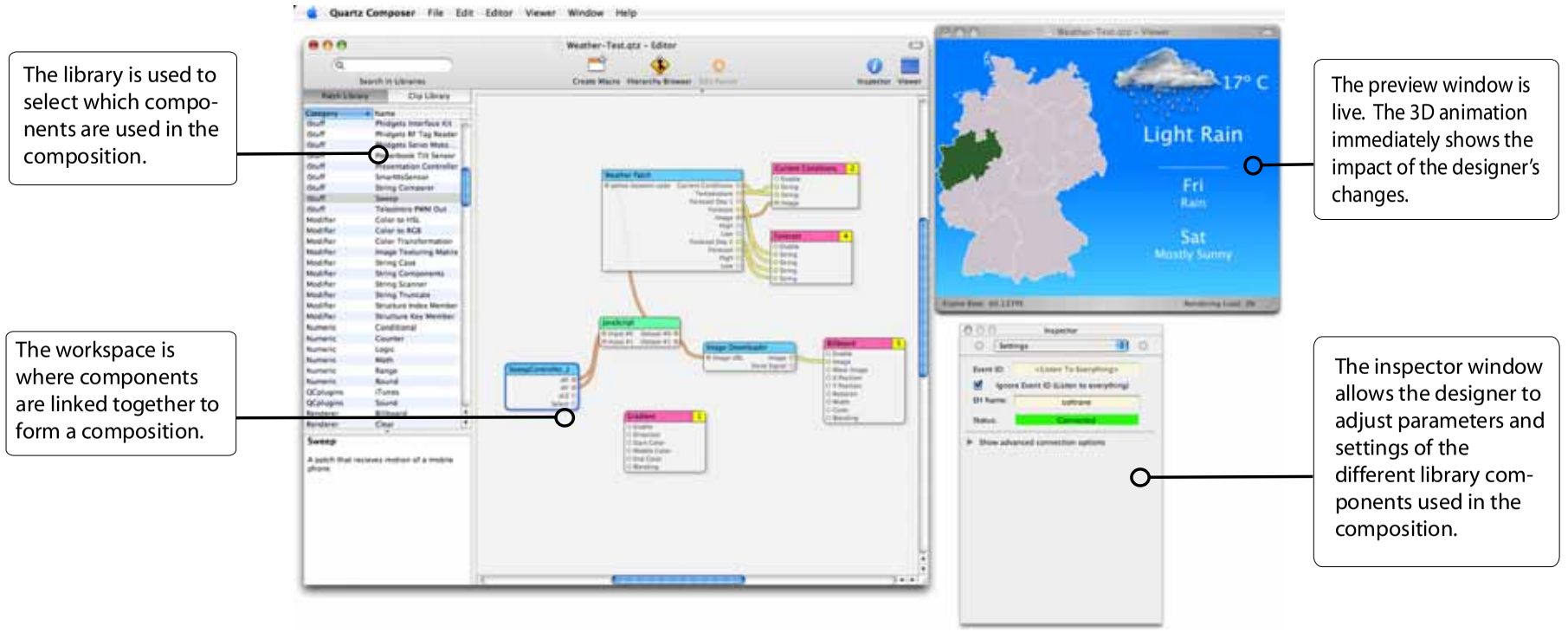


Figure 3.22: Apple's Quartz Composer is a visual programming environment designed to support rapid creation of 3D interactive visualizations. We have extended it to support prototyping physical user interfaces. This screenshot shows the development of a weather application for a large public display in a train station.

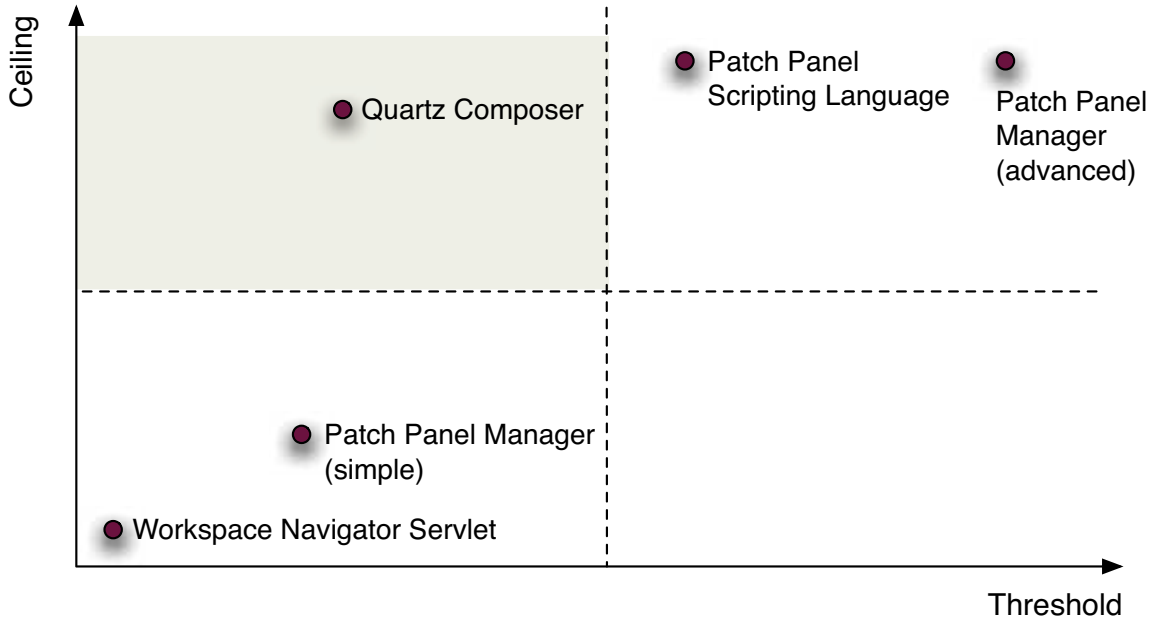


Figure 3.23: A rough comparison of the different Application Layer programming interfaces of the iStuff Toolkit.

Other Rapid Prototyping Environments

Many alternative visual programming environments exist

Quartz Composer is very similar to Max/MSP[11] and LabVIEW[12] in that it targets developers, researchers, and interaction designers. They differ in that Max/MSP's strength is audio processing, LabVIEW is geared towards electrical signal analysis, and Quartz Composer focuses on interactive multimedia and 3D rendering. Quartz Composer is a live editor, whereas Max/MSP and LabVIEW have separate edit and run modes. The Calder toolkit (Lee *et al.*, 2004a) supports the Macromedia Director[13] development environment; Phidgets (Greenberg and Fitchett, 2001) and Teleo provide hooks to work with Max/MSP and Adobe Flash[14] as development environments; d.tools (Hartmann *et al.*, 2006) supports visual programming through a custom visual statechart editor. Exemplar (Hartmann *et al.*, 2007) is an extension to d.tools that introduces techniques for authoring sensor-based interactions by demonstration, combining direct manipulation and pattern recognition techniques to enable designers to control how interaction rules are created. Other work in the realm of visual configuration of ubiquitous computing environments focused heavily on end-users, such as the Jigsaw Editor (Humble *et al.*, 2003) and the CAMP (Truong *et al.*, 2004) magnetic poetry interface. Although valuable for end-users, the low ceiling of these environments makes them less suitable for advanced prototyping activities.

[11] http://www.cycling74.com/products/maxmsp
[12] http://www.ni.com/labview/
[13] http://www.adobe.com/products/director/
[14] http://www.adobe.com/products/flash/flashpro/

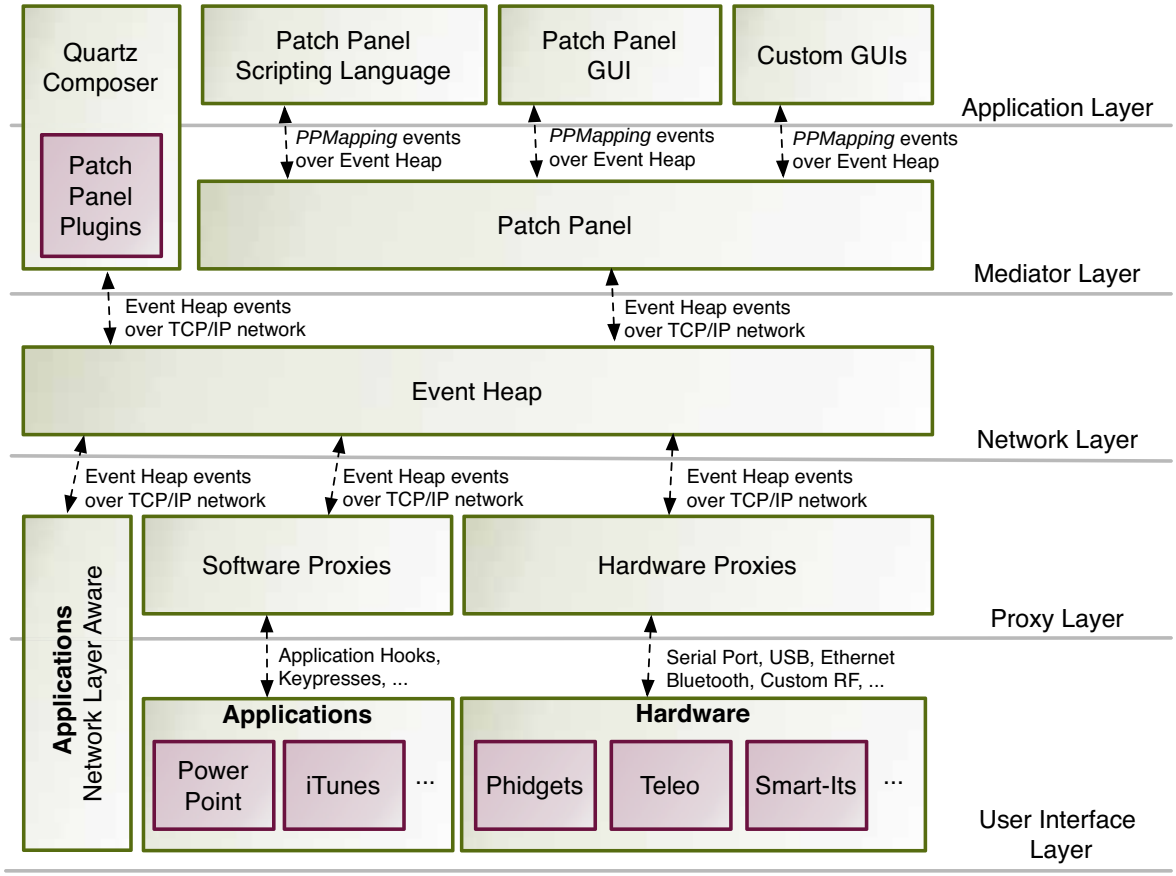


Figure 3.24: Summary of the layers of the iStuff Toolkit architecture.

Layer Summary

The application layer allows developers to rapidly prototype different application designs by defining interaction logic. Several interfaces for the iStuff Toolkit have been demonstrated, ranging from textual to visual programming, that allow the designer to define how the user interface responds to certain inputs. A rough comparison of the range of interfaces available for the iStuff Toolkit architecture is given in Figure 3.23.

3.2.6 iStuff Architecture Summary

Figure 3.24 places each of the pieces of the iStuff Toolkit architecture back into the layer diagram presented in Figure 3.1. Through the various layers of communication abstraction, the architecture solves issues of heterogeneity, interoperability, and dynamic composition of distributed components. The remaining sections will illustrate how this architecture is used in practice through a series of examples, and evaluations.

3.3 iStuff Prototyping Examples

High ceiling demonstrated through examples

Several examples applications such as iClub and Workspace Navigator have been used to illustrate how different layers of the iStuff Toolkit architecture interact. In this section, we will further demonstrate the *high ceiling* of the toolkit architecture through two examples we developed: Elope and iStuff Mobile.

3.3.1 Elope

The Elope project (Pering *et al.*, 2005) simplifies the process of connecting and configuring mobile devices to work with their environment. The proof of concept for this publication was built using the iStuff Toolkit. It uses RFID-tagged objects to invoke a distributed wireless application that combines a mobile device and an interactive space.

Phone storage continues to increase

The primary motivation for marrying mobile devices with interactive spaces is that the devices typically possess storage and computation resources to manage our personal data, but are not optimally suited for all tasks due to their small displays and keyboards. This observation served as the inspiration for the Personal Server Project (Want *et al.*, 2002) at Intel Research, a model of computing in which mobile devices wirelessly use the superior user-interface capabilities of nearby infrastructure. Interactive spaces afford a rich user experience; wirelessly blending them together with mobile devices results in the best of both worlds. Fully realizing the synergy between mobile devices and interactive spaces requires a smooth integration process not encumbered by messy cables or elaborate connection and configuration menus.

I/O resources in environment are better

Interactive spaces (Johanson *et al.*, 2002a), like the one discussed earlier in Figure 3.2, include wall-size interactive surfaces, surround-sound speaker systems, and specialized input devices to facilitate group interaction.Users naturally want to configure them to work with the personal data they carry on their mobile devices (such as laptops, PDAs, and mobile phones). Because two devices can meaningfully interoperate in a variety of ways, the result is often ambiguity for both devices. For example, specifying that a mobile phone should be connected to a wall-size interactive surface does not sufficiently specify a configuration. The display could be used to show pictures, movies, or presentations stored on the device. The number and variety of applications involving the combined use of the two computers is virtually unlimited.

Need to simplify configuration

The Elope approach uses tagged physical objects to embody different configurations, leveraging the affordances of the physical world to characterize the intended configuration (see Figure 3.25). For example, consider a presentation remote controller that represents a presentation service in a conference room. After scanning the remote, the phone is then "married" to the interactive space, simplifying the integration tasks (such as network,

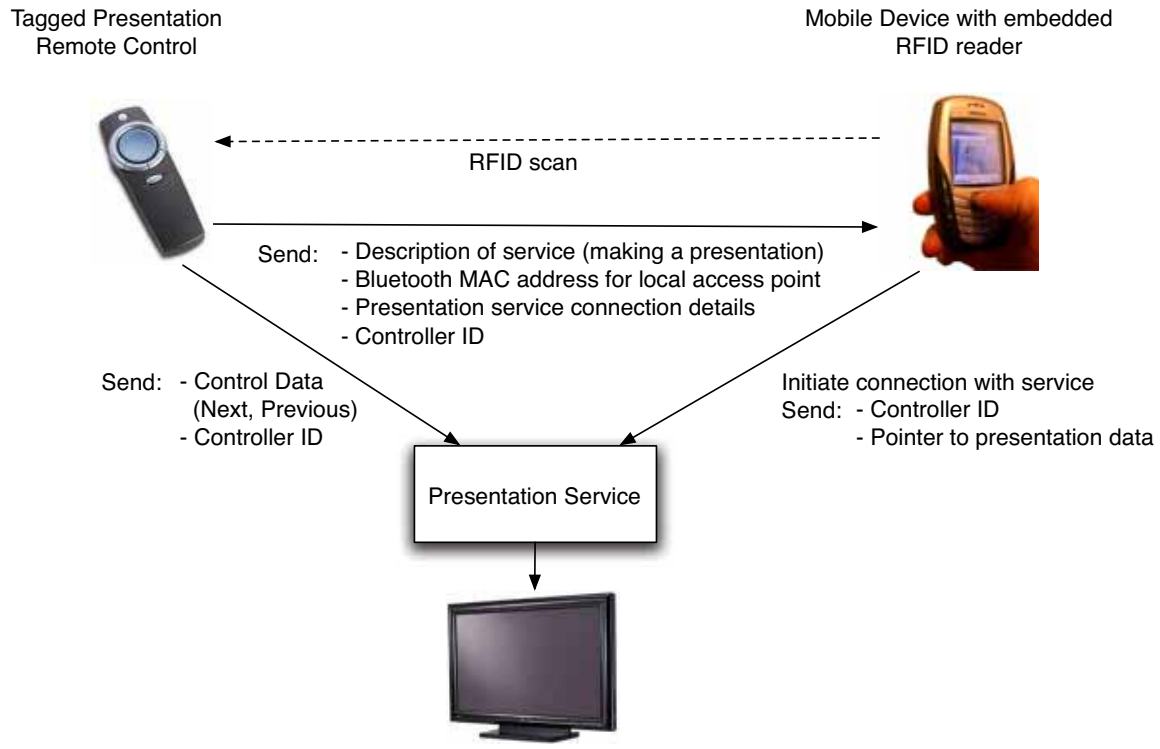


Figure 3.25: A system diagram illustrating the coordination between the components to set up a presentation by scanning a presentation controller.

device, and application configuration). As an analogy, instead of a formal wedding involving a complicated ritual, the union between space and device is accomplished quickly, as if a couple had eloped. This streamlined process significantly eases the integration burden for users, allowing them to concentrate on higher-level tasks (such as giving their presentation).

Use objects to represent intentions

The Elope system combines advanced mobile devices, interactive spaces, and tagged objects to enable the complete configuration of the space (based on simple user action), including launching the desired application and loading a user's personal data. These technologies combine to support a model requiring "near zero" configuration–the minimum interaction necessary to perform a task. The system encapsulates an "intention" in a physical object. The following scenario illustrates the potential of the Elope concept:

> At home, Ms. Grünberg prepares a presentation on her PC she will give to a valued client later that afternoon. When complete, she copies the presentation from her desktop computer to her smart (mobile) phone before heading out to the afternoon appointment.
>
> When she arrives, her hosts escort her to the conference room. After greeting the other participants, she scans the RFID tag embedded in a Meeting Minder device located in the room by pressing the "scan" button on her mobile device. She is now

signed into the meeting, and her email address is automatically registered to receive a copy of the meeting minutes.

Her hosts hand her the wireless Presentation Remote Control for the room. She scans the RFID tag in the remote control with her phone, triggering a list of likely presentations to appear on the room's main screen. Using the remote control, she selects the appropriate presentation, causing it to be accessed from her phone and displayed in full-screen mode on the projection surface. She then advances through her presentation while freely walking around the presentation space.

Halfway through the presentation, Mr. Jones wishes to share some supporting material from one of his own presentations. He borrows the remote control from Ms. Grünberg, scanning it with the RFID reader in his laptop, thus duplicating the display from his laptop on the presentation surface. After the meeting participants discuss his slides, Ms. Grünberg again scans the remote control, returning the screen to the place in her presentation where she left off.

At the end of the meeting, the participants press the "end meeting" button on the Meeting Minder device, closing the shared workspace and automatically sending a copy of the meeting minutes to all participants.

In this business presentation scenario, each of them (Grünberg and Jones) have scanners embedded in their personal devices (phones and laptops). The environment also contains shared presentation surfaces and two physical objects functioning as gateways to seamlessly engage their associated services. The interaction is supported by two physical objects in the space:

- *Meeting Minder*. This object allows users to form a shared information space facilitated by the proximity of tags to the mobile computing components. When wireless communication channels are established through Elope, the components are logically joined automatically, allowing users to take advantage of the rich interaction capabilities now provided by the space to share information. Interaction with the Meeting Minder could grant easy association with the meeting space, including a list of email addresses provided to all participants.

Prototyped using the iStuff Toolkit

- *Presentation Remote Control*. This physical object serves as a token for giving a presentation in the room. After the user's mobile computer scans the remote control's tag, it can interact with the presentation service to show the user's presentation. Tags automatically configure the initial presentation setup and transition between presenters fluidly and without encumbrance. The remote control can then be used to interact (such as to advance to the next slide) with the presentation.

A prototype of the Elope system was built using the iStuff Toolkit. The prototype system includes several technologies (see Figure 3.26):

Figure 3.26: Hardware prototype components, including tagged presentation remote control (upper left), prototype mobile device (right), and RFID reader (bottom left). The remote control is tagged with an RFID tag (black circle on end), and the RFID reader is exposed to show its inner circuitry. (A Euro, a British Pound, and a U.S. quarter are included for scale.)

Handheld RFID reader

- *Mobile device and RFID reader.* A Stargate mobile platform[15] serves as the user's mobile device, processing messages from the handheld RFID reader and delivering them to the infrastructure using the Bluetooth radio standard. The RFID reader is built around a small keyfob-size battery-powered RFID reader based on the M1-Mini reader from SkyTek[16]. The reader scans for tags when the user presses a button, communicating the results to the Personal Server using a Mote radio[17]. The cell phone then uses the Bluetooth radio to communicate with the room infrastructure. Although the RFID reader is physically separate from the user's device in the prototype, the reader and device effectively function as a single device to the rest of the system and are easily repackaged into a single compact unit, similar to the Nokia 3220 Near Field Coupling cell phone.

Tagged environment

- *Presentation Remote Control.* This physical device, augmented with a Texas Instruments (ISO15693) RFID tag[18] and several buttons, provides the necessary connection information and triggers events in the infrastructure. The tag contains up to 256B of information and is programmed with basic information (such as the object's unique ID, a description of the service provided, the Bluetooth address of a

[15] `http://platformx.sourceforge.net/`
[16] `http://www.skyetek.com/readers_Mini.html`
[17] `http://www.xbow.com/Home/HomePage.aspx`
[18] `http://www.ti.com/rfid/docs/products/transponders/1356mhz-encapsulated.shtml`

nearby access point, and the appropriate service for the mobile device to contact). The buttons trigger another Mote radio to provide local wireless broadcast communications.

Configuration information is encoded into RFID tag

Once a mobile device scans the RFID tag embedded in the Presentation Remote Control, the device uses the Bluetooth address obtained through the interaction to form a local-area IP-capable network connection using the Bluetooth personal area network profile. Then, using the additional information in the tag, the mobile device posts an event to the Event Heap with the remote control's ID, the desired service, and a self-referencing URL. Mappings in the Patch Panel are used to invoke the necessary services and to link the presentation remote control and the presentation application. The self-referencing URL points to a Web server running on the mobile device itself, providing the data necessary to show the user's personal presentation. The buttons attached to the Presentation Remote Control independently broadcast messages to the room that are then routed by the room's middleware to the presentation application. This system, based on the Elope architecture, provides a complete mechanism for showing and controlling a user's personal presentation while limiting the user's interaction to a single device–the Presentation Remote Control. Including an RFID tag makes this interaction possible because it simplifies the connection process, instead of requiring the user to perform multiple integration steps. The prototype is capable of showing a presentation served from a mobile device in approximately 13 seconds, a period representing the end-to-end delay from scanning to presenting.

Elope shows how the iStuff Toolkit architecture supports creating a fairly complex research prototype quickly and without wasting much time on infrastructure coding.

3.3.2 iStuff Mobile

Toolkit support for sensor-enhanced phones

iStuff Mobile is a toolkit that employs the iStuff Toolkit architecture to simplify prototyping of new sensor-based mobile phone interactions for ubiquitous computing environments. This extension of the iStuff architecture into the mobile phone domain enables us to explore a wide range of prototypes.

With iStuff Mobile, hardware prototypes can be built by simply attaching a commercially available sensor network module to the back of a standard mobile phone using Scotch tape (see Fig. 3.27).

iStuff Mobile Architecture

iStuff Mobile (Ballagas *et al.*, 2007a) is designed as a compound prototype architecture (Abowd *et al.*, 2005) where part of the software is distributed across separate computers. This compound architecture, shown in Fig. 3.28,

Figure 3.27: Back view of a mobile phone augmented with a Smart-Its sensor board in iStuff Mobile. The sensors can be attached to the phone in whatever position the designer finds most appropriate. The pictured Smart-It contains a 3D accelerometer, microphone, and sensors for light, pressure, temperature, and voltage.

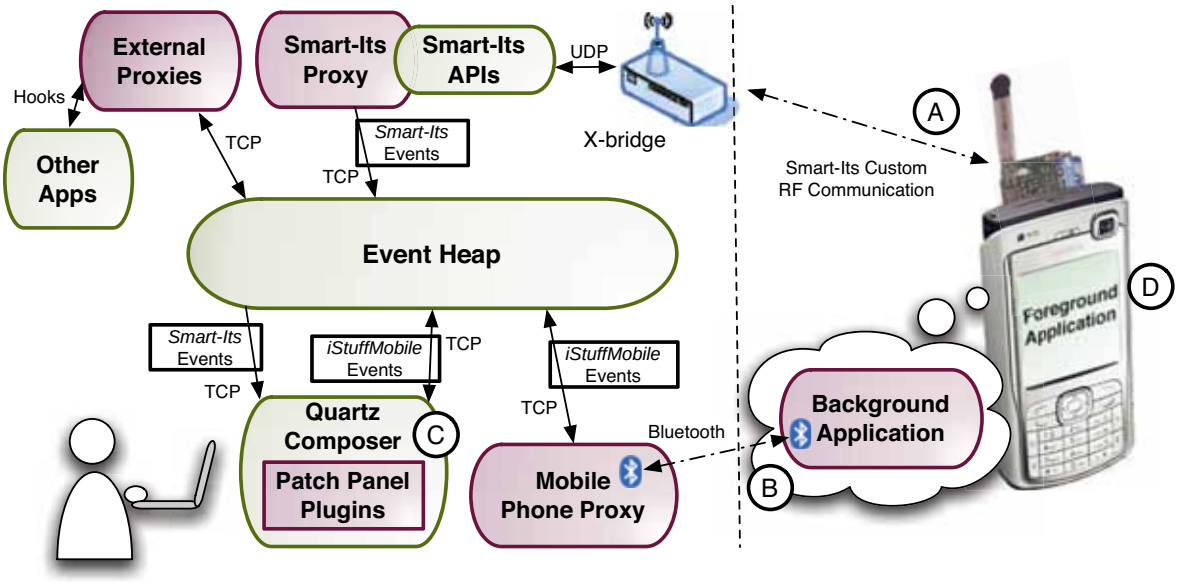


Figure 3.28: The iStuff Mobile architecture. Dark boxes indicate our own technical contributions; light boxes are third-party developments. (A) When the phone is moved, for example, the Smart-Its sensor board transmits the resulting sensor data wirelessly to the Smart-Its X-bridge. Our Smart-Its Proxy collects that data from the X-bridge over ethernet, and encapsulates it into *Smart-Its* events. (B) Mobile phone input events (such as key presses) are intercepted by the background application and passed to the Mobile Phone Proxy over a Bluetooth connection, and the proxy encapsulates the data into *iStuffMobile* events. (C) The Quartz Composer GUI is extended with special plugins for the Patch Panel to transform input events into desired output events. (D) For mobile phone output, the Mobile Phone Proxy listens for *iStuffMobile* events, and passes the resulting commands to the mobile phone background application over the Bluetooth connection. The background application then either executes the command directly, or forwards it to the foreground application as appropriate.

allows interface designers to prototype interactions that may be beyond the capabilities of current mobile phone hardware. In addition, this architecture provides communication capabilities necessary for ubiquitous computing application scenarios. The disadvantage of the compound architecture is that it spatially restricts experiments. While a direct communications channel between the sensors and the phone (e.g., through a Bluetooth connection) may be more efficient and less spatially restrictive, this would eliminate the prototyping benefits gained from using the Quartz Composer visual interface, which allows the relationships between user activity and application feedback to be changed at run-time using a comfortable desktop GUI. The delay between user action and application feedback is small enough to easily maintain causality (Card *et al.*, 1983) and support a wide range of tasks including continuous pointing tasks which require a latency much less than 100ms.

Compound prototype architecture has pros and cons

Mobile Phone Application Support

The iStuff Mobile architecture divides the mobile phone application into two parts. The *foreground* application is what the user interacts with during testing. The *background* application, provided by iStuff Mobile, is designed to simplify the work of the interaction designer creating the functional prototype. It is not directly visible to the user.

Foreground is what the user sees; Background helps developers

Designers can remotely execute commands on the phone by sending *iStuff-Mobile* events to the desktop-based Mobile Phone proxy, which relays the commands to the background application on the phone via a Bluetooth connection (see Fig. 3.28). The background application relays the commands to the foreground application or the operating system as appropriate. The background application can also intercept user actions, such as key presses, from the foreground application, which are relayed to the proxy over the Bluetooth connection and subsequently posted as events on the Event Heap. In terms of the iStuff Layer Architecture, this background application can been seen as part of the Proxy Layer to facilitate communication with the foreground application. The prototype implementation of the background application on the mobile phone was designed to include the following feature set. It does not cover the entire design space of interaction possibilities, but it does enable a wide range of interesting interactions, and the architecture encourages expansion to include more features.

1. *Bluetooth Communication*: communicate with the proxy through a low-latency wireless communications channel.
2. *Sound Playback*: trigger available sounds to be played and stopped.
3. *Vibrator Control*: trigger the vibrator to start and stop.
4. *Key Capture*: intercept key events from the foreground application and relay them to the proxy for processing.

5. *Foreground Application Key Simulation*: pass key events to the foreground application.

Wide range of functionality supported in Background application

6. *Launch External Application*: launch any application on the mobile phone.

7. *Profile Control*: programmatically change the ring profile of the mobile phone.

8. *Backlight*: control the backlight programmatically.

9. *Run Application in Background*: send the current foreground application to the background.

10. *Camera Control*: use the camera on the mobile phone for taking pictures, videos, or interactions using motion estimation such as the Sweep technique as discussed previously in Section 2.1.1.

Other platforms may be suitable.

We have built a prototype implementation of the background application using the Symbian Series 60 operating system. Our analysis shows that Windows Mobile 6.0 SmartPhone Edition would be a good candidate for porting the iStuff Mobile background application because it shares many of the same capabilities as Symbian Series 60. Java 2 Micro Edition, on the other hand, is currently not a candidate platform as the background application would be lacking critical functionality. We believe that it is possible to port the background application to Linux-based mobile phone platforms such as the Motorola E680i[19] or OpenMoko[20], but open source development efforts on these phones are still in their early stages.

Foreground can be produced using rapid prototyping tools of choice

The *foreground* application is the application the user sees and interacts with. iStuff Mobile is designed to be used with any foreground application and communicates primarily through system events (e.g., key presses). Designers are expected to prototype their own mobile phone application using rapid prototyping solutions such as static images, Adobe's Flash Lite[21], or a scripting language like Python[22]. Alternatively, designers have the option to program their own application using Java[23] or native code. Lastly, designers can prototype interactions with existing foreground applications that come with the mobile phone, such as its Address Book or Calendar, despite the fact that these applications were not explicitly designed to accommodate new interaction techniques. In terms of the iStuff toolkit layered architecture, this foreground application is part of the User Interface Layer.

[19] `http://sf.net/projects/e680`
[20] `http://openmoko.org`
[21] `http://www.adobe.com/products/flashlite/`
[22] `http://opensource.nokia.com/projects/pythonfors60/`
[23] `http://java.sun.com/j2me/`

Other Mobile Phone Interface Prototypes

Seminal mobile phone interactions represent the types of scenarios iStuff Mobile supports

The TEA project (Schmidt *et al.*, 1999) uses a predecessor of the Smart-Its platform to demonstrate mobile phone context interactions. Also, Harrison *et al.* (1998) and Hinckley and Horvitz (2001) built custom PDAs and mobile phones with integrated sensor hardware. These projects broke new ground and demonstrated a broad vision for what types of interfaces are possible for mobile phones, but the focus was on the interfaces, not the development process: Their software prototypes were one-off developments. iStuff Mobile focuses on providing a reusable prototyping framework for research and design scenarios like these.

Other Mobile Phone Toolkits

d.tools facilitates form factor exploration

d.tools (Hartmann *et al.*, 2006) allows designers to rapidly prototype handheld devices including mobile phones. Its strength is the ability to explore different mobile phone form factors and sensor placements. However, its lack of support for critical phone functionality, such as voice calls, and its wired sensors currently limit the toolkit's ceiling for many mobile phone application scenarios.

Topiary facilitates WOz prototypes of location-based interactions

Topiary (Li *et al.*, 2004) is a toolkit for building Wizard of Oz prototypes of location-based applications on existing handheld devices. While valuable for location-based scenarios, it does not address other types of physical sensors, and does not simplify the construction of functional prototypes. iStuff Mobile is the first toolkit to allow rapid prototyping of functional physical sensor-based interactions with existing mobile phones.

Recreating Seminal Mobile Phone Interactions

Example: Tilt-scroll

To demonstrate the utility of the framework, we have used it to recreate several mobile phone interactions discussed in previous literature. Harrison *et al.* (1998) introduced a tilt-scrolling interaction for mobile devices. The implementation consisted of a PDA augmented with an accelerometer and pressure sensors. To activate tilt-scrolling, the user squeezes the sides of the device with her thumb and forefinger. The more the user tilts, the faster the device scrolls. Figure 3.29 shows that, with iStuff Mobile, the researchers could have created their prototype in just six Quartz Composer nodes and one simple JavaScript node, and in particular without having to write any networking, driver or glue code, or doing any electronics work.

Example: Context-aware profile control

We were also able to recreate some context-aware interactions described in the literature. Schmidt *et al.* (1999) and Hinckley and Horvitz (2001) describe a scenario where the mobile phone ring tone profile is automatically switched to vibrator-only when the mobile phone is in the user's hand, since an audio notification is unnecessary in that situation. This interaction was

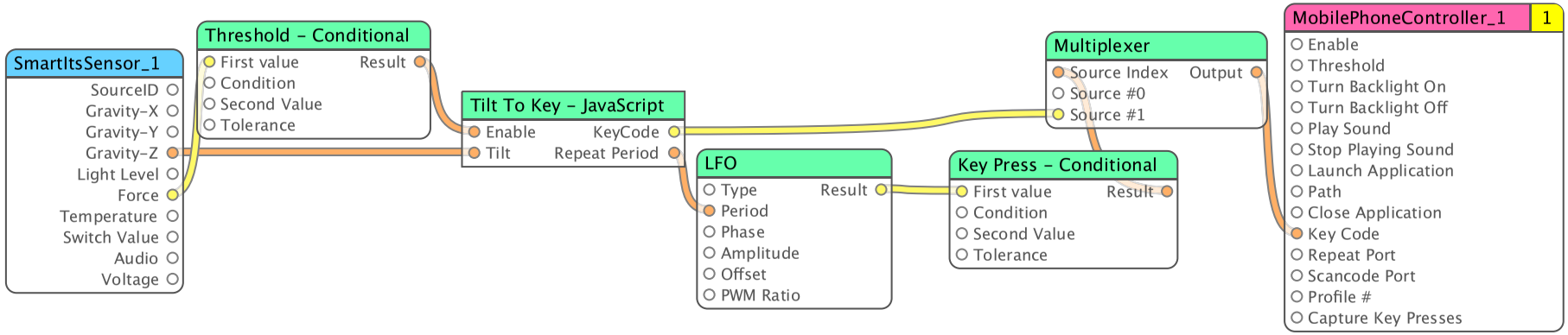


Figure 3.29: The Quartz Composer implementation of the tilt-scrolling interaction from (Harrison *et al.*, 1998). Squeezing input is measured by the "Force" output from the ***SmartItsSensor_1*** and is tested with a simple threshold. The result is passed to the ***Tilt To Key - JavaScript***, which maps various tilts in the Z-direction of the gravity sensor to different key codes and key repeat rates. The outputs from that JavaScript node include "KeyCode", which represents the appropriate key (up or down arrow) depending on the current tilt, and "Repeat Period", which specifies how fast the ***LFO*** (low frequency oscillator) node should operate. For this scenario, larger tilt is mapped to faster repeat rates. The ***Key Press - Conditional*** changes the oscillator to function like a binary clock, regularly switching between 0 and 1. "Source #0" (which defaults to 0) represents no key pressed, and "Source #1" represents the key specified from the JavaScript node. The key is then passed to the ***MobilePhoneController_1*** to forward to the mobile phone. The naming convention of the iStuff Mobile related nodes corresponds to the name of the device being controlled. (***_1*** helps distinguish multiple devices of the same type.)

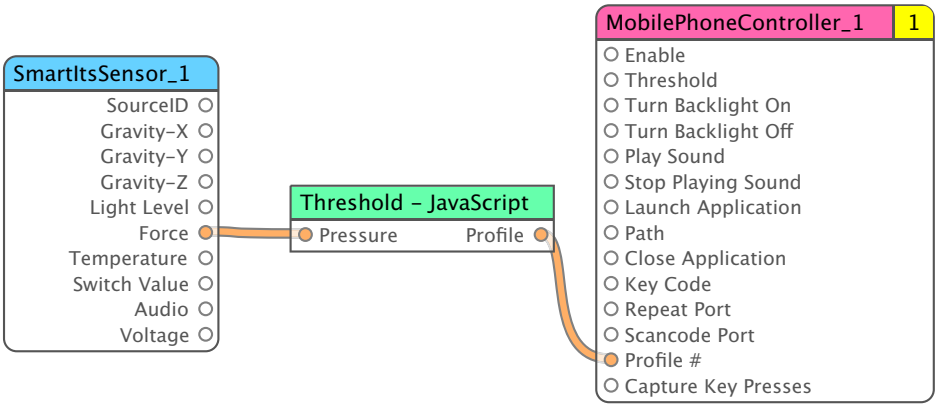


Figure 3.30: Implementation of the context based profile change described in (Hinckley and Horvitz, 2001; Schmidt *et al.*, 1999). The ***Threshold - JavaScript*** node changes the profile based on the pressure sensor (indicating the user is holding the phone).

recreated using the Smart-Its pressure sensor to detect when the user was holding the phone. Pressure sensor activity triggers a command to the mobile phone proxy to switch the ring tone profile, and inactivity triggers the command to switch the ring tone back. Using iStuff Mobile, building this prototype becomes trivial (see Fig. 3.30).

Example: TiltText

As another example, the TiltText (Wigdor and Balakrishnan, 2003) technique simplifies text entry using a numeric keypad by adding tilting. Tilting the phone to the left activates the first letter, tilting upward activates the second letter, tilting to the right activates the third letter, tilting downward activates the fourth letter (applicable for keys '7' and '9') and no tilt activates the standard numeric character. This technique has been shown to be faster than MultiTap and comparable to dictionary-based techniques. Again, we were able to quickly recreate this interaction using the iStuff Mobile framework (see Fig. 3.31). This example is a good illustration of how capable Quartz Composer is of expressing state-based interfaces. In the implementation, the tilt state is determined through the *Tilt State - JavaScript* node and passed to the *TiltType - JavaScript* node, which modifies the key presses based on the current state. Note that since there are 5 different states for 12 different phone keys, this would be cumbersome to model as a state machine, but it is fairly easily modeled using Quartz Composer.

Ubiquitous Computing Prototyping Scenarios

To show how the framework goes beyond the localized sensor-based interactions described so far, we will now demonstrate how it simplifies prototyping entire ubiquitous computing scenarios.

The idea of using the mobile phone as an input device for ubiquitous com-

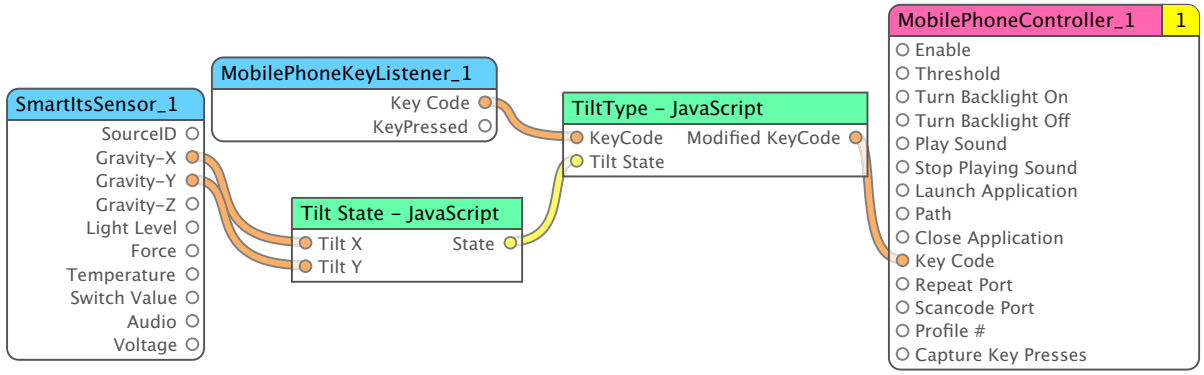


Figure 3.31: The TiltText (Wigdor and Balakrishnan, 2003) technique maps numeric keys to different characters based on the tilt of the device.

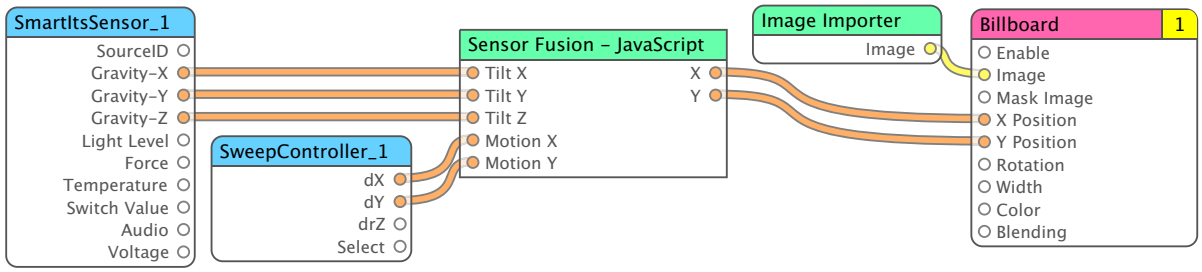


Figure 3.32: The Quartz Composer implementation for combining accelerometer data with camera-based motion detection to improve motion detection accuracy. The ***Sensor Fusion - JavaScript*** node implements the algorithm to combine the sensor values in a meaningful way. The JavaScript logic can be modified at run-time to test and refine the sensor fusion strategy. The standard ***Billboard*** node of Quartz Composer displays an image to the screen (e.g., a cursor). The output of the sensor fusion algorithm in the JavaScript node controls the position of the billboard on the screen.

Example: Large public display weather browser

puting application scenarios is very compelling because the phone is almost always with us (Patel *et al.*, 2006). One technique that has demonstrated some potential for this use is the Sweep (Ballagas *et al.*, 2005) interaction technique, which uses camera-based motion estimation to allow the phone to be used as a relative pointing device for public displays. The motion estimation algorithm on the low power mobile processor is not perfect and suffers from some estimation errors. We used the iStuff Mobile framework to combine accelerometer data with the camera information to improve the motion estimation, as shown in Fig. 3.32. This allows the mobile phone to serve as a more accurate pointing device, for example when interacting with public displays. The synergies of choosing Quartz Composer as a visual programming environment are demonstrated in this scenario because Quartz Composer makes authoring visually compelling interactive graphics very simple (see Fig. 3.33). This means that the iStuff Mobile input techniques can directly drive these 3D visualizations and the framework provides end-to-end prototyping assistance.

Mobile phones have also emerged as popular presentation remote controls,

Figure 3.33: The proof of concept weather browser application allowed users to navigate through regions on the map using the Sweep technique. The weather forecast is updated live using RSS feeds from Yahoo! Weather.

Example: Multi-screen presentation remote control

due in part to the success of tools like Salling Clicker[24]. But ubicomp environments like interactive workspaces (Johanson *et al.*, 2002a) have multiple screens that can be taken advantage of to enhance the presentation. As a proof of concept, we developed a multi-screen presentation interface (see Fig. 3.34). In this scenario, one screen is showing the current slide of the presentation, while the second screen is showing the presentation history. These presentations are controlled through simple PowerPoint proxies running on different machines in the interactive workspace. Each proxy listens for events with different names (e.g., *PresentationController_1*) so that they can be individually controlled. By pressing a key on the mobile phone, the user advances the slide on each screen.

Example: Keyboard redirection

Ubiquitous computing environments such as interactive workspaces (Johanson *et al.*, 2002a) are rich in input and output capabilities, including touch sensitive wall-sized displays, and interactive tabletop displays. PointRight (Johanson *et al.*, 2002b) demonstrates a system that lets users redirect the mouse and keyboard input to the different computers in the room. We wanted to demonstrate a system that would also let users redirect their keyboard input to a mobile phone. The resulting prototype is shown in Fig. 3.35.

Example: Continuous speech recognition

Using an almost identical configuration, we were able to prototype a scenario where the user could dictate text to the phone using continuous speech recognition. Mobile phones are years away from having the processing ability to support such continuous speech recognition, but iStuff Mobile enabled us to create a functional prototype today.

In summary, the preceding examples illustrate our key point: With iStuff Mobile, a researcher or designer with a new idea for a mobile or ubiquitous interaction technique can get to his "core business" – exploring and

[24] `http://www.salling.com/Clicker/`

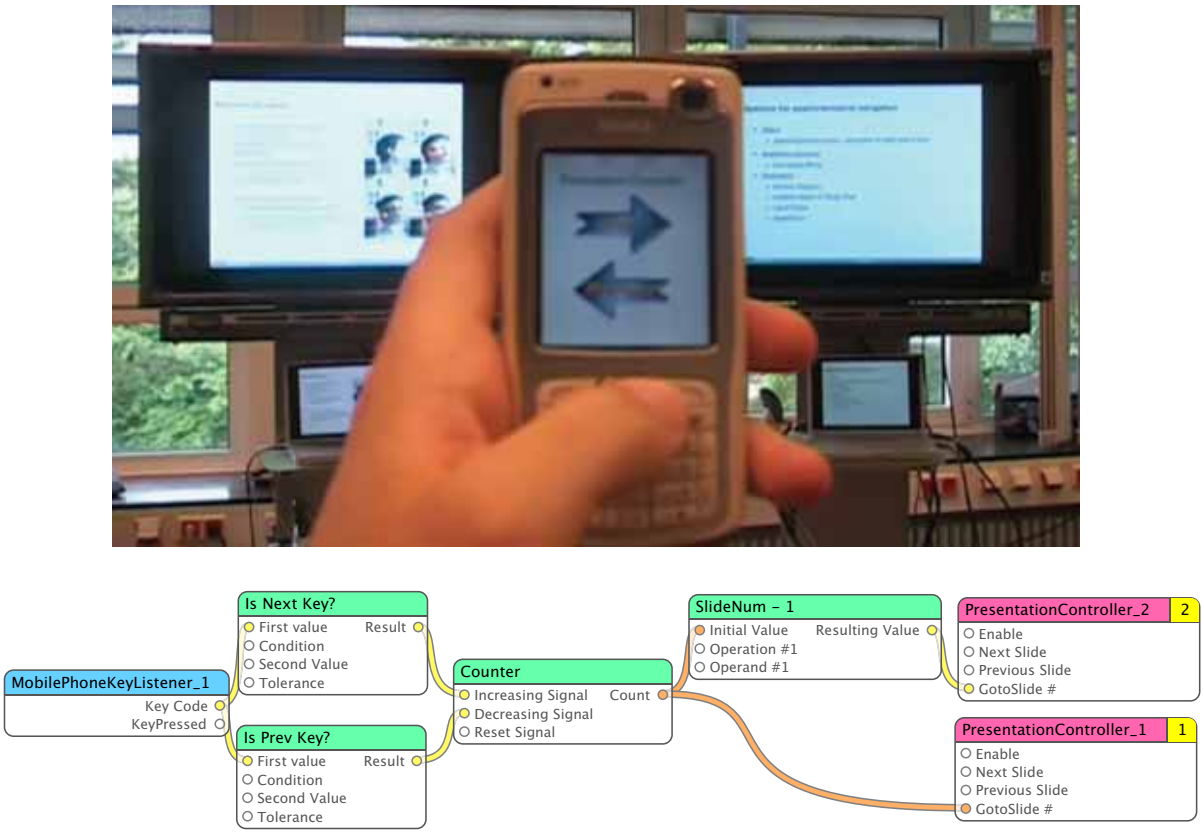


Figure 3.34: (Top) A multi-screen presentation application that uses a mobile phone as a presentation remote control. The foreground application in this example is prototyped using a static image. The right screen shows the previous slide, and the left screen shows the current slide. (Bottom) The Quartz Composer implementation for the multi-screen presentation application. On the far left, the ***MobilePhoneKeyListener_1*** node receives the key presses from the iStuff Mobile Proxy. The two nodes on the far right are iStuff modules to control two instances of the same PowerPoint presentation, each running on a different computer in the interactive workspace (Top). No JavaScript nodes are required for this composition.

evaluating the technique – very quickly, without getting sidetracked by infrastructure glue code development or electronics hardware work.

However, to show the validity of our claims of thus improving the development process, an evaluation is needed. This is given in the next section.

3.4 User Evaluation

Evaluating toolkits is hard

Conducting a formal evaluation of a toolkit or software framework is an extremely difficult task. The qualities that are important for toolkits, such as development effort, are difficult to measure directly. Evaluating a software framework has issues similar to evaluating middleware. Edwards *et al.* (2003) point out that although we have good techniques for designing and evaluating interactive applications, we are lacking well formed techniques

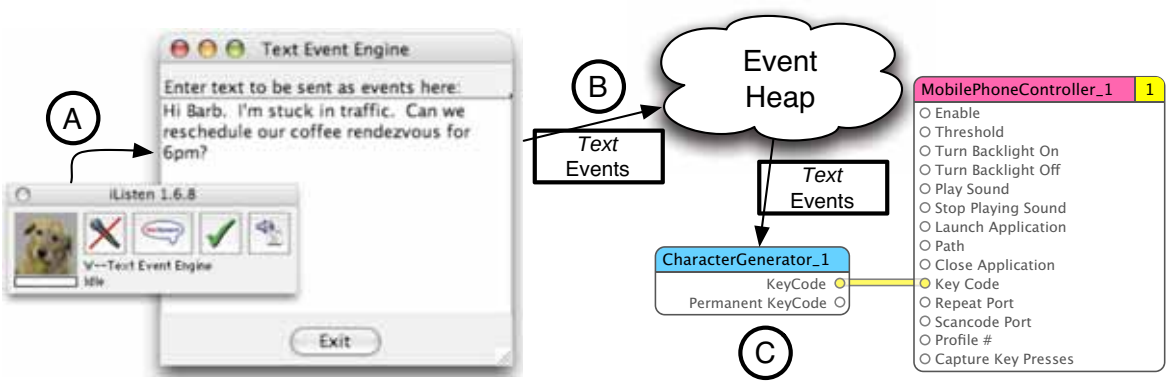


Figure 3.35: (A) The window floating on top belongs to iListen, a commercial application that supports continuous speech recognition (dictation) on Mac OS X. When speech is recognized, it is converted to ascii text and sent to the focused application as key events. (B) Our Text Event Engine is a Java application in focus that produces *Text* events for each key entered in the textbox. In this example, the user is dictating an SMS message. (C) *Text* events are recognized by the ***CharacterGenerator_1*** and transferred to ***MobilePhoneController_1***. This composition can alternatively be used to allow users to type messages onto their mobile phone using a standard keyboard on their desk.

for designing and evaluating the infrastructure to support application development. Klemmer *et al.* (2004) provide a detailed discussion of this topic, and describe their various approaches to evaluate the Papier-Mâché toolkit. One approach is to measure the efficiency (e.g., development time, or lines of code) of developers while using a toolkit, but efficiency is also related to the quality of the resulting prototype, making these metrics difficult to isolate.

High ceiling demonstrated through examples

New toolkits are typically validated by the breadth of coverage in the designs they support (Greenberg and Fitchett, 2001; Ballagas *et al.*, 2003; Klemmer *et al.*, 2004) and their ability to recreate important interactions from the literature more easily (Klemmer *et al.*, 2004). Similarly, the high ceiling of our iStuff architecture was demonstrated in the last section by the range of prototypes it enables.

Low threshold demonstrated through experiment

The rest of the evaluation is designed to show that the iStuff architecture has a low prototyping threshold. In this evaluation, we examine the efficiency of prototyping with the iStuff toolkit using development time and number of prototype iterations as the primary metrics. In the case of Quartz Composer, the lines of code metric is less relevant since only a small part of the modeling is done textually in JavaScripts.

In order to experimentally evaluate iStuff, we chose to compare the effectiveness of the new visual programming paradigm to the established Patch Panel scripting language.

To reduce task bias, we chose to examine four different design problems, each making use of different hardware components:

1. **Multi-screen presentation**. Participants were asked to prototype a presentation interface that used the mobile phone to control a presentation across several different screens, similar to Fig. 3.34 on p. 91. The station was equipped with a mobile phone and two remote machines connected to large LCD displays. The remote machines were running PowerPoint presentation software and connected to the design station via ethernet.

Variety of design problems chosen to reduce task bias

2. **Tilt-to-scroll**. Participants were asked to prototype a mobile phone interface that allows scrolling through large lists of data by tilting the phone, similar to (Harrison *et al.*, 1998) and Fig. 3.29 on p. 87. The station was equipped with a mobile phone and several Smart-Its sensors.

3. **Handheld music player**. Participants were asked to prototype a new handheld music player, including song selection and volume control functionality. The station was equipped with a variety of Phidgets sensors, and the iTunes software music player. This task was chosen to provide some level of comparability to d.tools and BOXES, where the portable music player serves as a prominent example.

4. **Remote steering**. Participants were asked to design a remote control mechanism for a model boat, where physical input would control an electronic motor to steer a boat rudder. The station was equipped with Phidgets sensors for input and Phidgets servo-motors and cardboard to build a low-fidelity boat prototype.

Full scenario descriptions from the test are provided in Appendix B.

The test consisted of 16 participants (10 male, 6 female) from a computer science course. On average, the students were in their 4th year of university studies. All students had completed lectures on Human Computer Interaction and were familiar with the fundamentals of iterative interaction design, prototyping, and evaluation.

The test was structured such that the group of participants received training before the design exercises with the following time schedule:

0:00 – 0:30: Introduction to iStuff prototyping principles
0:30 – 1:00: Training for Prototyping Environment 1
1:00 – 1:30: Design Task 1
1:30 – 2:00: Design Task 2
2:00 – 2:30: Training for Prototyping Environment 2
2:30 – 3:00: Design Task 3
3:00 – 3:30: Design Task 4

The test was designed as a within-groups study, with special care taken to avoid learning effects both in terms of the tasks completed and the prototyping environments used (see Table 3.1). The 16 participants were split up into 2 shifts of 8 and further split into 4 teams of 2 participants. During each shift, there was one team at each test station attempting one

Task order varied to reduce learning effects

of the 4 design tasks described earlier. After each task, the teams rotated to a different station such that each team performed the design tasks in a different order. The first shift of 4 teams completed their first 2 design tasks using the Quartz Composer GUI, then their last 2 tasks with the Patch Panel script. The second shift of 4 teams used the prototyping environments in the opposite order. Participants were instructed to stop after 30 minutes, regardless of whether or not they had a functional prototype ready.

Test Run #1 Group	A	B	C	D
Patch Panel GUI	Scenario 1	Scenario 3	Scenario 2	Scenario 4
Patch Panel GUI	Scenario 2	Scenario 4	Scenario 1	Scenario 3
Scripting Language	Scenario 3	Scenario 1	Scenario 4	Scenario 2
Scripting Language	Scenario 4	Scenario 2	Scenario 3	Scenario 1
Test Run # 2 Group	E	F	G	H
Scripting Language	Scenario 1	Scenario 3	Scenario 2	Scenario 4
Scripting Language	Scenario 2	Scenario 4	Scenario 1	Scenario 3
Patch Panel GUI	Scenario 3	Scenario 1	Scenario 4	Scenario 2
Patch Panel GUI	Scenario 4	Scenario 2	Scenario 3	Scenario 1

Table 3.1: User test scenario completion matrix.

We measured the time it took for the participants to build their first functional prototype, and the number of iterations completed in the time allotted. For this study, we define functional prototype as an artifact that successfully implements at least a portion of the functionality described in the design task. We define a design iteration as a full DIA cycle (design, implement, analyze). Every time the participants created a functional prototype, analyzed the problems, and made changes to the design we counted it to be a design iteration.

3.4.1 Experimental Results

Quartz Composer is faster and supports more iterations

Results from the experiment are shown using box-plots in Fig. 3.36. In the time measurement plot, if a group was unable to complete a prototype in the allotted 30 min., we assigned a time of 31 minutes to maintain the visual integrity of the box-plots, but omitted these data points during statistical analysis. The average time for the first functional prototype was 19.6 minutes for the Quartz Composer GUI, and 27.6 minutes for the Patch Panel script. However, we are unable to show that this difference is statistically significant because of the large number of incomplete first iterations for the Patch Panel script. Participants were able to complete at least one iteration of the test in the time alloted 81% of the time with Quartz Composer, compared to only 31% with the Patch Panel script ($p < 0.05$, Fisher's test). Participants were also able to complete an average of 2.5 iterations using the Quartz Composer GUI compared to 0.9 iterations using the Patch Panel scripting language ($p < 0.05$, Student's t-test). The combined results show

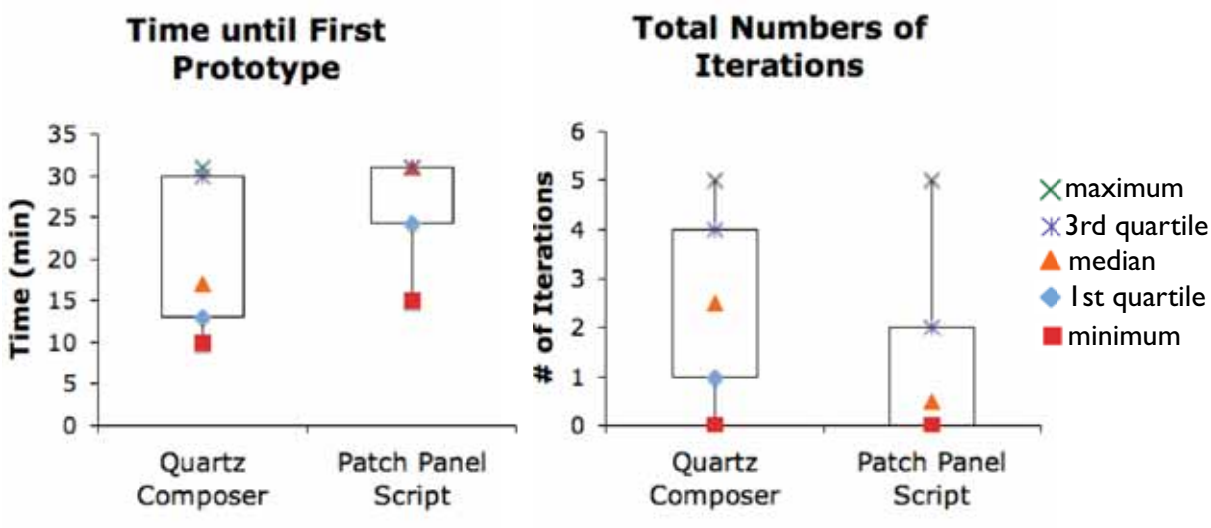


Figure 3.36: Results show that Quartz Composer is significantly faster and enables significantly more iterations than the Patch Panel scripting language.

that the Quartz Composer GUI is significantly faster than the Patch Panel script in building prototype iterations.

Clearly, there are disadvantages to this experimental design since the comparison is limited and doesn't reflect the range of prototyping approaches that exist today. The main take-away is that the study demonstrates a low prototyping threshold: with very little training, users could often build early prototypes in less than 30 minutes. The Patch Panel scripting language serves as a point of reference to interpret the results.

Main takeaway: low prototyping threshold

3.4.2 Questionnaire Results

Participants were given ten questions that should be judged by ranking on a 1 to 5 scale (see Appendix C). Additional free-form questions encouraged feedback on different aspects of the different rapid prototyping environments.

Quanititative Results

More than 90% of the participants preferred the graphical approach but also provided some critical thoughts that are presented in the qualitative results. Participants stated that the development capabilities of Quartz Composer were better than the scripting approach with an average score of 4.5 (5 = strongly agree). A rating of 4.6 (5 = strong preference) for the graphical programming approach compared to a rating of 1.6 (5 = strong preference) of the scripting approach further demonstrated a strong preference for the graphical programming. The appropriateness of the Quartz

Quartz Composer preferred

Composer metaphor was scored with an average of 3.9 (5 = very appropriate) which shows that participants generally agreed that the interface worked with the event passing concept of the iStuff toolkit. These results suggest the continuation of Quartz Composer development in the future.

highly extensible

An average score of 3.2 (5 = very frequent) was given to the question regarding the frequency of use of the built-in Quartz Composer patches, supporting the decision to extend Quartz Composer and therefore benefit from its built-in functionality. The Quartz Composer graphical approach was felt as being extensible in the future which is supported by a score of 4.4 (5 = very extensible). This result encourages the extension of the iStuff prototyping suite and strengthens the decision that was made to develop a software framework that makes the integration of new components very easy.

Training was sufficient

The clarity of the iStuff project concepts was rated an average score of 4 (5 = very clear), demonstrating that the training was effective for the scope of the user test and the participants did not struggle with the overall understanding.

Scripting approach more powerful

The scripting approach was felt as being more powerful with an average score of 3.1 (5 = very powerful). This justifies the need for state machine support in future versions of the Quartz Composer extensions as well as several other custom patches that support the prototyping process. The inner structure of the passed events should also be conveyed in a better way. In following iterations of the Quartz Composer extensions, existing features will be improved and new ones added.

More than 90% expressed to be more encouraged to undergo more iterations with their design using the graphical approach. This demonstrates general willingness to refine designs using the graphical programming model.

Qualitative Results

The participants were also given a chance to freely express their criticism of the system, tell what features or patches were missing or what should be changed in future versions. It turned out that the scripting approach was judged as immature although it has been used for a long time inside the iStuff project. This point was stated because the users had to apply a lot of workarounds to achieve a working solution. The results demonstrate the difficulty in creating a custom scripting language. Although it was refined and tested for over two years, the testers were still able to find new bugs.

Too many windows

Participants also expressed difficulty dealing with the array of support applications such as the Proxy Manager and the Event Logger in addition to the rapid prototyping environments. One suggested that the Event Logger application could be combined with the Proxy Manager such that events sent by the configured proxies could be directly analyzed.

Participants requested state machine capabilities, like a "Toggle Switch", in Quartz Composer. State machine like capabilities are available using the JavaScript capabilities, but this was not obvious to the testers. Prototyping scenarios might benefit from a custom state machine extension to better support these needs, but it is unclear how such a module might work.

Need for FSM support in Quartz Composer

Participants also felt the range of sensor types in the evaluation was limited and requested the integration of more sensor types.

This concludes our user-centered evaluation. The remaining section assesses the system performance of the iStuff Toolkit architecture.

3.5 Performance Evaluation

The distributed nature of ubicomp environments inevitably leads to system latency. However, latency is a critical aspect of the user experience. It is important to understand the implications of tampering with the tight interaction loop users require for fluid interactivity. Pointer input in traditional desktop systems is specially integrated into the operating system, bypassing normal system event queues, and allowing device movements to be echoed almost instantaneously. However, these optimizations are not available for the loosely-coupled, distributed infrastructure of ubicomp environments. We have identified three general types of latency in ubiquitous computing systems (Fig. 3.37):

Distributed interactions have latency

- *Internal latency* is internal to a system component, application, or device (e.g., program execution times).
- *External latency* occurs in communication between system components (i.e., networking)
- *Human latency* arises from human factors such as perception, cognition and reaction times.

From our experience in using the system, the Patch Panel has no problems maintaining causality by keeping delays below 100ms (Card *et al.*, 1983) for command-based interfaces (e.g., "turn on the lights"). However, other tasks, such as pointer manipulation, require a much lower latency. The distributed, loosely-coupled system architecture (shown in Fig. 3.37) has many different sources of external latency (due to communication overhead) and internal latency (due to internal processing overhead). In addition, there are several conditions that are known to cause perfomance degradation in the overall intermediation system.

- The most obvious degradation happens when the underlying network becomes overloaded with external TCP/IP traffic, for example if several people start to download a large file at the same time.

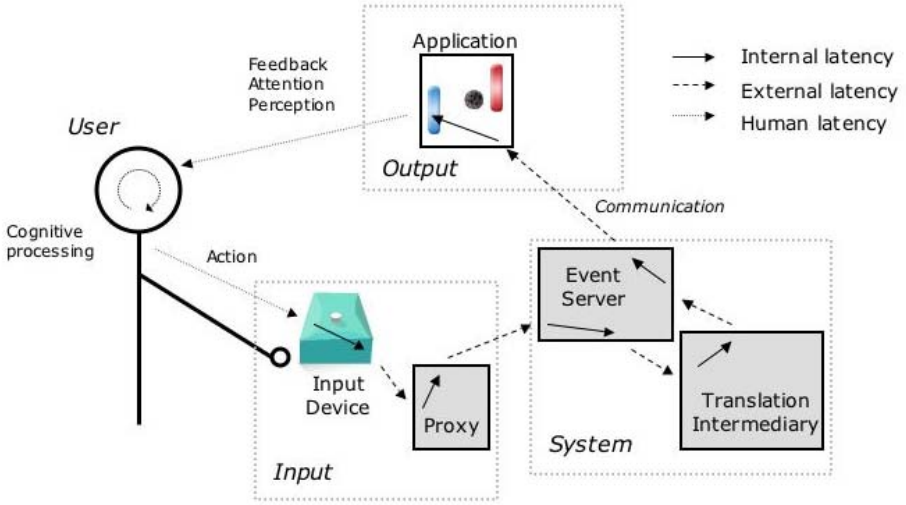


Figure 3.37: Sources of latency in tangible UIs for ubicomp.

Sources of latency in the iStuff Toolkit architecture

- An extremely high number of Event Heap clients (> 1000) slows down performance because this increases the template matching burden of the Event Heap for each incoming event (although this is typically never realized because of the Boundary Principle (Kindberg and Fox, 2002)).
- Large numbers of equations per event mapping affect performance because the Patch Panel must resolve the equations for each incoming trigger event.
- Large numbers of mappings affect performance because this increases the template matching burden of the Patch Panel, although trees and hashes are used to reduce this effect.

Benchmark performance analysis

To portray an example of performance degradation, we measured the performance as the number of mappings increased by increments of 100. Results are shown in Fig. 3.38. The benchmark consisted of a 2 computer setup networked over a 1Gb LAN, including a dual-2GHz G5 PowerMac running the Event Heap, and a 1.67GHz G4 PowerBook running the Patch Panel and a benchmark client. The benchmark client configured the Patch Panel with the appropriate mappings before measuring the time between sending a benchmark event and receiving the translated output of the intermediation. The translation mappings included a single equation to resolve at run-time. For each test condition, 200 round trip times (RTT) were measured. Note that variance is extremely high in the benchmark measurements due to the large number of random latency sources identified in Fig. 3.37.

To determine if this lag is acceptable for pointer manipulation, Fitts' law is the most commonly referenced human performance model. The mean time to select a target using a pointing device can be predicted as follows:

$$MeanTime = C_1 + C_2 I_D$$

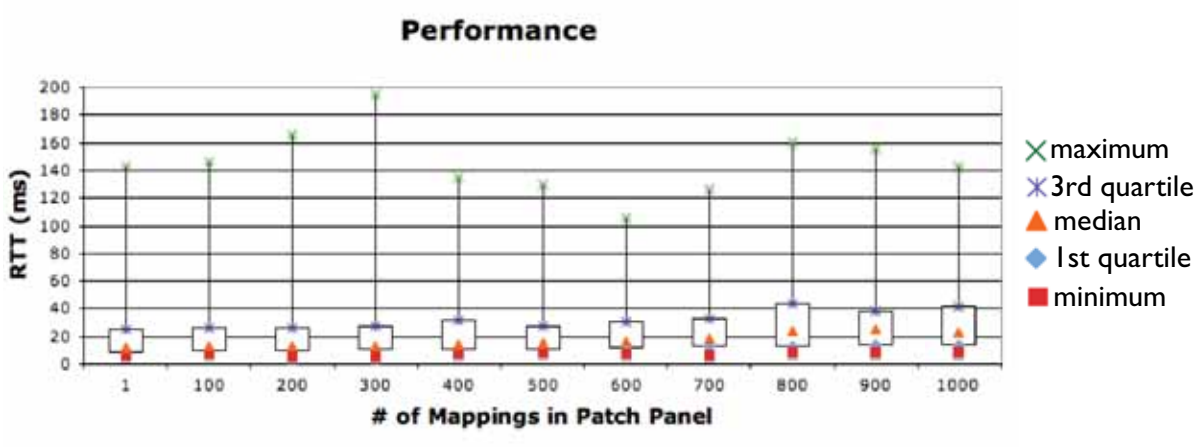


Figure 3.38: Benchmark measurements and example performance degradation for combined Event Heap and Patch Panel round trip time. Note the periodic delay spikes indicated by the large positive skew for each group of measurements.

where I_D represents the index of difficulty of the pointing task, which can be interpreted as a measure of the average number of movement corrections required to acquire a target. C_1 and C_2 are empirically determined constants that vary with input device. A slightly modified version of Fitts' law that incorporates lag has been proposed (MacKenzie and Ware, 1993; Ware and Balakrishnan, 1994) for the mean task performance time:

Modified Fitts' Law incorporates lag

$$MeanTime = C_1 + C_2(C_3 + MachineLag)I_D$$

where C_3 represents the human processing time required to make a corrective movement, $MachineLag$ represents the system processing time, $C_2 I_D$ represents the average number of iterations of the human control loop, and C_1 represents the sum of the initial response time and the time required to confirm the acquisition of the target. This version of Fitts' law was applied to 3D virtual reality systems and experimental measurements revealed a lag target of less than 50ms for input devices (Ware and Balakrishnan, 1994). This threshold for 3D virtual reality systems can serve as approximate threshold for the lag in ubiquitous computing. Based on our benchmarks, our system is below this critical threshold on average. However, the high variance of the latency in our system or performance degradation under system load can push system latency above this threshold. Also, variance in latency impedes usability further by making the input device seem more erratic. Thus our system architecture, in its current implementation, is borderline for fluid pointer manipulation.

Lag target for ubicomp should be less than 50ms

3.6 Conclusions

The quality of a user interface tends to increase with more iterations in the design process, motivating the need for rapid prototyping solutions. The iStuff framework greatly simplifies the exploration of new physical and mobile phone interactions in ubiquitous computing environments from very

Deployable systems for Interactive Workspaces

early stage prototypes all the way to a deployable system for interactive workspaces. For mobile scenarios, the toolkit is prototype-only because its compound prototype architecture and spatial proximity restriction are not acceptable for deployment and make user studies "in the wild" difficult to perform. However, Kjeldskov and Graham (2003) show in a review of mobile evaluations that controlled user studies, similar to those supported by iStuff Mobile, are common in the mobile domain, and useful for identifying critical interaction problems. In a later paper, Kjeldskov and Stage (2004) noted that mobile field experiments are expensive and found that laboratory tests approximating field use of mobile systems identified usability problems at a lower cost.

Prototype-only for mobile phones

In this chapter, we have demonstrated how the iStuff Toolkit architecture can be used to easily recreate seminal ubiquitous computing interfaces, and to significantly simplify the creation of novel ubiquitous computing scenarios. Our evaluation shows that the visual programming environment allows prototypes to be built faster and encourages more design iterations to be performed. System performance is also shown to be adequate for human–computer interaction in ubiquitous computing. By making this toolkit available as open source[25] to the interaction design and research community, we hope to advance the pace of innovation and improve the quality of interface designs in ubiquitous computing.

Radically simplifies prototype creation for ubicomp

This also concludes Chapter 3 of this work, in which we introduced a layered architecture model to support prototyping ubiquitous computing user interfaces. Using this model, we introduced the different layers of functionality of the iStuff Toolkit architecture. The toolkit architecture was demonstrated to have a low threshold for prototyping, and a high ceiling for the prototypes that can be realized.

In the next chapter, we will transition to supporting the evaluation process. Here we will discuss how to more effectively evaluate the prototype input devices created using the toolkits presented in this chapter.

[25] `http://istuff.berlios.de`

Chapter 4

Supporting Evaluation: Expressiveness as an Evaluation Tool for HCI

As human–computer interaction extends beyond the desktop, the need emerges for new input devices and interaction techniques. However, many novel interaction techniques must be prototyped in a proof-of-concept form, using toolkits such as iStuff or iStuff Mobile introduced in the last chapter. Such prototypes can suffer from low *expressiveness*: their ability to convey the intended meaning is limited due to their technical immaturity. Currently, we are lacking the conceptual frameworks to properly assess input devices with low expressiveness. This chapter fills the gap by presenting a conceptual framework that clearly characterizes the *expressiveness* of input devices allowing for more objective comparison with advanced devices. With this definition, designers will also be able to match the expressiveness of the input device to the user interface to optimize comfort and performance of pointing tasks.

Prototype input technologies can suffer from low expressiveness

4.1 Motivation

The inspiration behind this work is the *Sweep* technique (Ballagas *et al.*, 2005) that uses the camera on a mobile phone to detect motion of the handset in several dimensions. This technique enables many direct manipulation interactions with large public displays, including cursor control. Currently, the technique suffers from low sampling rates and resolution because of the limitations of current mobile processors and cameras. With future improvements in mobile processing capabilities, the resolution and sampling rate will improve. However, we want to develop applications that can be used today that still provide a fluid user experience. (Note that this problem also occurs when someone uses standard desktop input devices with a very high resolution display.)

Example: Sweep with modern mobile phones

4.2 Background

Definition: Expressiveness

The expressiveness of input devices was first defined by Card *et al.* (1991) as an evaluation criterion capturing how well the input conveys the intended meaning. Without explicit guards, a mismatch of expressiveness can cause problems in the user interface. These problems were formally described using parameters of input devices. The **In** parameter represents the input domain, which describes the range of values that can be expressed in the physical world. The **Out** parameter represents the output domain set of the input device, which describes the logical values that an input device can produce.

> "In the design of input devices, an expressiveness problem arises when the number of elements in the **Out** set does not match the number of elements in the **In** set to which it is connected. If the projection of the **Out** set includes elements that are not in the **In** set, the user can specify illegal values; and if the **In** set includes values that are not in the projection, the user cannot specify legal values." (Card *et al.*, 1991)

Example: Simple radio

To clarify, consider Card *et al.*'s original example of a simple radio control panel in Figure 4.1. For the selector knob, the **In** set is the continuous set from 0-90 degrees, which corresponds to the range of knob positions that are physically possible. The **Out** set includes <0,45,90>, which are the only possible values that this knob can produce. Here there is an expressiveness mismatch between the input and output domains, which can allow the user to specify illegal values if explicit guards are not taken. One simple way to guard against this particular problem is to use a resolution function that rounds the value from the **In** set to the nearest valid output.

Example: Touch Panel

For a different example of an expressiveness problem, consider a touch panel overlaying a display where the resolution of the touch panel (where transducer values form the **Out** set) is much less than the resolution of the display (where pixels form the **In** set). If a user wanted to select an individual pixel, he would not be able to express that request exactly. This example is straightforward because the touch panel is an absolute input device, making the **Out** set fixed. However for relative input devices, such as a mouse, mathematically characterizing the expressiveness is more complex and has yet to be clearly addressed in the literature. One may be tempted to argue that the expressiveness of any relative input device is infinite, since any value can be specified given enough time. This definition is flawed because it inherently means that all relative input devices are equivalent (and superior to absolute input devices) in terms of expressiveness, preventing us from using it as a metric to characterize and compare different devices.

The expressiveness problem is closely related to the problem of device precision. Pointing precision characterizes how small of a target can be conveniently selected with the device. Quantifying the pointing precision

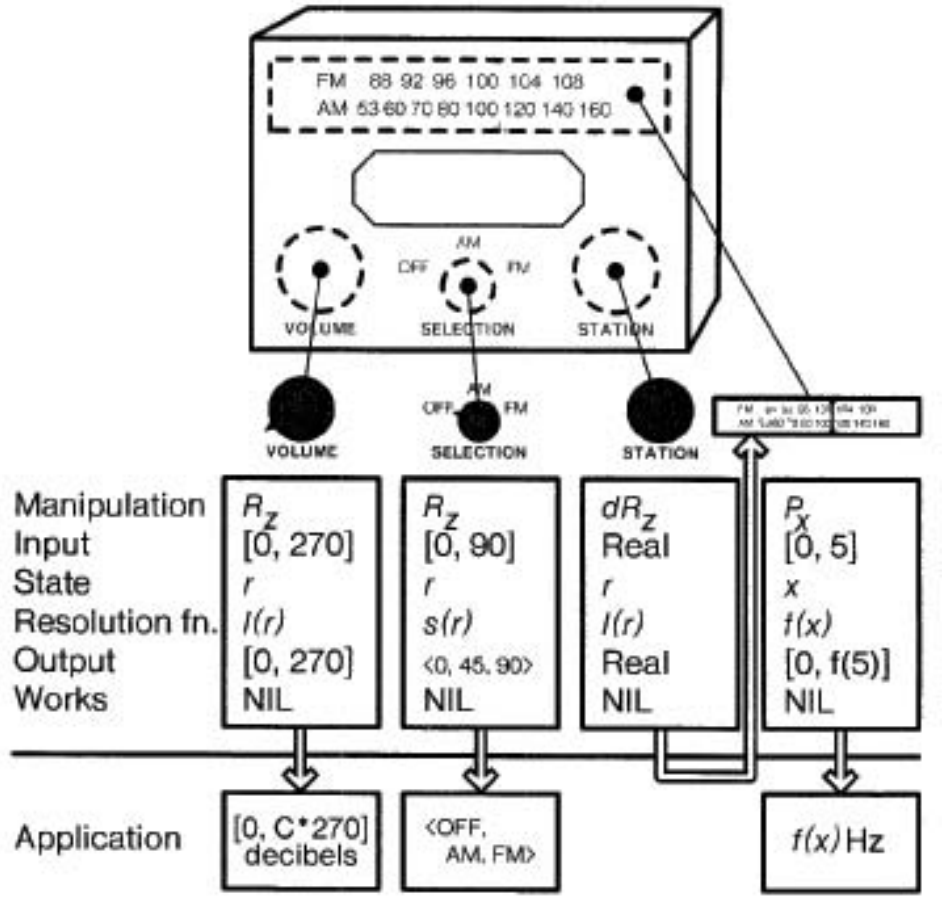


Figure 4.1: Analysis of a simple radio from (Card *et al.*, 1991). Two rotation devices are connected directly to the application. The third rotational device is connected to a positional device, which is then connected to the application.

of absolute input devices in terms of screen area is relatively straightforward.However, quantifying the precision of relative pointing devices is more difficult. Card *et al.* (1991) quantify precision of input devices in terms of bits using insights from subjective ratings of the difficulty of pointing tasks using the mouse in text editing applications, where the threshold between easy and hard tasks lies between selecting a word and selecting a character; selecting a word is the *hardest easy task*, and selecting a character is the *easiest hard task*. Thus, device precision is defined by Card *et al.* (1991) as follows:

Expressiveness is closely related to device precision

> "We characterize the precision of a device as the *ID* that requires the same amount of time as the easiest hard task of the mouse." (Card *et al.*, 1991)

Here, *ID* is the Fitts' law index of difficulty of the pointing task measured in bits. The problem with this definition is that it requires empirical testing to determine the precision of the device, limiting its utility as a design tool.

Empirical testing limits value as design tool

In this chapter, a new definition to quantify the precision of relative input devices based on device parameters is proposed, and it is demonstrated how this definition can be used to predict changes in precision without empirical testing.

4.2.1 Expressiveness of Relative Pointing Devices

The expressiveness of a relative pointing device can be characterized using the precision of the input technique based on models of human motor performance including Fitts' law (Fitts, 1954; MacKenzie, 1992), and the linear speed-accuracy tradeoff (Schmidt *et al.*, 1979). The reasoning behind our definition can be best explained using the conceptual framework presented by Meyer *et al.* (1990), which was used to create the stochastic optimized submovement model. This model attempts to reconcile the strengths of Fitts' Law (better suited to model spatially constrained tasks, such as cursor positioning using a mouse), and the linear speed-accuracy tradeoff (Schmidt *et al.*, 1979) (better suited to model temporally constrained tasks, such as cursor positioning using a joystick to control cursor velocity) into a unified model capable of expressing a wider range of movement tasks. The stochastic optimized submovement model is based on the assumption that the subject attempts to hit the center of the target region with their first submovement (see Fig. 4.2). If the primary submovement successfully acquires the target, then the action terminates. The model anticipates noise in the motor system to affect the primary submovement, causing a slight variation from the intended movement and the actual result. If a miss occurs, then a secondary corrective submovement will be used, and so on. Thus, for a pointing task to be valid for this model, the input device *must* theoretically allow a subject to reach a target in the primary submovement, even though noise in the motor system may require additional submovements before the target is successfully acquired.

Users attempt to hit a target in the first submovement

> We define the *expressiveness* of a relative pointing device as the number of distinct positions a user can theoretically express in a single submovement.

The duration of a submovement is defined by the basic human reaction time.Using the human processor model (Card *et al.*, 1983), the basic reaction time is approximated by $T_{sub} = \tau_p + \tau_c + \tau_m$, or one cycle for each of the perceptual, cognitive, and motor processors. This equation approximates the time for the user to observe the motion progress (τ_p), decide on a correction (τ_c), and perform the correction (τ_m). For the average user ("middleman" in Card *et al.* (1983)), this sum would result in an approximate submovement duration of 240ms.

Submovement duration comes from Psychology literature

We define the expressiveness of a relative input device in terms of the motion throughput of the transducer (dX in dots per sec.), and the psychological

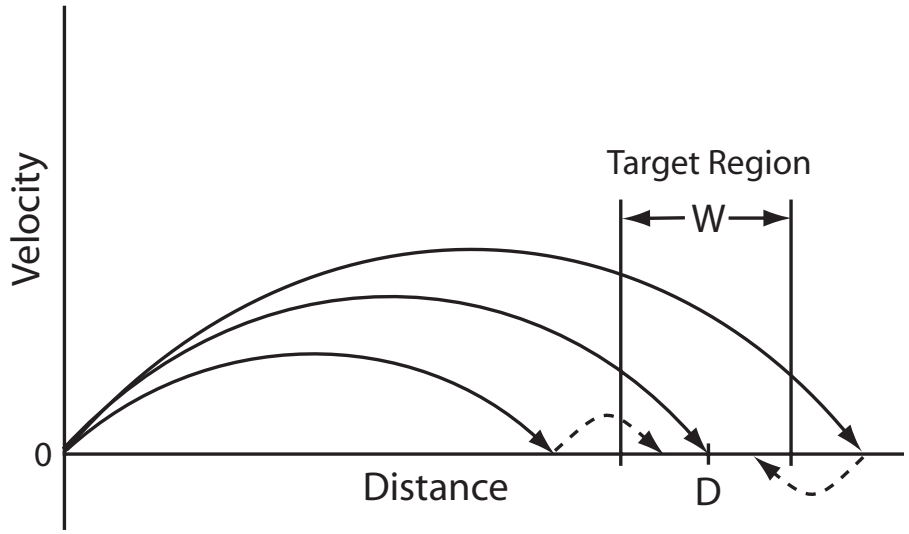


Figure 4.2: The optimized dual-submovement model is a variation of the optimized submovement model with two submovements. Hypothetical primary submovements are marked with a solid line, secondary submovements with a dashed line. (Based on Figure 6.8 from Meyer *et al.* (1990).)

limits of human information processing in terms of the submovement duration (T_{sub} in sec.). Motion throughput describes the rate at which motion information can be processed by the input device. Industry typically reports the device specifications in terms of device resolution (R in dots per inch) and maximum supported rate of motion (v in inches per sec.). We can use these parameters to define motion throughput as follows:

Expressiveness in terms of the physical properties of the device

$$dX := R * v \tag{4.1}$$

$$\left[\frac{\texttt{dots}}{\texttt{sec}}\right] = \left[\frac{\texttt{dots}}{\texttt{inch}}\right] * \left[\frac{\texttt{inches}}{\texttt{sec}}\right]$$

Alternatively, the motion throughput can be defined using the transducer resolution (N in dots per sample) and sampling frequency (f_{sample} in samples per sec.), assuming a single transducer per axis of motion. To simplify the definition, we first define the notion of sample reach length (L_{sample} in dots per sample) as the maximum distance that can be reached in a single sample.

$$L_{sample} := .5 * (N - 1) \tag{4.2}$$

The scaling factor (.5) is necessary because the transducer resolution is split into positive and negative values along the motion axis. One sample is removed to account for the zero value which has no motion. If positive and negative measurements are separated into multiple transducers, this equation may require slight adjustments.

The maximum supported rate of motion (v) can then be defined in terms of sample reach (L_{sample}), sampling frequency (f_{sample}), and device resolution (R) as follows:

$$v := f_{sample} * L_{sample}/R \tag{4.3}$$
$$\left[\frac{\texttt{inches}}{\texttt{sec}}\right] = \left[\frac{\texttt{samples}}{\texttt{sec}}\right] * \left[\frac{\texttt{dots}}{\texttt{sample}}\right] / \left[\frac{\texttt{dots}}{\texttt{inch}}\right]$$

The motion throughput (dX) can then be alternatively formulated by combining Equations 4.1 and 4.3,

$$\begin{aligned} dX &= R * v \\ dX &= R * f_{sample} * L_{sample}/R \\ dX &= f_{sample} * L_{sample} \end{aligned} \tag{4.4}$$
$$\left[\frac{\texttt{dots}}{\texttt{sec}}\right] = \left[\frac{\texttt{samples}}{\texttt{sec}}\right] * \left[\frac{\texttt{dots}}{\texttt{sample}}\right]$$

We further define the submovement reach (L_{sub} in dots per submovement) as the maximum distance that can be reached during a single submovement (T_{sub} in sec. per submovement).

Definition: Submovement reach

$$L_{sub} := dX * T_{sub} \tag{4.5}$$
$$\left[\frac{\texttt{dots}}{\texttt{submovement}}\right] = \left[\frac{\texttt{dots}}{\texttt{sec}}\right] * \left[\frac{\texttt{sec}}{\texttt{submovement}}\right]$$

We note that L_{sub} is measured in dots per submovement, which is a unitless quantity relating to the number of distinct values that can be expressed in a single submovement.

The expressiveness *set* ($\mathbb{E}$) can then be defined as the set of points that a user can theoretically express in a single submovement. For a one-dimensional input device, assuming a C–D gain function of $S(t)$ (i.e., cursor acceleration), we can define its expressiveness set as follows:

Definition: Expressiveness set

$$\mathbb{E} := \{S(x) : \text{where } 0 < x < L_{sub} \text{ and } x \in \mathbb{Z}\},$$

where $\mathbb{Z}$ is the set of all integers. We note that in the case of no C–D gain ($S(t) = t$), this reduces to the set of integers between 0 and L_{sub}, and the cardinality of the expressiveness set ($|\mathbb{E}|$) is $L_{sub} + 1$.

Increasing C–D gain does not increase expressiveness

An input device is said to have higher expressiveness when the cardinality of the expressiveness set ($|\mathbb{E}|$) is greater than that of another device. We also note that changing the C–D gain function from $S(t) = t$ cannot increase the cardinality of the expressiveness set; instead, it may create unreachable gaps or even decrease expressiveness if multiple transducer values map to the same cursor displacement. This theoretical observation fits with the findings of Jellinek and Card (1990) who explored second order cursor acceleration using several different C–D ratios on a mouse, but found no performance improvements.

Figure 4.3: An optical-mechanical mouse: (1) Motion across the desktop surface moves the ball. (2) Grips transfer the ball movement to turn (3) optical encoding disks. (4) Infrared LEDs shine through the holes. (5) Infrared sensors accumulate light pulses and convert them into motion along the X and Y axes. (Source: Wikipedia)

4.2.2 Examples

A case study of specific input devices should serve to illustrate the theoretical definition with practical examples. They show how expressiveness can be used as a design metric to gauge the suitability of a relative input device for a particular interaction scenario.

Opto-Mechanical Mouse

Example: Opto-mechanical mouse

Figure 4.3 shows the inner workings of an optical-mechanical mouse. The device resolution (R in dots per inch) of the mouse is determined by the ratios of the physical dimensions of the ball, the grips, and the discs. The current industry standard for device resolution of these mice ranges from 400 to 800 dpi. For this example, assume that the pulse accumulator is capable of storing 8 bits of information for each dimension of motion, resulting in a transducer resolution (N) of 256 possible dots (positions) per sample.

For traditional PS/2 mice, common in the 1990s, the default sampling rate (f_{sample}) under Windows 95 / 98 was 40 Hz. Thus, their motion throughput

(dX) can be calculated by combining Equations 4.2 and 4.4 as follows:

$$\begin{aligned} dX &= L_{sample} * f_{sample} \\ &= (N-1) * .5 * f_{sample} \\ &= (256-1) * .5 * 40 \\ &= 5100 \left[\frac{\texttt{dots}}{\texttt{sec}}\right] \end{aligned}$$

Then their submovement reach (L_{sub}) follows as:

$$\begin{aligned} L_{sub} &= dX * T_{sub} \\ &= 5100 * .240 \\ &= 1224 \left[\frac{\texttt{dots}}{\texttt{submovement}}\right] \end{aligned}$$

Thus a user would start to have problems with expressiveness using this device with a screen resolution of 1225 dots or higher in any dimension. These problems would manifest themselves as consistently undershooting targets beyond the submovement reach with the primary submovement.

Optical Mouse

Example: Optical mouse

As another example, consider the Agilent Technologies (2004) ADNS2610 optical mouse sensor. It supports a device resolution of 400 dpi, and rates of motion of up to 12 inches per second. Thus, following Equation 4.1 its throughput can be calculated as follows:

$$\begin{aligned} dX &= R * v \\ &= 400 * 12 \\ &= 4800 \left[\frac{\texttt{dots}}{\texttt{sec}}\right] \end{aligned}$$

Then its submovement reach (L_{sub}) follows as:

$$\begin{aligned} L_{sub} &= dX * T_{sub} \\ &= 4800 * .240 \\ &= 1152 \left[\frac{\texttt{dots}}{\texttt{submovement}}\right] \end{aligned}$$

Thus a user would start to have problems with expressiveness using this device with a screen resolution higher than 1152 dots in any dimension. Note that if the resolution of the sensor was bumped to 800 dpi (as with most more modern optical mice), then L_{sub} also doubles, supporting resolutions much higher than 2000 pixels in either dimension.

Analog Joystick

Consider a USB analog joystick (shown in Figure 4.4) that has a tranducer resolution (N) of 256 distinct dots per sample on each axis after digital

Figure 4.4: An analog joystick measures absolute tilt of the stick in the rX, rY dimensions.

Example: Joystick

conversion. The joystick measures absolute tilt in the (rX, rY) dimensions, but is used as a relative positioning device by mapping tilt to control the velocity of cursor movement. Positioning a cursor with velocity control is a temporally constrained task. Modern operating systems support sampling such USB devices at a rate of 125 Hz. Thus the rate of motion dX can be expressed as follows:

$$\begin{aligned} dX &= (N-1) * .5 * f_{sample} \\ &= (256-1) * .5 * 125 \\ &= 15937.5 \left[\frac{\texttt{dots}}{\texttt{sec}}\right] \end{aligned}$$

Then the resulting submovement reach (L_{sub}) follows as:

$$\begin{aligned} L_{sub} &= dX * T_{sub} \\ &= 15937.5 * .240 \\ &= 3825 \left[\frac{\texttt{dots}}{\texttt{submovement}}\right] \end{aligned}$$

Higher expressiveness is not necessarily higher efficiency

This indicates that this particular joystick has a very high expressiveness, notably higher than the mice examined above primarily because of the higher sampling rate. It should be noted that expressiveness does not necessarily indicate that a device can be used with a higher efficiency (Card *et al.*, 1991). Instead, it indicates the boundary where efficiency will start to decrease, because users will consistently undershoot targets further than the submovement reach during their first submovement. To draw conclusions about the normal pointing efficiency (or device bandwidth), it is still necessary to compare the devices using an ISO 9241-9 (ISO, 2000) standard empirical evaluation.

Sweep

Example: Sweep

Sweep (Ballagas *et al.*, 2005) is an experimental input technique that uses the camera on a mobile phone to detect relative motion of the phone. The interaction is intended to support novel interactive applications for large public displays. The current implementation of this technique detects relative motion in the (X, Y) dimensions with a sample frequency (f_{sample}) of 12.5 Hz and a transducer resolution (N) capable of detecting 9 distinct dots (displacements) per sample. The actual physical sensor in this example is the camera which has a relatively high resolution. However, for the purposes of expressiveness we consider the output of the motion detection algorithm to be the transducer output since those are the actual values that can affect the cursor. Thus,

$$\begin{aligned} dX &= (N-1) * .5 * f_{sample} \\ &= (9-1) * .5 * 12.5 \\ &= 50 \left[\frac{\texttt{dots}}{\texttt{sec}} \right] \end{aligned}$$

Then the submovement reach (L_{sub}) follows as:

$$\begin{aligned} L_{sub} &= dX * T_{sub} \\ &= 50 * .240 \\ &= 12 \left[\frac{\texttt{dots}}{\texttt{submovement}} \right] \end{aligned}$$

Prototype input techniques suffer from low expressiveness

Thus the largest resolution that would not result in expressiveness problems would be 13×13. This means that the use of this input device with standard desktop resolution will result in difficulties because of the severe mismatch in expressiveness. This matches the difficulties expressed in a previous evaluation of the Sweep technique (Ballagas *et al.*, 2005).

4.2.3 Expressiveness of Absolute Pointing Devices

Example: Tablet

For direct surface interaction, such as a touch screen, the expressiveness $|\mathbb{E}|$ is simply defined as the resolution of the input surface (dpi) multiplied by its size along each dimension. For example, the high resolution Wacom Cintiq 21UX tablet[1] reports a input resolution of 5080 locations per inch with an active area of 17" $\times$ 12.75". This results in an expressiveness of $86,360 \times 64,770$ selexels. The submovement reach can be limited by the physical characteristics of the relevant parts of the human body, such as arm length, with larger form factors.

The example illustrates an instance where the input device expressiveness $(86,360 \times 64,770)$ is much higher than the expressiveness of the display (1600×1200). Typically in this case, software is used to reduce the input expressiveness so that it is matched to the output expressiveness.

[1] `http://www.wacom.com/cintiq/index.cfm`

4.3 Selexels: Using Expressiveness as a Design Tool

There are several scenarios where expressiveness can inform the design of an interactive system.

1. **The display resolution is specified, and the input device can be chosen**. In this scenario, the input device can be selected to have an expressiveness that matches or exceeds the display resolution to achieve best user performance.
2. **The input device is specified, and the display resolution can be chosen**. In this scenario, the display can be selected to have a resolution the matches or is less than the input device expressiveness. This solution assumes that reducing the display resolution can be compensated for in the UI design such that it has no effect on the quality of the interface.
3. **The input device and display resolution are specified, but the expressiveness of the input device is less than the resolution of the display**. In this case, an interesting option remains: to match the selection resolution of the user interface to the expressiveness of the input device without sacrificing display resolution. This can be accomplished using "selexels" (see below).

4.3.1 The Selexel Approach

The traditional conceptual framework for analyzing pointing tasks separates the task into two spaces: motor space and display (or visual) space. This is reflected by the frequent use of the control–display (C–D) ratio to describe the relationship between motion distance in the physical world (meters) and motion distance on the screen (pixels). Our new conceptual framework adds selection space as a level of indirection between motor space and display space. Using this framework, a traditional desktop interface is a special case where the selection space is identical to the display space. We note that since the motor space is mapped to selection space and not display space, the concept of C–D ratio is replaced by the notions of Control–Selection (C–S) ratio for the relationship between motor and selection space, and Selection–Display (S–D) ratio for the relationship between selection and display space.

Definition: selection space

The selection resolution of the user interface can be matched to the expressiveness of the input device by dividing the screen into atomic selectable elements, or *selexels*, with a resolution that is independent of the pixel resolution of the screen (see Fig. 4.5). By separating selexels from pixels, the range of motion in the interface is adjusted to support a smooth and fluid user experience for the input device in use, while preserving the screen resolution and information capacity of the display.

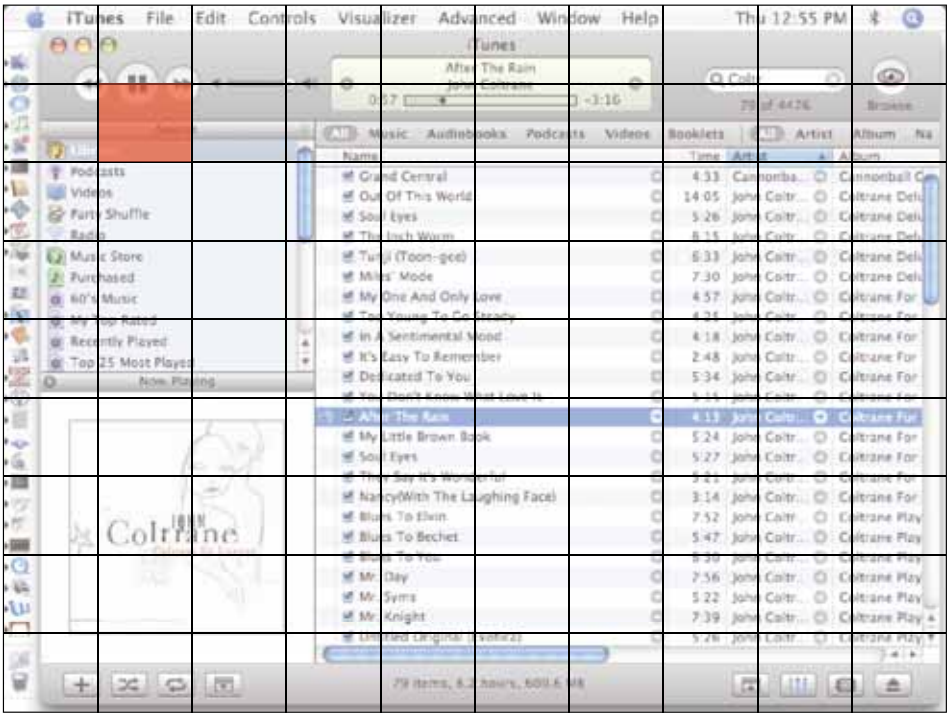


Figure 4.5: A sample selexels screen division over a typical desktop interface. It indicates that existing desktop interfaces may need to be modified to disambiguate selection when using a low resolution selection space.

Definition: *Selexel*

Selexel:
An atomic selectable element in Selection space, analogous to a "pixel" or picture element in Display space. The resolution of Selection space is independent of display resolution.

Expressiveness is different from resolution

Expressiveness is easily confused with the notion of device resolution. Resolution of an input device, usually measured in dpi (dots per inch), describes precision in motor space; how small of a movement, in motor space, can be distinguished by the transducer. Expressiveness, characterized by selexels (unitless), describes precision in selection space; how many distinct positions, in selection space, can one theoretically express, or "reach", using this input device in a given timeframe.

4.3.2 Practical Application of Selexels

Mismatches in expressiveness can disrupt the user exprience. Our new method of characterizing expressiveness allows us to identify and resolve these mismatches to provide a more fluid experience. Selexels provide a level of abstraction that allows us to reduce the selection resolution of the display to match the expressiveness of the input device, without sacrificing valuable display resolution.

Usage Scenario

We may apply these concepts to interactions in the domain of large public displays, as illlustrated in the following scenario.

> While Hans is waiting for his train to Berlin, he notices a large public display on the platform displaying advertisements and community news. He recognizes the style of a 2D bar code next to the display. He has previously used a similar barcode to initiate interaction with a demo display in the T-Mobile showroom. He takes a picture of the code (just as he had done in the showroom), and his phone automatically connects wirelessly to the display via Bluetooth and transfers information characterizing its input expressiveness. The advertisement transforms into an interactive menu with a selexel resolution matched to his input device, allowing Hans to select from several options including: browsing the news, checking the weather, or even contributing to an interactive community bulletin board where people post images, video, and text. Hans waves his phone through the air using the Sweep technique to select the weather option. A map of Germany appears and as he navigates the cursor over different regions, the current conditions and forecast are displayed to the right of the map. It looks like the previously forecast afternoon showers are no longer a threat in Berlin. Hans then calls his wife to invite her to go to their favorite restaurant for dinner on the quaint cobblestone patio.
>
> Shortly after Hans leaves, Maria wants to check up on her bulletin board post from last week about the proposed new theatre to be built next to the train station. Her phone is much older than Hans' and the expressiveness of the Sweep technique is much lower. As she connects to the display, the cursor grows and the menu options move further apart to adapt the selexel resolution of the interface to match that of her phone. Now she is able to use her more limited phone as a pointing device, and yet the system still provides a fluid experience.

Example: adapting UI to match the capabilities of the Sweep technique

This scenario shows how expressiveness can be used to tailor a user interface on the fly to the capabilities of the input devices that are used in a particular interaction. These concepts may also be useful for scenarios using very high resolution displays such as those described by Czerwinski *et al.* (2006).

4.3.3 Comparing Selexels to Other Selection Techniques

There are many approaches to improving selection performance in pointing tasks that increase the size of the cursor. The *area cursor* has an active selection region that spans a screen area, rather than a single point.

Selexels vs. Area Cursors

Kabbash and Buxton (1995) show that selection of a point-sized target with an area cursor can be accurately modeled using Fitts' law by setting W to the cursor width, reducing the index of difficulty for small targets. However, area cursors can overlap multiple targets, making user selection ambiguous. Worden *et al.* (1997) propose an improved area cursor with an additional hot spot at its center to disambiguate between multiple closely-spaced targets within the cursor area. This cursor performs better than the standard point cursor when targets are far apart, and identically to point cursors when targets are closer together. The Bubble cursor (Grossman and Balakrishnan, 2005) dynamically resizes the active cursor region to always encompass exactly one selectable object that is nearest to the center position of the cursor. This effectively changes the width of the target to the size of the Voronoi region surrounding the target, maximizing its size in motor space. This technique is superior to previous techniques in that it also demonstrates benefits over the point cursor for densely packed targets. However, for each of these solutions, cursor motion is governed by pixels, making them unsuitable for tasks for input devices with low expressiveness.

A cursor in the selexel domain is similar to these techniques in that it has an active region that spans a screen area, rather than a single pixel. However, a cursor highlighting a single selexel is actually a point cursor in selection space, even though the active region may span a screen region in display space. Thus, the techniques proposed above should result in the same performance improvements if applied to targets in selection space, but we leave the experimental validation of this theory to future work.

Selexels vs. Adaptive C–D gain

Several selection techniques have been shown to increase pointing performance by dynamically adjusting the control–display (C–D) ratio (Blanch *et al.*, 2004; Baudisch *et al.*, 2005), sometimes referred to as cursor acceleration. Pointing performance can be improved by increasing the control–display ratio while approaching the target (and thereby decreasing the distance traveled in motor space), or by decreasing the control–display ratio while inside the target (and thereby increasing the target size in motor space). In this paper, we focus on a static C–S ratio. However, these adaptive techniques should still be applicable in the selexels domain by replacing the C–D ratio with the C–S ratio, but again we leave the validation of this theory to future work.

Eliminating selection ambiguity

A selexel cursor has the same ambiguity problem as the area cursor when covering multiple targets, as demonstrated in Fig. 4.5. There are several options to eliminate this ambiguity:

Resrict input device

- The input device could be restricted to one that supports the minimum selexel resolution required for the interface to have no cursor overlap of multiple targets. In this way, the selexels framework allows us to define device criteria for interacting with existing applications. However, most desktop applications are not compatible with a grid selection scheme, requiring the standard "one selexel per pixel" scenario.

- Alternatively, if a situation requires the use of a low precision pointing device (e.g., in the aforementioned case of large public display interactions using the Sweep technique), the layout of the interface could be altered to be compatible with the selexel resolution of the input device. This can be accomplished through manual rearrangement, or through automatic layout of user interfaces as in Gajos and Weld (2004). In this way, the selexels framework allows us to define graphical layout constraints for applications designed for low-expressiveness pointing techniques. (See Future Work for more discussion on this topic.)

Restrict to one selectable UI object per selexel

- Lastly, drawing inspiration from the Bubble Cursor (Grossman and Balakrishnan, 2005), the size of the selexel in pixels (the S–D ratio) can be dynamically varied to always contain one and only one selectable target, essentially mapping each selexel to the Voronoi regions of the targets on the display, and thereby minimizing the number of selexels in an interface. A simplified 1D version of this approach is embodied by the tabbed interface, where a user can cycle the selection focus through the selectable items in an interface, often by pressing the tab key. The problem with this symbolic and irregularly spaced approach is that the resulting C–D ratio (a combination of C–S and S–D ratios) might be unpredictable for users, preventing them from accurately planning their movements. In this paper, we focus on the case of a constant S–D ratio.

Adapt selexel size to Voronoi regions around selectable objects

4.4 Evaluation

Card, English, and Burr's (Card *et al.*, 1978) seminal work showed that the *efficiency* of pointing devices can be analyzed using Fitts' law (Fitts, 1954; MacKenzie, 1992), which models human motor performance by predicting movement time in a pointing task as follows:

$$MT = a + b * ID$$
$$ID = \log_2(\frac{D}{W} + 1),$$

where D is the target distance, W is the target width, and ID is the index of difficulty of the pointing task. a and b are empirically determined constants that vary with the characteristics of the input device. These empirically determined constants are affected by a wide variety of input device characteristics including mass, friction, resolution, sampling rate, lag, and C–D gain (MacKenzie, 1992). Changes in any of these parameters will also change the result of the regression analysis.

In a series of experiments, we attempt to characterize how pointing tasks are affected by the relationship between the expressiveness of the input device and the selexel resolution of the display. A custom test program allows us to vary the selexel resolution of the display, and vary the expressiveness of input devices by artificially limiting the resolution and sampling rate.

The test application hides the system cursor and displays a point cursor in selexel space (an area cursor in pixel space) that moves selexel-wise, appearing to jump from one selexel position to the next. The application was implemented in Objective-C under Mac OS X.

4.5 Experiment 1

Can pointing in selection space be modeled by Fitts' law?

Given that the selexel cursor represents a point cursor in selexel space, Fitts' Law (Fitts, 1954) should be a suitable model for predicting target aquisition time in pointing tasks. However, the selexel cursor is an area cursor in pixel space. Previous studies of area cursors have demonstrated that selection using area cursors lowers the index of difficulty for smaller targets (Kabbash and Buxton, 1995; Worden *et al.*, 1997). Also, as users have come to expect pixel-wise cursor motion, the selexel-wise motion may impede or annoy the user. Thus, it is important to examine whether pointing in the selexel paradigm can be modeled by Fitts' Law. This experiment is specifically designed to examine if the input device expressiveness affects the empirically determined constants of the input device (a and b).

4.5.1 User Study Design

Horizontal tapping test

Using a within-groups design, users were asked to complete a horizontal tapping test based on ISO 9241-9 (ISO, 2000), but modified to accommodate selexel pointing. For each test condition, the same range of target distances ($D = 256$, 640, and 1024 pixels) and target widths ($W = 64$ and 128 pixels) were used.

Matched expressiveness conditions

To simulate input devices of varying expressiveness, the sampling period of the mouse was artificially increased to 20ms to guarantee a constant sampling rate for all test conditions. The transducer resolution (N) of the input device was varied to have an L_{sample} of 1280, 256, 40, and 20 dots per sample. The selexel resolution of the test UI was designed to match the L_{sample} of the input devices with selexel resolutions of 1280×800, 256×160, 40×25, and 20×13 selexels respectively (selexel sizes of 1×1, 5×5, 32×32, and 64×64 pixels). By matching the selexel resolution of the UI (S–D ratio) to the transducer resolution of the input device (C–S ratio), the resulting C–D ratio remains effectively constant (see Table 4.1) across the different conditions.

Varying selexel block size

The cursors were all displayed as point cursors in selexel space, except for the one selexel per pixel condition. In this condition, a 5×5 pixel area cursor was used instead of a 1×1 pixel point cursor for visibility. This particular cursor still maintained the other properties of a point cursor in pixel space in that it had an active region of 1×1 pixel at the center of the cursor, and its motion was pixel-wise.

C-S ($Reach_{sample}$)	Selexels	Pixels	S-D	C-D
1280	1280	1280	1	1280
256	256	1280	0.2	1280
40	40	1280	0.03125	1280
20	20	1280	0.01563	1280

Table 4.1: By matching the selexel resolution to the C–S ratio (L_{sample}) we maintain a constant C–D ratio across the test conditions.

Constant target placement in display space

The pixel placement for target pairs remained constant across test conditions. However, the pairs of targets were placed such that some were aligned to the selexel boundaries and some weren't (depending on the selexel condition). When a target is not aligned to selexel boundaries, its width may span over several selexels even if its size in pixels is much less than a single selexel.

Ordering varied to control for learning effects

Both the ordering for the different expressiveness conditions, and the ordering for the different *ID*s were varied to reduce learning effects in a within-groups study. A fully crossed design resulted in a total of 24 combinations of D, W, and matched (N, selexel resolution) pairs. For each combination, 25 target selections were required.

4.5.2 Participants

10 volunteers (4 females, 6 males) ranging in age from 22 to 27 participated in the study. All of the participants were students (8 computer science, and 2 political science) from RWTH Aachen University.

4.5.3 Equipment

The tapping test was performed using a Logitech M-BJ58 (800 dpi) USB optical mouse, an Apple 23" Cinema LCD monitor with the display resolution set to 1280×800, and a PowerMac Dual 2.0 GHz G5 computer.

4.5.4 Results

The results from this experiment are shown in Figure 4.6. The graphs indicate that using the distance and width in units of pixels results in a valid ($p < 0.05$), but poor model ($R^2 = 0.71$) for selexel pointing tasks. If the same data is reinterpreted using target distance (D) and target width (W) in units of selexels, the model becomes much improved ($p < 0.005$, $R^2 = 0.959$). An analysis of variance (ANOVA) shows a significant effect

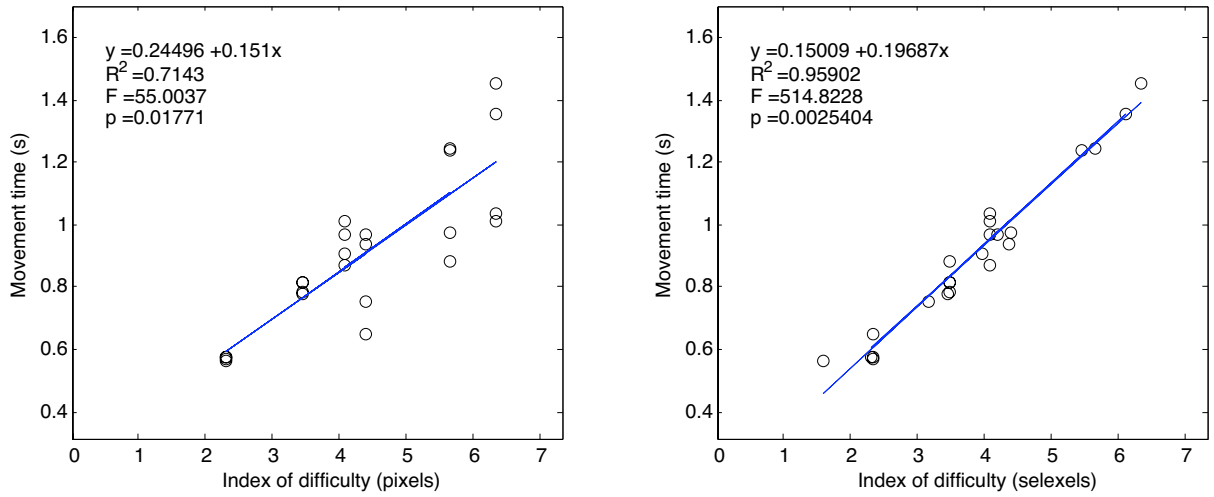


Figure 4.6: (Left) Index of difficulty calculations in terms of display pixels are not well suited for selexel pointing tasks. (Right) The same data reinterpreted with index of difficulty in terms of selexels provides a strong correlation. The correlation is strong despite the fact that this data is mixed across selexel conditions, demonstrating that selexel resolution has no impact on device performance as long as the C–D ratio is preserved. Here the C–D ratio is preserved by matching the expressiveness of the UI to the expressiveness of the input device by changing the selexel resolution.

for index of difficulty [$F(15, 23) = 42.99$, $p < 0.0003$], and no significant effect for the expressiveness conditions [$F(2, 23) = 1.64$, $p < 0.28$].

4.5.5 Discussion

Pointing in selection space can be modeled by Fitts' Law

These results demonstrate that selexel-wise cursor motion, despite being potentially visually disruptive to the user, had no significant effect on task completion time compared to the pixel-wise cursor motion. This is further demonstrated by the fact that the mixed results can be modeled very well ($R^2 = 0.959$) using a single linear regression.

These results also confirm previous findings by Kabbash and Buxton (1995) that show that area cursors lower the index of difficulty for small targets, but these results further show that movement time in selexel pointing tasks for targets larger than one pixel can be accurately predicted using units of selexels to measure target distance and width for the calculation of *ID*.

By matching our the device resolution to the selection resolution of the screen (see Table 4.1), the distance in motor space required to move the cursor across the entire width of the screen in display space was preserved across conditions (although target sizes and distances in selection space slightly varied). Given that, these results support previous findings (MacKenzie, 1991; Blanch *et al.*, 2004; Baudisch *et al.*, 2005) that demonstrate target distance and width in motor space are the key factors for performance time.

A more significant result from this experiment is that, contrary to what one might expect, transducer resolution (N in dots per sample) has no effect on task completion time, as long as the C–D ratio is preserved across test conditions. This result has important implications for evaluating prototype input techniques with low transducer resolution, such as the Sweep technique. As mobile processors and cameras continue to improve, motion detection will become more powerful, and the resulting transducer resolution will improve. Using this study as a model, an evaluation can be structured such that conclusions can be made about pointing performance of future mobile phones as the transducer resolution (dots per sample) continues to rise (assuming all other parameters, such as C–D ratio, remain the same).

Transducer resolution has no effect on performance if C–D is constant

The error rates for pointing under selexels were lower than pointing under pixels. This can be explained using the speed-accuracy trade-off since the target widths effectively expand to the selexel span of the target, reducing the physical accuracy required for the pointing task.

4.6 Experiment 2

To further validate our conceptual framework, it is necessary to experimentally verify the notion of submovement reach (L_{sub}), the maximum distance that can be reached in the first submovement. Based on the conceptual framework, the selexel resolution should match the submovement reach (L_{sub}) preventing any target distances from exceeding the maximum distance. This experiment examines the effect of target distances exceeding the submovement reach.

Targets exceeding submovement reach lower pointing efficiency?

4.6.1 User Study Design

This experiment was structured as a horizontal tapping test, very similar to Experiment 1. It used a within-groups design, and reused the parameters for target distance (D), target width (W), and target placement. However, in this experiment we maintained the transducer resolution (N) to have a constant L_{sample} of 20 dots per sample. With a sample period of 20 ms, the L_{sub} can be calculated as follows:

Constant device resolution

$$
\begin{aligned}
L_{sub} &= dX * T_{sub} \\
L_{sub} &= f_{sample} * L_{sample} * T_{sub} \\
L_{sub} &= (1/.020) * (20) * .240 \\
L_{sub} &= 240 \left[\frac{\texttt{dots}}{\texttt{submovement}} \right]
\end{aligned}
$$

This submovement reach (L_{sub}) of 240 dots per submovement was maintained across all conditions.

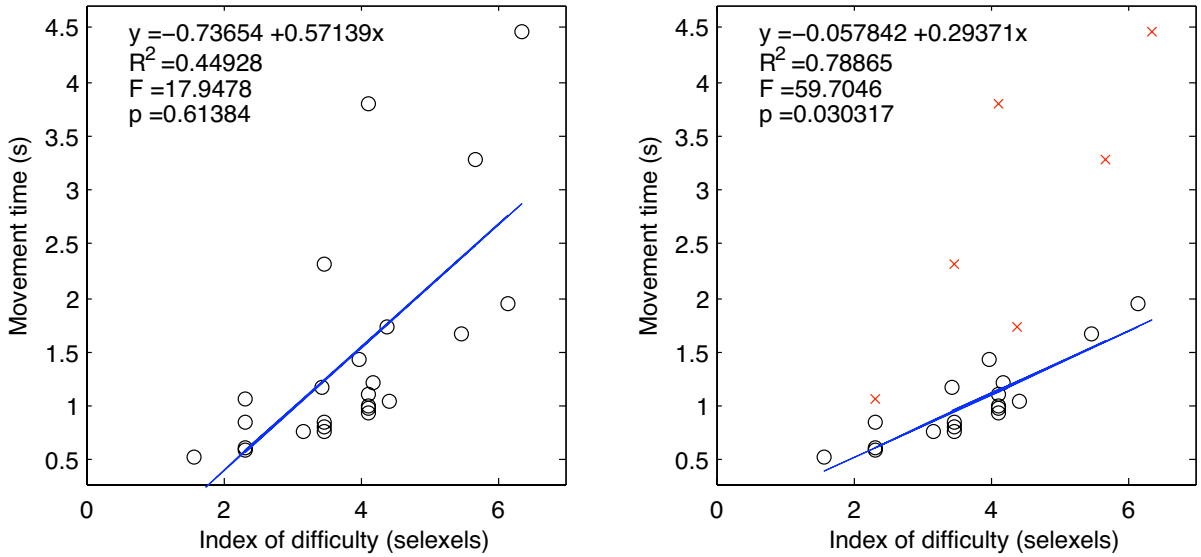


Figure 4.7: (Left) In this experiment, contrary to the previous experiment, the mixed results across experimental conditions cannot be modeled with Fitts' Law. (Right) The same data is shown with target distances greater than submovement reach (L_{sub}) removed (marked as "x"). When the conditions that exceed the submovement reach are removed, the data can be modeled by Fitts' Law ($p < 0.05$).

Varying selexel resolution resulting in varying C–D ratios

The selexel resolution of the display (and resulting pixel size) was varied as in Experiment 1. As a result the C–S ratio remained constant, while the S–D ratio varied, resulting in a range of C–D ratios.

The ordering for the different selexel resolutions, and the ordering for the different *ID*s were varied to reduce learning effects in a within-groups study. A fully crossed design resulted in a total of 24 combinations of D, W, and selexel resolution. For each combination, 25 target selections were required.

4.6.2 Participants

11 volunteers (2 females and 9 males) ranging in age from 22 to 29 participated in the study. All of the participants were students (8 computer science, 1 political science, and 2 undisclosed) from RWTH Aachen University.

4.6.3 Equipment

The equipment used was the same as for Experiment 1.

4.6.4 Results

The results from this experiment are shown in Figure 4.7. In the first graph, all of the conditions are mixed, resulting in data that is unable to be modeled using Fitts' Law ($p = 0.61$). In the second graph, the same data from the first graph is reinterpreted by removing conditions (marked with an "x") where the target distance (D) exceeds the submovement reach (L_{sub}) to allow comparison with the original data. The second graph results in a data set that can be modeled by Fitts' Law ($p < 0.05$). Using an analysis of variance (ANOVA), a binary "reach exceeded" variable has a significant effect [$F(1,23)=28.79$, $p<0.00003$] showing that target distances exceeding submovement reach (L_{sub}) increase task completion time.

4.6.5 Discussion

Targets beyond submovement reach reduce pointing efficiency

In the second graph of Figure 4.7, the conditions where the targets were placed only slightly further (256 selexels) than the submovement reach (240 dots per submovement) are still relatively close to the other samples, indicating that slight mismatches in expressiveness are only a minor issue. However, the extreme mismatches in expressiveness with target distances of 512 and 1024 selexels are extreme outliers in terms of target acquisition time. This indicates that target distances exceeding $Reach_{sub}$ disrupt the user experience and increase task performance time.

Varying C–D conditions decrease the quality of the mixed model

The reader may have noticed that the quality of the input device model ($R^2 = 0.788$) is much lower than that of the previous experiment. This is due to the fact that the data has mixed C–D ratios across the different experimental conditions. As mentioned before, the C–D ratio is one of the input device parameters that is captured by the Fitts' law coefficients (a and b). Separating the different C–D ratio conditions, as in Figure 4.8, should result in models that account for much more of the variance in the data (a higher R^2). Figure 4.8 demonstrates that this is true except for the condition with a selexel size of 1×1, which contains only target distances greater than submovement reach (L_{sub}). This further indicates that an upper bound to the validity of Fitts' Law can be predicted using our definition of input device expressivness.

4.7 Chapter Summary

Expressiveness as a design tool

Previous characterizations of the expressiveness of relative input devices required empirical testing, limiting their utility as a design tool. We have presented a conceptual framework to characterize the expressiveness of input devices based on the physical properties of the hardware, allowing its appropriateness for a particular interaction scenario to be more easily assessed. Our selexel framework allows the user interface to be tailored (even

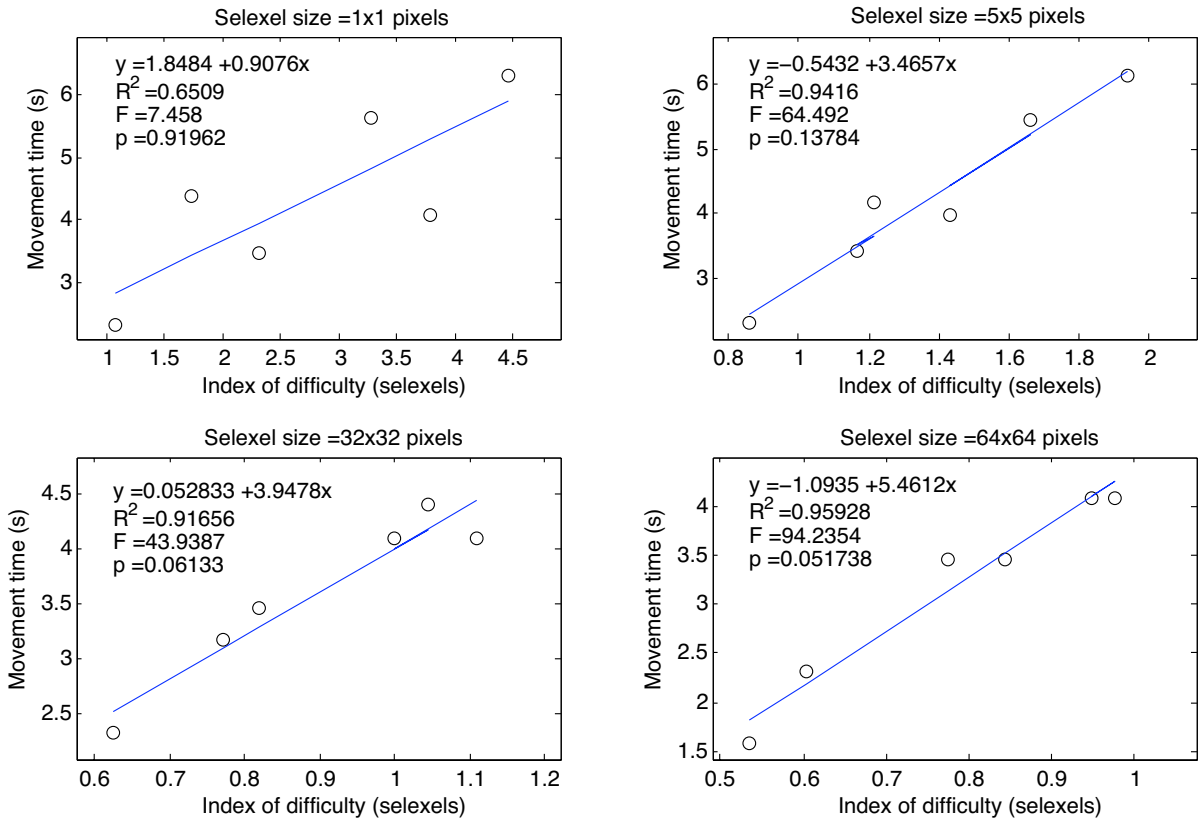


Figure 4.8: The data from Figure 4.7 separated into charts based on C–D ratio results in more accurate models (a higher R^2) for conditions where the targets were within L_{sub}. Note that the condition with target distances greater than L_{sub} (selexel size of 1×1) still poorly correlates with the model despite separation.

adapted at run-time) to match the expressiveness of the input device without sacrificing the screen resolution, which is important to preserve the information output capacity of the display.

Our experiments have shown that pointing under selexels can be modeled by Fitts' Law, demonstrating that selexel-based motion has no effect on task performance time. Experiment 1 further demonstrates that transducer resolution has no effect on task performance time as long as the C–D ratio of the UI is preserved.

In experiment 2, our conceptual framework was validated by experimentally verifying the notion of L_{sub}. These results demonstrate that the upper bound for the validity of the Fitts' Law model can be predicted based on the physical properties of the input device.

Evaluate input devices in prototype form

Using these results, this new conceptual model can be used to structure the evaluation of prototype input devices. The input bandwidth (the slope of the Fitts' Law regression analysis) of a prototype device measured under low expressiveness conditions is a good approximation of the same device with higher sampling rates and resolution under high expressiveness conditions.

This concludes Chapter 4 of this work, where we demonstrated how to evaluate prototype input devices in a technically immature form and still draw conclusions about their future performance after expressiveness is improved. In the next chapter, we will discuss iterative design in practice by showing how the theoretical concepts presented throughout this work can be applied to a real project.

Chapter 5

Iterative Design in Practice: Player-Centered Iterative Design for Pervasive Games

"Any sufficiently advanced technology is indistinguishable from magic."

—Arthur C. Clarke

This chapter introduces REXplorer as a real example of a ubiquitous computing application that uses mobile phones as an input device for the surrounding environment. This application serves as a test bed to bring the theories described in the earlier chapters to practice. REXplorer has applied aplayer-centered iterative design throughout the design process.

Bringing theory to practice

The fields of Human–Computer Interaction and Game Design have long recognized that user interfaces should be designed iteratively (Nielsen, 1993; Buxton and Sniderman, 1980; Gould and Lewis, 1985), because the requirements for an interactive system cannot be completely specified at the beginning of the lifecycle (Dix *et al.*, 2004). Instead, the design needs to be prototyped and tested with real users to reveal any false assumptions or unforeseen problems with the existing design. These problems can then be corrected in the next iteration of the prototype, which should then again be tested to ensure the problems are resolved.

Interfaces should be designed iteratively

Currently, we are lacking the tools and conceptual frameworks to fully support iterative player-centered design in the domain of pervasive games, because existing methodologies for the desktop computing, such as paper prototypes, do not scale to ubiquitous computing applications (Liu and Khooshabeh, 2003). A desktop environment is targeted for one user, one set of hardware, and a single point of focus. In pervasive games, complexity is added in every direction; there are multiple players and player backgrounds, dynamic contexts of use, diverse spatial qualities, different

Ubicomp applications require new design tools and techniques

metrics for successful interfaces, varying stakeholder interests, as well as political and economical interests that may change over time. This work starts to fill the gap by showing how to apply the concepts presented in this thesis to pervasive game design.

Earlier, in Section 2.3, we have seen how the design space of mobile input techniques can be used to analyze mobile input alternatives for a new application. This chapter walks through the rest of the player-centered iterative design process for this game. Each design iteration was important in reaching the current design which balances conflicting forces and satisfies both stakeholders and players in our user tests.

5.1 Game Overview

Game designed to make learning history fun

REXplorer is a pervasive game that helps tourists explore the history of Regensburg, Germany. In the game, historically-based spirits are stationed at points of interest throughout the city. Players use a special "paranormal activity detector" (a device composed of a mobile phone and a GPS receiver in a protective shell) to interact with location-based and site specific spirits. A novel mobile interaction mechanism of "casting a spell" (making a gesture by waving the wand-like detector through the air) allows players to awaken and communicate with spirits to receive and resolve quests. The game is designed to make learning history fun for tourists and influence their path through the city.

Extending the museum experience

REXplorer is a part of the Regensburg Experience[1] (REX) museum in Regensburg, Germany. The museum itself contains interactive exhibits to allow visitors to experience different aspects of the city's cultural heritage, such as medieval music, and poetry. REXplorer is designed to extend the visitor experience beyond the museum walls, showcasing the most significant attraction of Regensburg: its mostly gothic and romanesque urban silhouette and architecture. Regensburg is a UNESCO world heritage site and one of the best-preserved medieval cities in Germany, mostly untouched by the widespread bombings during WWII. REXplorer engages players to narratively and, by physical mobility, link city sites, creating an interconnected mental map, and changing the visitors' perception of the destination.

Paranormal activity linked to a secret language

The target group of REXplorer mainly consists of younger visitors with German language proficiency. The theme of the game is techno-magical: Visitors are asked, as scientific assistants, to examine paranormal activity recently discovered outdoors in the Regensburg medieval city core within one hour. Fictional "scientists" have discovered that the phenomena seem to be linked to a child's gravestone inscribed with a mysterious secret language shown in Figure 5.1. The gravestone is a real artifact in the Regensburg cathedral, and historians (factual) have found that the symbols used instead

[1] http://www.rex-regensburg.de

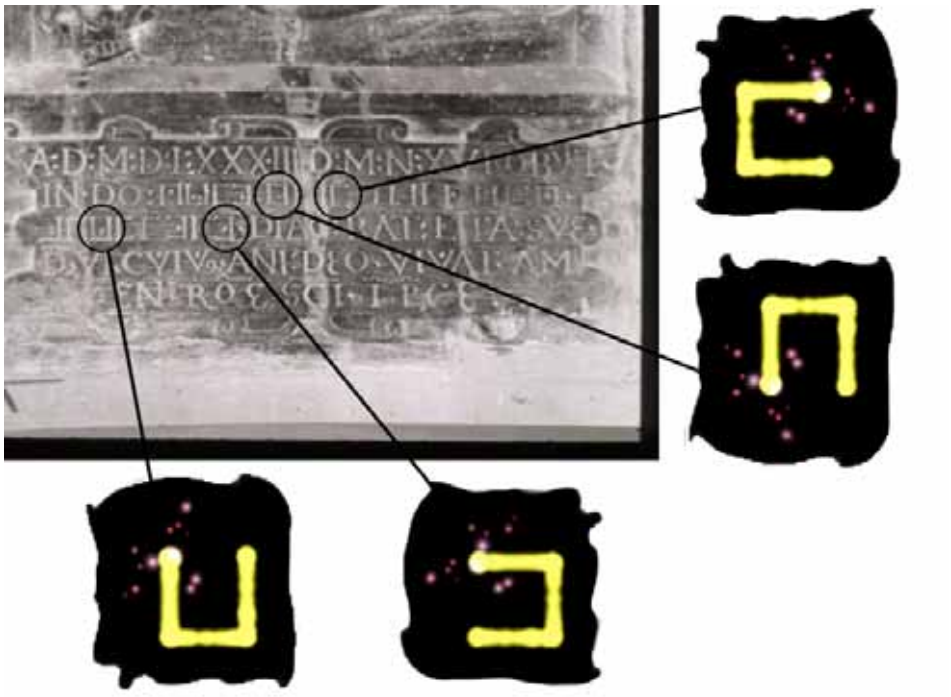


Figure 5.1: A child's gravestone inscribed with a secret language serves as inspiration for the gesture vocabulary of REXplorer. The long-term goal of the players is to unveil the mystery behind these symbols by solving as many other challenges in the city as possible during their game session.

of letters cover up the identity of the child buried, who is thought to be an illegal offspring of a Regensburg cleric - a scandal in the 16th century!

For field research, the scientists have developed a special detector device (see Figure 5.2) that is able to measure paranormal activity at specific sites in the city core. The detector has artificial intelligence and is able to talk directly to the players. This makes the device a character in the game, encouraging players to anthropomorphically relate to it as a team member helping them achieve their goals. The detector notifies players when they are in the vicinity of paranormal activity (and a point of interest) through it's own excited heartbeat, further emphasizing its human qualities.

Detector indicates presence of spirit

When near a historically significant site, players can evoke one of the gravestone symbols by drawing the symbol through the air, similar to casting a spell with a magic wand. Each symbol draws power from one of four medieval elements (earth, water, fire, or wind) and establishes a communication channel to the spiritual world, allowing the either historical or mythological spirits to tell their "cliff-hanger" stories through the loudspeaker of the device. Each story challenges the players to fulfill a quest by going to a different point of interest in the city. Players need to listen carefully to the spirits to capture the verbal clues indicating which gesture to use to accept a quest. When the quest is completed at another site by interacting with another spirit, the "cliff-hanger" narrative is resolved, and

Spirits share history and issue quests

Figure 5.2: The REXplorer "detector" consists of a Nokia N70 mobile phone and a GPS receiver packaged together in a protective shell. A soft and stretchable textile overlay with a zipper on the back transforms the standard phone keypad into an 8-key game interface.

a new quest is offered. For each completed quest, players receive points, allowing them to level up from rookie to master research assistant during their game session.

Blog summarizes tourist experience

During the game, the player's progress is tracked and used to create a personalized souvenir geo-weblog (blog). The player blog documents their route over space by interfacing with Google maps and their route over time by chronologically listing all sites and characters the player has visited during her session (see Figure 5.3). The blog provides de-briefing web links concerning the characters appearing during gameplay to help the players learn more about the history behind the sites. During their game session, players can shoot pictures of their field research. This image material is also automatically added to the blog with corresponding locations marked on the interactive map.

5.2 Detector Functionality

The detector's simplified keypad, refined through several iterative design stages, interface provides users access to the following functionality during the game:

Check game status

- **Status button:** Players can check their status at any point in the game. The status menu shows them their current point score, their current level, and their current open quests including a short description to remind the players of the nature of the quests.

Clutch to activate gesture input

- **Gesture button:** REXplorer is the first pervasive and mobile game to enable magic wand style spell-casting. Players hold down this button while performing a gesture. Releasing the button indicates the end of the gesture. Gesture recognition is accomplished using camera-based motion estimation, as in (Ballagas *et al.*, 2005; Wang *et al.*, 2006). As motion samples are collected, they are rendered to the screen to allow players to see their gesture progress. After the gesture is complete, the motion trail is normalized and the data is

Figure 5.3: A souvenir blog documents the player's route, visited points of interest, and player-generated content (pictures and videos). Clicking on a point of interest in the map takes the visitor to in-depth historical information with external links and a bibliography to explore and learn more.

passed to a gesture recognition algorithm. A legend of gestures is provided in a souvenir brochure players receive at the start of the game (see Fig. 5.4).

- **Repeat button:** Players can always repeat the verbal communication from the spirits at any point during game play if they missed important details because of real-world interruptions such as traffic noises or the many other roaming tourists during the peak season.

  Replay audio

- **Volume buttons:** Players can adjust the volume of the device at any time to adapt to changing environmental conditions and social scenarios.

  Adjust volume

- **Map button:** Since the players are tourists, they generally have difficulties navigating through a foreign city. To compensate this, we provide a physical tourist map in the souvenir brochure, indicating the paranormal activity sites (see Figure 5.4). As an additional aid, the players can see their current position on a smaller on-screen map using this button. The on-screen map also shows the destinations for

  Show current position

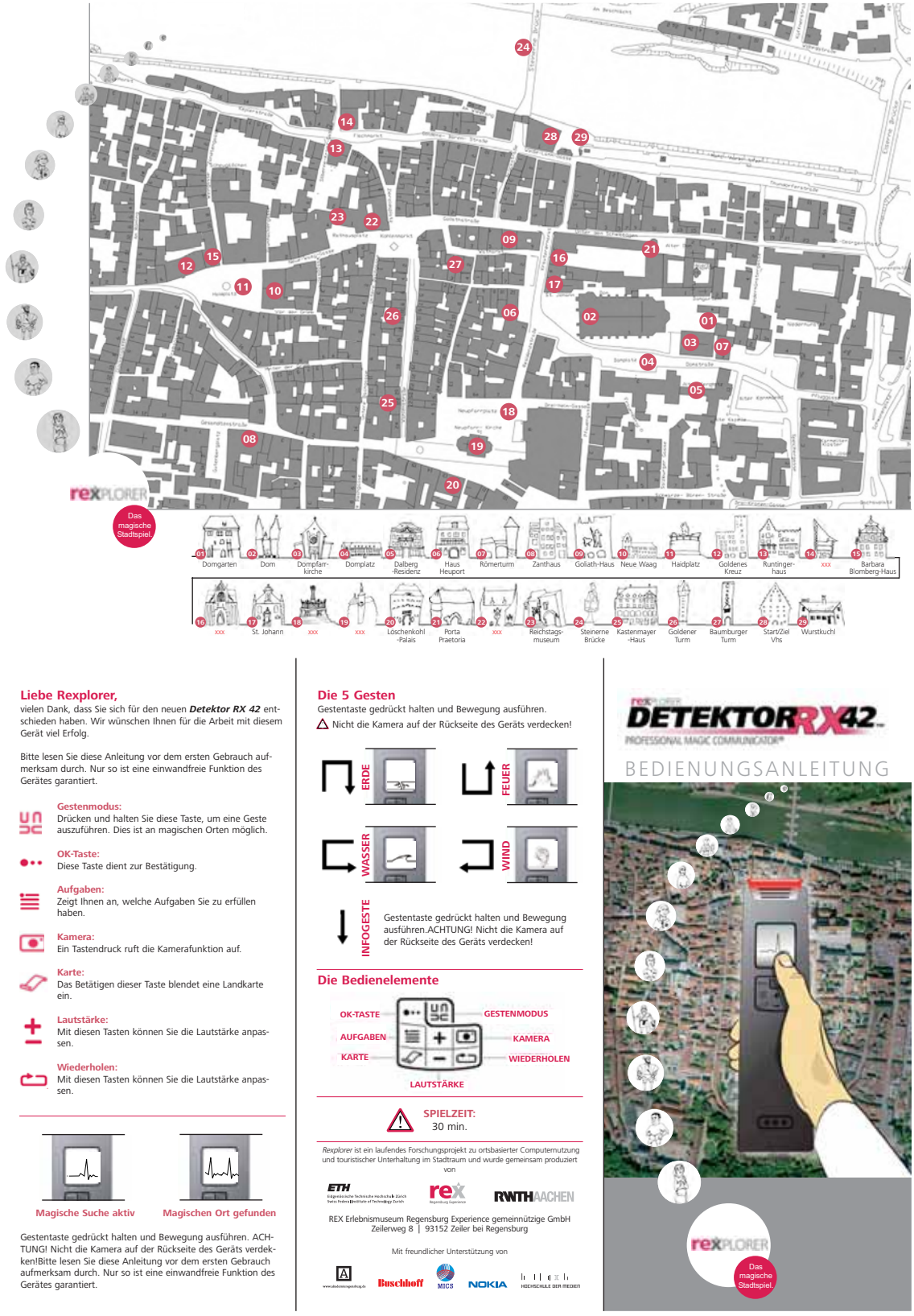


Figure 5.4: (Top) The front of the souvenir brochure has a large map with points of interest marked. (Bottom) The back of the brochure displays a legend for device buttons and gestures. Players receive the brochure when they rent the detector to start playing the game.

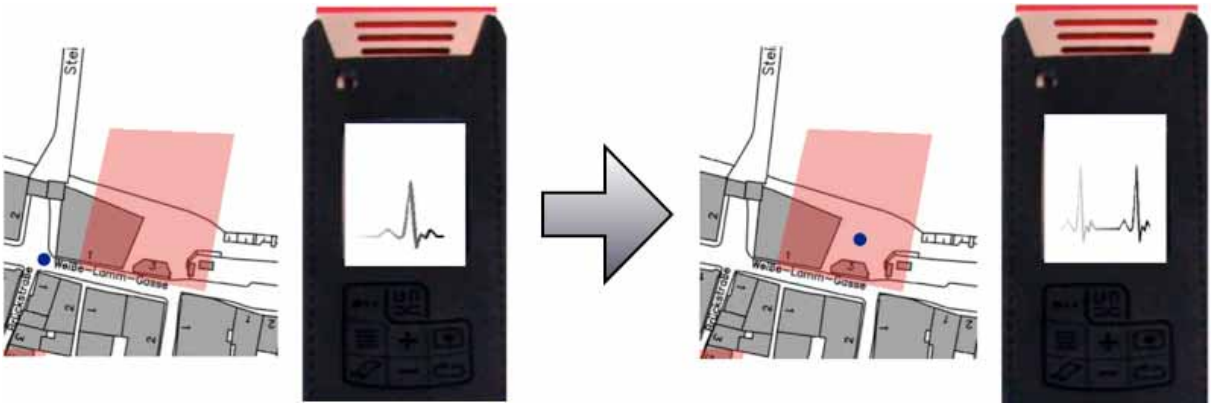


Figure 5.5: As players move through the city, a slow heartbeat indicates that there is no unusual paranormal activity. When a player moves close to a point of interest, inside a hotzone, the detector's heartbeat gets excited and speeds up. In the excited state, there is additional vibration and audio feedback to emphasize the new state.

the current open quests to help the player navigate through the city to fulfill quests.

- **Camera button:** Players are encouraged to capture their own memories with photos and videos using the built in camera. The player-generated content is automatically uploaded to a personal weblog with the location marked on an interactive map.

Take photos

5.3 Gameplay Scenario

Anna and Peter are a young couple visiting Regensburg on a day trip. At the tourist information office, they notice REXplorer advertised as a city-experience game, and decide to try it out. They are renting the detector (see Fig. 5.2) and a souvenir brochure (see Fig. 5.4) directly at the tourist information center. Then they are shown a short three minutes movie introducing them to the gravestone, the paranormal activity, and their task as a scientific assistant to help solve the mysteries of the city.

Game starts at tourist info office

As they leave to start playing, Anna is holding the detector and Peter is in charge of the brochure. They turn the corner and Anna notices a heartbeat vibration indicating the detector is excited (see Fig. 5.5) and that the couple has reached a point of interest. From the introductory movie, Anna knows that there is a spirit here that she can awaken by casting a spell.

Game played in small groups

She looks at Peter, who flips over the brochure map, looks at the different gestures, and points to "wind" for Anna to try. After glancing at the legend to get an idea for the gesture shape, Anna holds down the gesture button and waves the device through the air, just as she saw in the introductory video. As she moves the device, she sees her gesture progress on the detector screen (see Fig. 5.6) and hears the gesture mode audio sample. Once the gesture is complete, she releases the button, and a short "tornado" video

Brochure provides a gesture legend

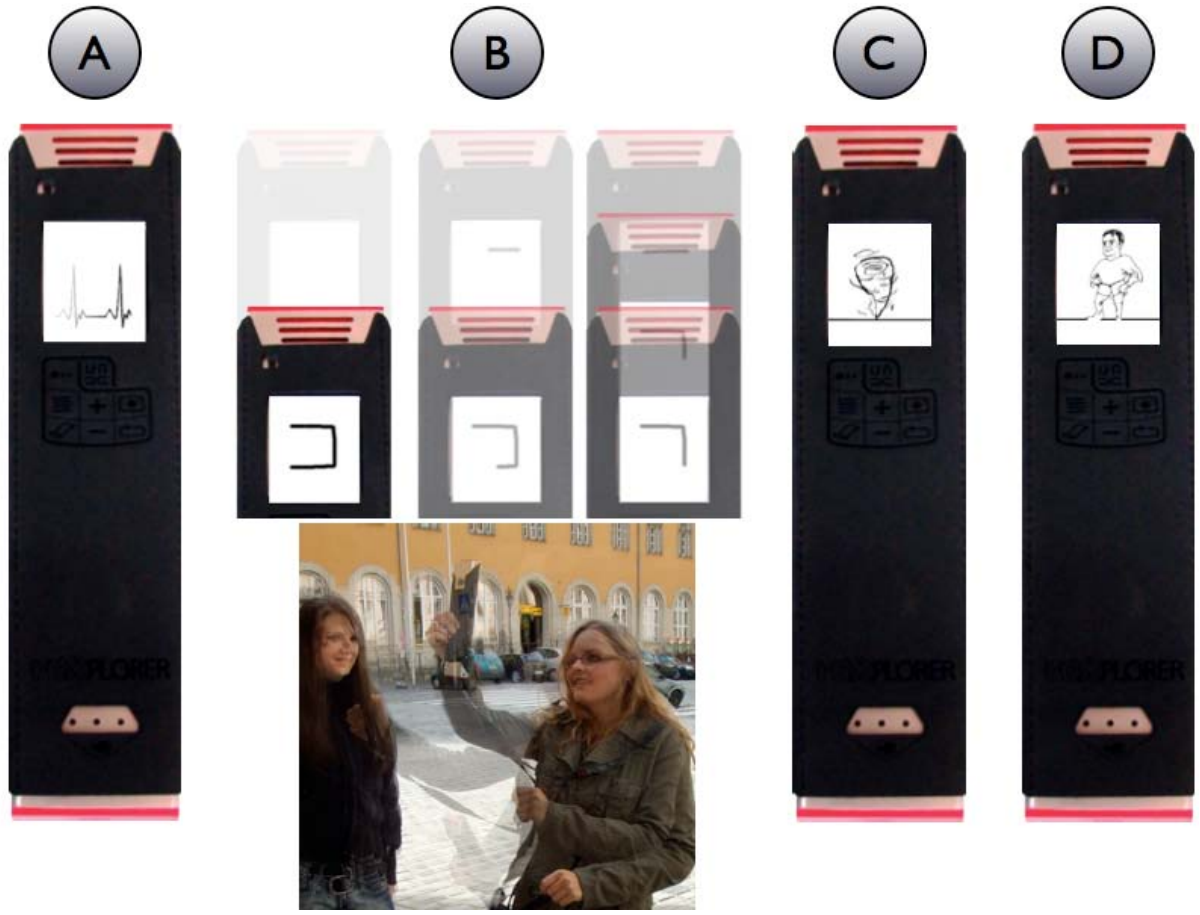


Figure 5.6: (A) The excited heartbeat indicates that the player is in a hotzone and can interact with a spirit. (B) The player casts a spell by drawing one of the gravestone symbols in the air. Audio feedback, as well as visual feedback is given to the player displaying the gesture progress. (C) An element animation (in this case 'wind') confirms which gesture was recognized, whilst the attached audio plays back. (D) The likeness of the character is displayed on the screen as the spirit communicates with the player.

with audio playback confirms that she has successfully completed the wind gesture.

A figure is shown on the detector screen and a spirit begins to speak to the players:

Spirit conveys historical info and issues a quest

REXplorer*! It's nice to see you. I am a salt trader. People like me used horses to pull heavy ships, full of expensive salt, up the river Danube to Regensburg until around 1820 A.D. Usually, the excursions last 4 weeks at a time. Yep, my life is tough and dangerous. Thieves plague the salt trading routes, but I have a loving wife who constantly prays in a nearby church for my safe return. Only the fire of her love keeps me alive. Would you be willing to deliver a message to my wife? Then show me the appropriate gesture.*

After listening carefully to the text, Anna understands that she must cast the "fire" spell to accept the quest. She looks at Peter and asks: "Which one was fire, again?". Peter shows her the gesture legend, and Anna successfully completes the fire gesture to accept the quest. Then she hears:

Quest details are given after acceptance

> *I thank you from the bottom of my heart! It pleases me that you are willing to deliver my love letter to my wife at the St. Ulrich Church near the Cathedral. Oh! My colleagues are already waiting for me at the river. Good luck! Take care of yourselves.*

Device map complements the brochure map

Peter looks on the brochure map and quickly finds the next location. He looks to Anna and asks: "where are we now?". She presses the map button on the detector which shows them their current position and the destination of their open quest. After orienting themselves, they start walking towards the St. Ulrich Church to complete their first mission.

5.4 Other Pervasive and Mobile Games

In recent years, the field of pervasive and mobile experience design has been growing rapidly, bringing forth exciting works. Although heterogenous in scope and intent, a number of recent projects can be related and compared to REXplorer in order to contour the scope of our project.

Previous subscription-based games

Most pervasive games to date are event-based and of an experimental nature. The most notable exceptions have been (a) commercial and (b) attempts to establish continuing, subscription based servicing. Mogi (Licoppe and Inada, 2006), for instance, is a cell phone and web based persistent item collection and trading game where the actual geo-position of a subscriber correlates to the position in the game world. Created by the French Telecom supported Newtgames and commercialized in Japan by the operator KDDI in 2003, Mogi was discontinued in 2006. Another example of a mobile phone subscription service is the GSM-cell positioning SMS-shooter game Botfighters (Sotamaa, 2002), created by Swedish company *It's Alive* and launched for the first time in 2001 by Swedish operator Telia. The goal of REXplorer is to achieve a research prototype of a robust, sustainable service, but as opposed to Mogi or Botfighters, the REX museum and the Regensburg tourist information will operate REXplorer as a local, site-specific offer, using rental smartphones embedded into custom made shells as game controlling devices.

Serious pervasive games

A number of pervasive games have been designed for non-entertainment purposes such as city marketing, learning, or emergency simulation. Amongst the earliest examples of a serious pervasive game is the multi-player indoor experience M.A.D. Countdown (Walz, 2005), where a "rescue" team of players – one of whom roleplays a saboteur – has to locate an

atomic bomb within eight hours and disarm it; players use wirelessly networked PocketPCs, browse puzzle websites, call fake answering machines, and interact with physical game elements such as lockers containing game clues. In an educational game, "Savannah" (Benford *et al.*, 2005), children role play lions, practicing hunting, and thereby learning about prey behavior in wildlife habitats. Environmental Detectives (Klopfer and Squire, 2005) embeds high schoolers into an authentic situation where teams of players representing different interests have to locate the source of pollution by drilling "wells" and "sampling" with PDAs. The role playing game Frequency 1550[2] blends Internet and mobile phone gameplay with location-based puzzles to supplement the city history curriculum at the Montessori school in Amsterdam.

Site-specific knowledge

Specifically Frequency 1550 is of particular interest in our context, as it demonstrates how to convey site-specific knowledge with the help of game mechanics. Both de Souza e Silva and Delacruz (2006) and Thomas (2006) describe a number of other relevant projects, examining potential uses of pervasive gaming for educational purposes. These theoretical approaches are interesting for the REXplorer gameplay, which aims at conveying knowledge about touristic sites.

Linear vs. Non-linear storytelling

Similar to REXplorer, site-specific narratives and spatial storytelling - that is, connecting site A with site B through a story - are eminent features in History Unwired (Epstein and Vergani, 2006), which was tested during the 2005 Biennale of Contemporary Art in the most touristed city worldwide, Venice. History Unwired is not a game, but an innovative and entertaining linear walking tour around one of Venice's less-traveled neighborhoods, involving location-aware smartphones and interactive art pieces at sites which are embedded into the tour. Contrary to REXplorer, the designers of History Unwired decided for linear storytelling, where users had few opportunity to "choose their own adventure", which is an important feature in the non-linear gameplay of REXplorer.

5.5 Mobile Phone Turned Magic Wand

One of the most significant ways we have differentiated ourselves from previous work is through our inclusion of a novel ubiquitous mobile interaction technique of casting a spell.

5.5.1 Camera-based Motion Estimation

Block matching algorithm

The motion information used for the gesture input is acquired using camera-based motion estimation as in the Sweep technique(Ballagas *et al.*, 2005). We employ a block-matching algorithm similar to the ones described in (Rohs, 2005a; Wang *et al.*, 2006). A correlation between the current and

[2] http://freq1550.waag.org

previous video frames is performed. Both frames are subsampled to roughly $\frac{1}{8}$th of their former size. We test the MSE (Mean Square Error) between the old and a shifted version of the new frame, with a shift range of 5×5 pixels. The candidate with the lowest MSE then delivers the motion vector. The subsampling and limited shift range can result in motion tracking errors, but the simplifications are necessary to make this algorithm work in real time on mobile phone computational resources. Wang *et al.* (2006) have previously used a very similar motion estimation algorithm for gestural interfaces and reported difficulties with the low frame rate and relatively noisy motion data.

5.5.2 Gesture Recognition

A lot of work has been done in the field of handwriting recognition systems (Plamondon and Srihari, 2000) that could be adapted to our needs. However, most of these algorithms are very complex, employing neural network classifiers (Oh and Suen, 2002) or using stochastical methods such as Hidden Markov Models (Hu *et al.*, 1996) for classification. These traditional gesture recognition engines typically use a library of predefined gesture traces, which are often entered by the user in a learning phase. As the typical REXplorer user will only use the device once, a learning phase for the gesture classifier is undesirable. While these algorithms feature a very good recognition rate operating on large and complex gesture vocabularies, they are too complex and resource-intensive for our target platform and application area.

Traditional gesture recognition approaches are too resource intensive

The storyline required us to use gesture symbols from the historical gravestone depicted in Figure 5.1. We carefully selected a few relatively simple symbols whose motion vectors were as orthogonal as possible to simplify the gesture recognition process.

Simple gesture set

We have designed a new gesture recognition algorithm that is tailored for our gesture set and is suitable for the constrained computational resources and mediocre motion estimation data. This algorithm uses state machines, modeled from a gesture rule set, that parse the motion data and interpret the gesture the user has performed. The algorithm incrementally matches the input to a gesture by verifying the entered motion data reaches certain predefined distance offsets.

Simple FSM gesture recognition

To formalize the description of the algorithm, let $M = m_1, \ldots, m_n$ be a sequence of motion tuples with $m_i \in \mathbb{N}^2$, obtained by user input. When adding up the motion tuples we obtain a "gesture trace" $T = t_1, \ldots, t_n$ with

$$t_j = \sum_{i=0}^{j} m_i \quad m_i \in M$$

As shown in Figure 5.6b, a gesture trace can be plotted in 2D space to obtain a graphical representation of the gesture.

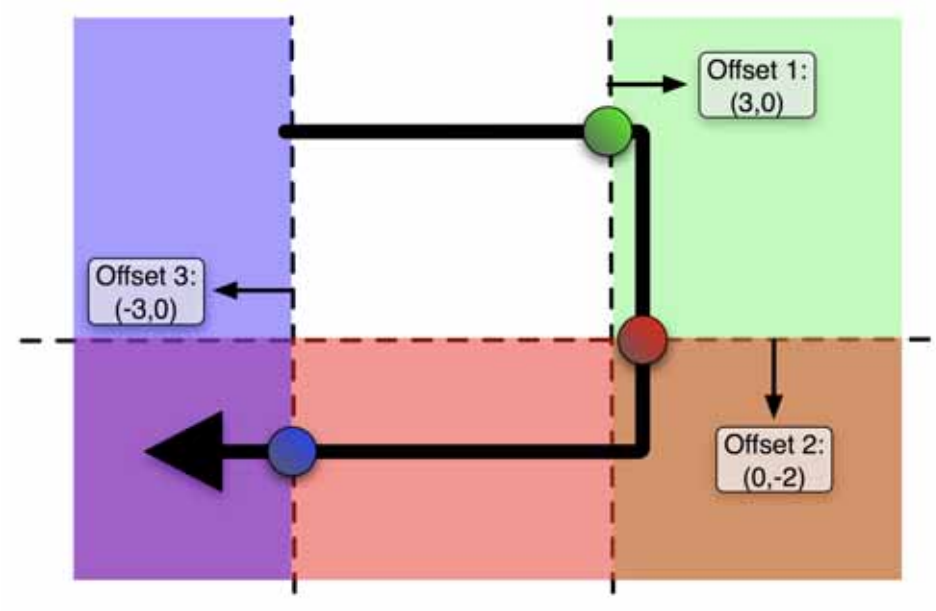


Figure 5.7: Recognizing the gesture using offsets

Normalize gesture trace

Due to the fact that users perform gestures that cover a varying area of pixels and that we only specify fixed offsets in our gesture configuration files, the input gesture trace is normalized to make its points lie within an area of 100×100 pixels before gesture recognition.

Our algorithm determines which predefined rule set is matched best by the entered motion data. A rule set defines a sequence of distance offsets that must be fulfilled by the trace of the entered motion data.

Rules specify distance thresholds

We model each gesture as the acceptance state of a state machine $\mathfrak{G}$. Upon initialization, a gesture configuration file with the sequence of predefined offset rules is loaded. Each state q_i of a state machine $\mathfrak{G}_n$ represents an offset rule. An offset specifies the (Manhattan-)distance (x, y) in pixels between the states q_i and q_{i-1}, in other words the first occurrence of the value $\delta = t_j - t_i$ with $j > i$ and t_i, t_j being two points in the gesture trace. A gesture is accepted if $\mathfrak{G}_n$ reaches it's acceptance state q_{accept}.

For instance, if, as in Figure 5.7, q_2 is defined as being $(-2, 0)$ pixels away from q_1, which was matched at the coordinates $t_i = (2, 6)$, then the automaton will change it's state to q_2 when incoming motion data can be traced to a location $t_j = (2 - k, n)$, where $k > 3$ and $j > i$.

The gesture recognizer A is thus modeled as the following union automaton:

$$\mathfrak{A} = \mathfrak{G}_1 \cup \mathfrak{G}_2 \cup \ldots \cup \mathfrak{G}_k$$

So it is evident that

$$F = \{q^{(1)}_{\text{accept}}, \ldots, q^{(k)}_{\text{accept}}\}$$

is the acceptance set of $\mathfrak{A}$.

Because it is theoretically possible that multiple gestures can be recognized from a single user input. We pragmatically modify the union criterion so that the first automaton that accepts, be it $\mathfrak{G}_n$, will yield the recognition of gesture n, while the results of the remaining automata that have accepted will be discarded. This modification is justified because we use a carefully chosen gesture vocabulary (see Figure 5.1) composed of a few very distinct gestures that not only lowers the user's learning curve but also eliminates almost all falsely recognized gestures.

Very forgiving of imperfect motion input

Our algorithm has proven itself to robustly handle the mediocre motion data. This is because, conceptually, our system is very forgiving and usually leads to a correctly recognized gesture, even if the graphical representation of the gesture trace is very dissimilar to the predefined shapes. An explanation for this is that we are, abstractly seen, testing if our internal cursor has entered the infinite plain[3] specified by the next automaton state's offset vector. This cancels out noise from inadequate motion data quite effectively as it is possible to move away from or orthogonally from the specified direction before moving correctly and finally reaching the next state.

simplifications won't work with larger gesture sets

Due to our algorithm's coarse matching criteria, it does not scale well to large and complex gesture vocabularies. Here a finer analysis of the data is required. For instance, angle sums, point distances or mean absolute distances can be used for more detailed classification. But taking all into account, our algorithm's low complexity but relatively high recognition rate allows for an efficient implementation that performs well with a simple gesture set on our target platform.

5.5.3 Iteratively Designing the Spell-Casting Experience

Design space helped analyze input alternatives

After the spell-casting game concept was settled, an iterative human-centered design process was used to ensure that the spell-casting input is an intuitive, enjoyable, and appropriate interface for REXplorer. As presented early in Section 2.3, we analyzed the design space of mobile input techniques to understand the design alternatives. Our initial decision was to use camera-based motion estimation because the physical motion required was the most similar to the spell-casting metaphor. Also, the physical style of gesturing with the arm was more likely to create an engaging experience (Hummels, 2000).

iStuff Mobile used to create initial prototypes

We used the iStuff Mobile toolkit (presented in Section 3.3.2) to create a first low-fidelity functional prototype of the gesture recognition system using camera-based motion estimation through the Quartz Composer visual programming interface. This prototype was instrumental in convincing the various stakeholders (including sponsors) of the feasibility of the gesture recognition input technique. It also allowed us to explore and narrow in on our current gesture recognition algorithm.

[3]These plains are represented by the shaded areas in Figure 5.7

Early prototypes helpful for demonstration and controlled settings

The advantage of the iStuff Mobile prototype was that it allowed us to quickly experiment with different gesture recognition and motion visualization strategies in a controlled setting. The disadvantage of the iStuff Mobile gesture recognition prototype is that it was inconvenient to do experiments "in the wild", because the compound architecture required proximity to a laptop. In order to ensure that players would find gesture recognition socially acceptable in context, and to verify that camera-based motion estimation would work with public traffic, we needed an ecologically valid evaluation. We decided to build a more advanced, integrated, and self-contained prototype to support these ecologically valid evaluations more effectively.

User Reactions to Gesture Recognition in a Field Study

Finding: Noisy gesture visualization disrupts user experience

After completing the integrated prototype, we performed play sessions in the wild, followed by focus group interviews to collect impressions of users. The gesture recognition system in our integrated prototype proved to be very tolerant and worked fairly well in public. Players were surprised of the gesture tolerance. Aaron: "What I thought worked really well was, even when I made a round C gesture, the device still would recognize it – in any case, it has a really high tolerance."[4] However, the extra noise in the motion data due to public traffic disturbed many of the users to the point where many abandoned the gesture in the middle, preventing the gesture recognition system from functioning properly. Ironically, if the player had completed the gesture, the recognition system would have in most cases tolerated the extra noise in the motion data and successfully recognized the gesture. Lacy described her experience as follows: "In terms of the gestures (motion)... the device didn't recognize everything entirely."[5] This led us to realize that the smoothness of the gesture trace visualization was very important to the spell-casting experience, because players had preconceptions about the robustness of the recognition system.

Limiting motion vectors to match gesture set

To improve the smoothness of the motion data, we loosely applied the concept of expressiveness from Chapter 4. Specifically, there is a mismatch between the motion vectors required for our gesture set, and the motion vectors that can be expressed by our input device. The gesture vocabulary only requires Manhattan-style motion along the X and Y axes, but our motion estimation algorithm can generate motion vectors along our 5×5 motion search grid. To translate the 2D motion vector to Manhattan-style motion, we simply take the larger of the X or Y motion parameters. In the case they are equal, the motion history is used as a tie breaker.

Filtering noise

To further improve smoothness in the presence of noise, we employed a momentum heuristic where the direction of motion would not be changed

[4] Original German Text: "Was ich ganz gut finde, ist dass selbst wenn man ein rundes C macht, das Geraet selbst das noch erkennen wuerde - recht grosse Toleranz auf jeden Fall."

[5] Original German Text: ""Das mit den Gesten... das Gerät hat das alles nicht so ganz erkannt."

until 3 consecutive samples were received moving in the same direction (after Manhattan conversion). Any motion samples received that are not in the current direction of motion are ignored in the gesture trace until the direction has changed.

Finding: Troublespots

Several testers had problems with repeated recognition failures at a particular location (e.g., when they were facing a street with moving cars). Danny said: "I made one of these (gestures) five times, and then I thought, if it fails one more time, then I'll lose interest and leave it behind."[6] This points to the need for an alternative or backup method for selecting a spell in these difficult locations.

Social awkwardness vs. Physical engagement

Some participants (especially our older participants who played in solitaire) found the gestures socially awkward, and they requested dedicated spell buttons instead of gestures. However, others mentioned that the gestures were an important part of the experience adding to the sense of magic and mysteriousness. They also argued that the game shouldn't be without challenges. In a focus group interview, Maria said: "We had fun with the fact that it was hard to trace out the gestures. When it works everytime, then it's boring. It shouldn't be too easy."[7] Tom followed up with: "It was somehow funny"[8] Emotional reactions were also common when people successfully performed a gesture. Irene said: "Bravo... Yeah!"[9] after performing a correct gesture during a play session.

Alternative spell input required

We designed an alternative spell selection mechanism to help balance the forces of physical engagement and social awkwardness. This alternative selection works through an on-screen menu (see Fig. 5.8) that can be used anytime an invalid gesture is performed, effectively allowing people to avoid gestures altogether by intentionally entering invalid gesure input. Players who choose to avoid gestures are penalized by not earning points.

5.6 Other Prototyping Iterations

Iterative design was not only used to refine the spell-casting experience, but also for many other aspects of the game.

5.6.1 Early Concept Prototyping

In the early stages of the design, we generated a number of game ideas and formalized these ideas into one page conceptual design treatments, briefly

[6] Original German text: "Ich habe so eine (Geste) fünf Mal gemacht und dann habe ich gedacht, wenn Du das jetzt nochmal machst und es nicht klappt, dann hast Du keinen Bock mehr, dann lässt Du es bleiben."

[7] Original German text: "Wir hatten Spaß daran, dass es schwierig war es hinzumalen. Wenn es auf Anhieb klappt, dann ist es ja langweilig. Es darf nicht zu einfach sein."

[8] "Es war witzig irgendwie."

[9] Original German text: "Na bravo... Yeah!"

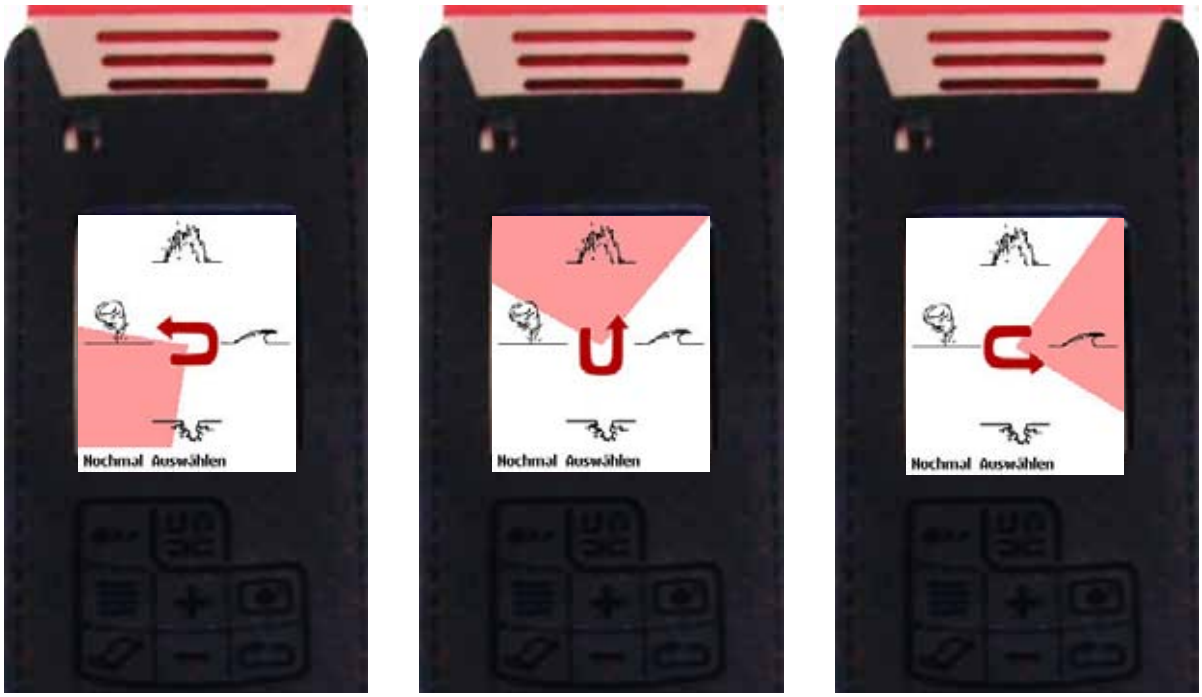


Figure 5.8: The alternative spell selection menu was a rotating, clock-like interface where the red highlight continuously rotated around the screen. The gesture symbol in the middle discretely rotated to eliminate ambiguity as to which element was currently highlighted. Note that the shape of the gesture in the middle matches the gesture that would be performed to evoke the corresponding spell for the medieval element. Users press "Auswählen" to select, and "Nochmal" to try the gesture again.

Early conceptual design treatments

describing formal and dramaturgical elements of the games (Fullerton *et al.*, 2004). We presented these treatments to our main stakeholder, the REX museum. After REX stakeholders had opted for the basic idea that eventually became our game, we created scenarios and storyboards outlining potential core mechanics of the game (see Figure 5.9), as well as a first physical prototype in the form of a board game (see Fig. 5.10), next to a simple map application indicating Regensburg sights and the walkability of the city core.

Presented storyboards to users

We used this demonstration and simulation material to collect feedback from the target group. We presented the game to two German high school classes – 10th and 11th grade – living in a city three hours away from Regensburg, followed by a questionnaire and a focus group discussion. Generally, the feedback showed that the magic theme was very important in attracting the interest of this age group and that the idea of a history game alone wasn't as attractive. However, the feedback helped us to move away from the original purely magic theme to a techno-magic theme involving science elements. The target players expressed high interest in playing the game themselves, and they found the technical aspects novel and interesting. In general, the board game prototype and the storyboards were a success as mechanisms to flush out the different gameplay elements and get high-level feedback on the early game concept from our target players.

Figure 5.9: Storyboard conveying REXplorer game play

Figure 5.10: Board game prototype of REXplorer

Early prototypes helped communicate ideas

Following our target group research, we created a design document draft with functional specifications and an iteration of the board game prototype, as well as a video walkthrough of the gestural interaction. These conceptual techniques served a very important role in communicating our ideas to various stakeholders to win their support. In REXplorer, there were a wide variety of stakeholders whose concerns needed to be addressed. Our clients, the REX museum, were trying to portray an image of historical integrity. Our conceptual material was effective in assuaging fears of too much fantasy and too little historical content. We also used the mentioned material to win over support of the municipal government to provide space and resources necessary for the proper operation of the game. The conceptual material was also instrumental in convincing local tour-guides that we were not attempting to compete with them and even helped recruit them for content oversight. Eventually, these early concept prototypes helped communicate the abstract game concept in a concrete fashion to sponsors.

5.6.2 Board Game Prototyping

Board game testing is low cost

In addition to being a demonstration tool, a board game prototype provides a world-in-miniature that allows the gameplay to be tested easily. This form of prototype is very useful for early stage content testing by reading the content aloud as the players progress through the game. It helps express spatiality, get a feeling for travel times, oversee proximities of sights, achieve narrative consistency, and helps to ensure that the underlying game is fun.

Dice and event cards can be used to regulate the progress through the city streets to give a more realistic simulation of the way people actually move in the city.

5.6.3 Game Statecharts

FSMs helped formalize UI

The formal UI state machine diagrams were important for defining exactly what text needed to be written for each character. By flushing out the design formally, we were able to ensure that we had accounted for every possible game state and error condition before the script was written and recorded. The statecharts also served as a design document for the software implementation of the game engine.

5.6.4 Content Prototyping

The content directly supports the main purpose of this serious game: educating the players. Therefore, it was very important to carefully craft a high quality dialogue. To prototype the different characters in the game we used character sheet format. This is a one-page description of the different characters that provided an at-a-glance overview to simplify the review process. The sheets included the following entry fields:

- Bookkeeping elements: including name, building address, character ID.
- Physical Characteristics: including age, height, hair color, weight, vocal characteristics, species, remarkable features, and sketches of each character.
- Inspiration for spoken text: including typical quotes, description motivating why this character belongs at this location, and a brainstorming list of what the content could include.

Local tour-guides oversaw content

These character sheets were important in communicating our more detailed content ideas with the local tour-guides for content oversight. The character sheets provided a compact and highly-browsable format which supported an effective review process. The tour guides were able to suggest improvements or changes in character selection very easily using this format. The changes at this stage were easy to incorporate and they prevented significant rewriting of the full script later on.

5.6.5 Hotzone Prototyping

GPS can have problems in urban spaces due to buildings or even clouds obstructing signals from the satellites. It is very important to test the

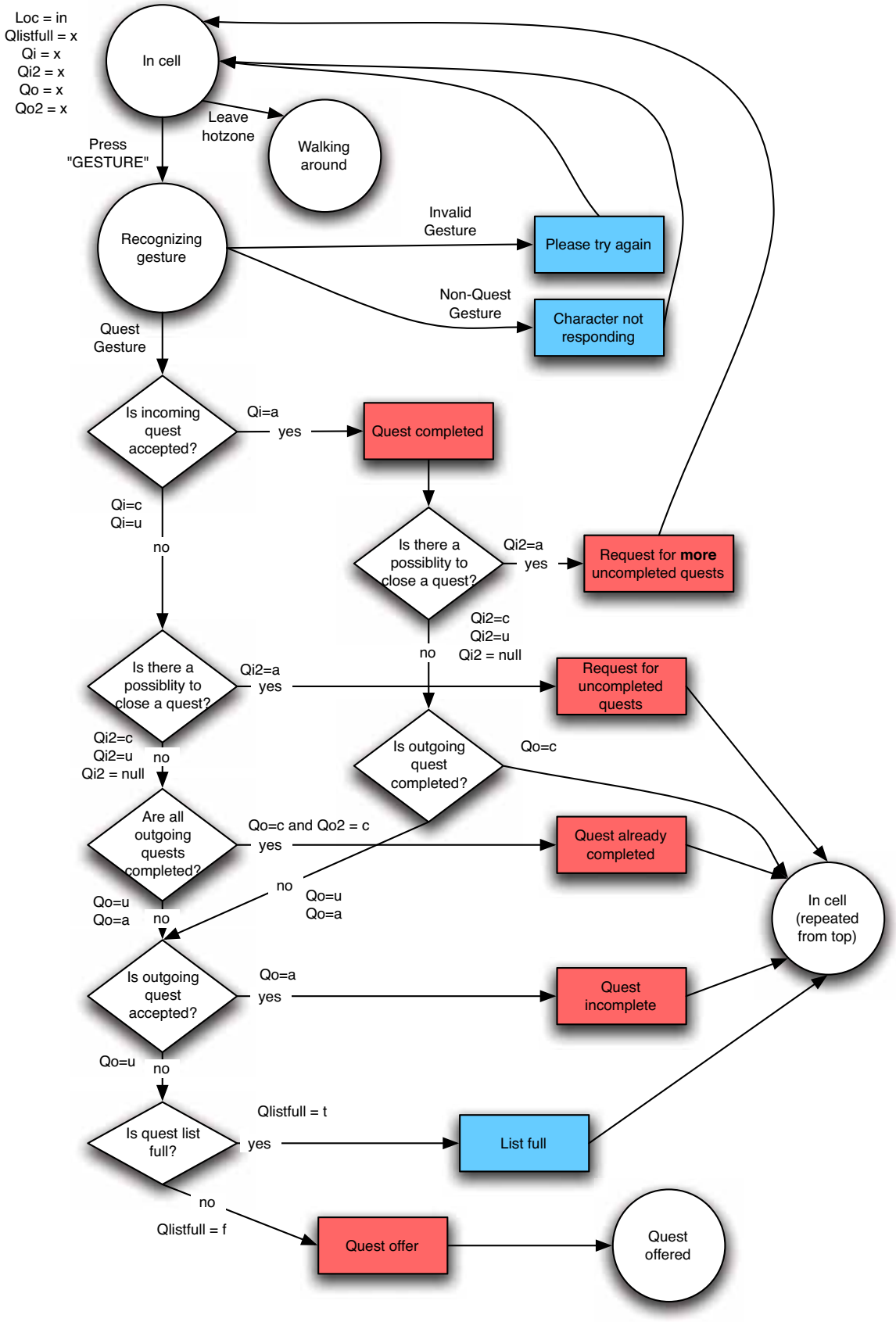


Figure 5.11: Finite State Machine showing the reaction to a gesture. The colors determine whether the spoken text is from the character or the device itself. This is only an excerpt that defines behavior for the "In cell" state, similar diagrams were created for each of the remaining game states.

Figure 5.12: Map tool that allows us to visually define hotzones based on GPS measurements from testing

location system thoroughly to ensure proper functionality. Our hotzones are defined iteratively based on GPS measurements. Leveraging our previous experience in tools for ubiquitous computing applications(discussed in Chapter 3), we developed a tool (see Fig. 5.12) that allows us to define the hotzones visually based on the GPS measurments from testing. Using this tool we were able to iteratively define hotzones, and were able to determine that GPS alone is not sufficient for the accuracy that we require. To support location detection, we also considered using Bluetooth beacons, as well as providing the ability for players to select their own location when the location detection fails.

Visually defining hotzones iteratively from play session data

5.6.6 Detector Prototyping

The form the detector (game controller) including the keypad went through many iterations before we ended with the final design (see Fig. 5.13). A small group of students co-created the detector, seeking input from industrial design professionals, and from a manufacturer specializing in leighweight metal bending and laser cutting. There are many considerations

Many stages of form prototyping

that need to be addressed in the design, for example it needs to:

1. house the phone and GPS receiver together in a tamper-proof, protective shell.
2. support the atmosphere of the game by providing a look that fits the story description of a scientific detector and a feel that mimics a techno-magic wand.
3. provide a skin for the phone keypad to provide a customized game interface.
4. allow for quick recharging of devices.

During the prototyping phases, different materials were tested. Plastic was the first choice, but it proved not to be robust enough. A thin aluminum skeleton was used in the final design, wrapped with a soft and stretchable textile with a laser-etched keypad layout.

5.6.7 "Wizard of Oz" Playability Tests

Simulate parts of functionality for experience testing

For late stage prototyping, we performed an on-site player study. Our location detection enhancements were not ready yet, so we used a "Wizard of Oz" prototype to simulate a fully functional game. In this prototype, we used a Nokia 770 internet tablet with a custom application that allowed a test administrator (a wizard) to follow the player and manually input the current player location on the touch screen as the player moved through the city streets to simulate a near-perfect location detection system. The Nokia 770 was connected via a Bluetooth connection to the mobile phone housed in the game controller. The communication protocol from the tablet to the REXplorer detector was identical in format to a standard GPS receiver, so the software implementation of the detector required no additional changes for this prototype to function properly.

Focus group interviews after play sessions

For the structure of our study, we used a product-interactive focus group (Lee *et al.*, 2004b). Traditional focus groups center on discourse regarding a early concept guided by a moderator, similar to the techniques we used with our storyboards and early board game prototype with high school students. In product-interactive focus groups, on the other hand, users are asked to perform a certain set of tasks using a product before the group discussion. In the case of REXplorer, 18 participants played a full one hour game session before joining a focus group discussion with several other players at the end of their game session.

Analysis

We transcribed the video data from the focus group interviews and the playing sessions and then analyzed the data for patterns of behavior using

Figure 5.13: The detector houses the GPS receiver and the mobile phone in a protective shell; it is wrapped by a stretchable and soft textile with custom key labels. (Left) An early prototype consisting of construction paper, wrapping a wooden core, and buttons sketched on with a metallic pen. (Middle-Left) A more advanced prototype consisting of cardboard, a real mobile phone, and a close to final textile skin. (Middle-Right) A prototype for materials exploration, this plastic design housed both the mobile phone and the GPS device, but did not properly expose the buttons. (Right) The final design with an aluminum protective shell covered in a soft textile wrapping.

an affinity analysis (Beyer and Holtzblatt, 1998) as shown in Fig. 5.15.

5.7 Design Rationale

Towards a language of design patterns

Each iteration in the design process helped formulate the current design based on a great deal of feedback from our stakeholders and players (especially the data captured in our affinity analysis). In this section, we will outline the most significant factors that resulted from the many prototype iterations. Specifically, we will discuss how we were able to balance conflicting forces that pulled the design in competing directions. This rationale should serve to inform the design of future pervasive game systems, eventually leading to a language of design patterns (Borchers, 2001; Bjork and Holopainen, 2005).

Figure 5.14: For the playability tests, Wizard of Oz techniques were applied using the Nokia 770 tablet. A test administrator uses the tablet to manually input the players' position as they follow the players through the city to simulate a fully functional location detection system.

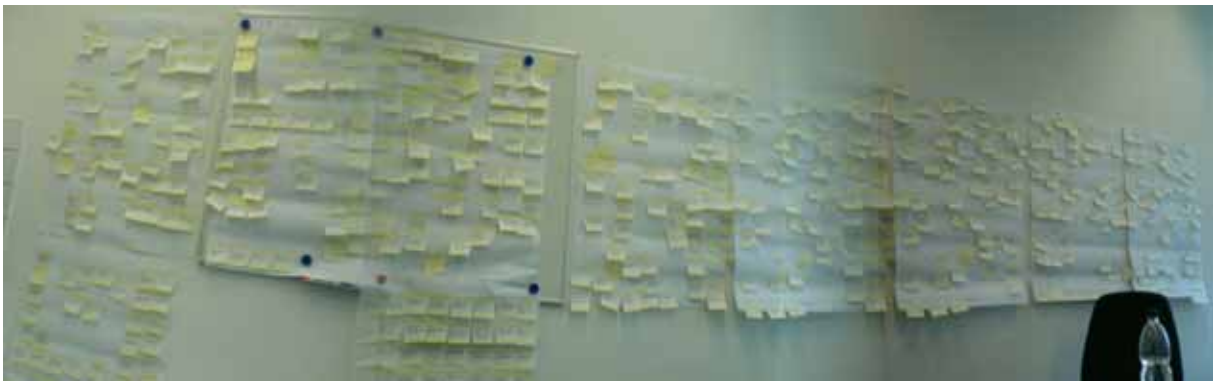


Figure 5.15: An affinity analysis consists of writing individual quotes from the interviews and play sessions to try to isolate patterns of behavior across the different sessions.

Designing for Narrative Consistency

Many important historical periods

One of the primary challenges of designing a history game for the city of Regensburg is that the city's history spans several periods starting from the Roman empire as a strategic military fortress, through the middle ages as a vital trade center, up to the early 1800's where it served as an important political center as the permanent seat of the Imperial Diet for the Holy Roman Empire of German Nation. Focusing on any of these periods would

not do justice in conveying heritage of the city, yet having a game that spans the different periods risks inconsistency, or overly structured linearity. The difficulty is finding a story that appropriately bridges these periods. In REXplorer, we use the "haunted house" and "techno-magic" themes to bridge these historical movements in a non-linear story.

Entertainment vs. Education

Early on in our discussions, the "magic" theme was negatively received by many of the stakeholders. It was perceived as too close to the "Harry Potter" series or too much oriented towards "Disney", as our client valued a strong academic and historical basis that would be seen as educational instrument and not pure entertainment. On the other hand, we wanted to create a game that was fun and exciting, and not just an uninspired audio guide regurgitating the facts from the history books. By the way of the iterative design techniques, we were able to find an appropriate balance of fun and seriousness, blending fact and fiction in a way that conveyed historical information effectively, yet preserved the engaging techno-magic background.

Folklore is an important part of culture

One key breakthrough that helped us reach this goal is recognizing that local folklore is an important part of the history and culture of the city. We leveraged local folklore to create a number of our fictional characters. We also derived the techno-magical gesture vocabulary using a real historical artifact, blending game make-belief with fact. These were key design decisions that helped us ultimately win over skeptics. The tradeoff here is that we make the game less portable to a different site, but with a history game it is important that all of the content is tailored to the site, including details such as the inspiration of the gestures.

Balancing Competitiveness and Leisure

Point score moved to background

From the playability tests, points and game levels were a polarizing topic. People who were avid games players demanded it. However, non-gamers (especially our older participants) mentioned that points made the experience feel competitive and detracted from a relaxing tourist experience. Our compromise was to keep points away from the main screen and move them to the status menu. This made points visible to those seeking them, and hidden in the background for those who weren't interested.

Balancing Cooperative Experience vs. Outdoor Play

Promoting shared memories through cooperative experience

Tourism is rarely an activity enjoyed in isolation; instead, it is an activity that is shared with family and friends to create common experiences and memories. REXplorer is designed to support shared experiences through cooperative gameplay, where two or three people can share a single controller. To support this model, the controller was designed to use the phone loudspeaker instead of headphones. The shell is designed to reflect the audio playing from the phone loudspeaker towards the players when holding

Figure 5.16: REXplorer is designed to be played in groups of two or three.

the device in front of them, making use of the shell as a resonator. The controller is surprisingly loud, and in our playability tests, most had no problems hearing the audio. In some scenarios, unusual levels of street noise made the audio difficult to hear, but players were easily able to cope by holding the loudspeaker closer to the ears between their heads. The controller also has a "repeat audio" button to handle cases when something prevents the text from being heard. Plugging in headphones is also supported by the controller by the way of a cut-out, but headphones isolate the player and should only be used in single-player scenarios.

Designing for a "Heads Up" Experience

One criticism that we received early in the design process was the concern that people would have their attention focused primarily on the device screen, and that this would detract from the real attraction of Regensburg:

the medieval architecture. There are also safety concerns when people are moving through a physical space while visually engaged with a handheld screen. This criticism focused our design to discourage the player from looking at the device screen. First of all, this led us to choose the small screen form factor of a phone instead of a larger PDA, or tablet computer. On the screen, only simple visualizations are used. The black & white static sketches were chosen over vivid color animations to enable people to get an impression of the character at-a-glance and spend the rest of the time visually absorbing their environment. During our playability test, the black & white content had another unexpected benefit of providing a high-contrast interface suitable for bright sunshine.

Keep focus of attention on city architecture

The small display makes it more difficult to navigate with the on-screen map. However, we provide the players with a map brochure intended for navigation. The device map additionally serves to help pinpoint their current position. The small on-screen map makes it undesirable to focus attention on the map while moving through the city, which helps to divert attention away from the device screen.

Small map difficulties compensated with brochure

Multi-modal feedback also allows players to keep their visual attention away from the screen during gameplay. When players enter a hotzone, the device notifies them with an accelerated heartbeat that they can hear, see and feel. The haptic and audio feedback allows them to keep the device at their side until they have reached their next destination.

Multi-modal feedback reduces need for visual attention

5.8 Chapter Summary

In this chapter, a real example of a ubiquitous computing application that uses mobile phones as an input device for the surrounding environment was presented. The chapter demonstrates how player-centered iterative design can be applied from drawing board to deployment. The concepts presented in earlier chapters contributed to enabling an iterative design process. In the early design stages, the design space of mobile input techniques (presented in Chapter 2) helped reason about design alternatives and select the mobile input techniques that were most appropriate for the application domain. For the early prototyping stages, the iStuff Mobile toolkit (presented in Chapter 3) helped with early-stage functional prototyping of spell-casting using gesture recognition. These early functional prototypes were instrumental in convincing stakeholders of the feasibility of the idea, and quickly narrowing in on an appropriate gesture recognition algorithm. The concept of expressiveness (presented in Chapter 4) also helped optimize the motion estimation input to best match the given gesture set. Other non-obvious prototyping techniques, such as board game prototypes, helped iron out some of the playability issues of the game earlier in the design process. The high fidelity prototypes helped us perform ecologically valid evaluations to isolate issues related to bringing a ubiquitous computing application into a real public setting. Clearly there are difficulties and challenges remaining in the application of human-centered iterative design

Theories from previous chapters successfully applied in a real context of use

to the ubiquitous computing domain, but this example demonstrates that the tools and conceptual frameworks presented in this thesis are helping to close the gap.

This concludes Chapter 5 of this work, in which we show how iterative design for ubiquitous computing is applied in practice. The next chapter will summarize the contributions of this work and provide an outlook for continuing the research vision.

Chapter 6

Conclusion

This dissertation demonstrates that despite the numerous challenges, iterative design can be applied to the ubiquitous computing application domain.

6.1 Contributions

In this section, the contributions of the thesis are reiterated with a summary of how each contribution was achieved.

1. *Organizes mobile phone input techniques into a design space that:*

   (a) *Organizes mobile phone input techniques into families to help reason about their relationships.*

   The structure of our design space used the subtask the input technique performed with other dimensions of interest such as feedback (continuous vs. discrete), interaction style (direct vs. indirect), dimensionality, and relative vs. absolute. Laying this structure out into a design space allows visual association of similarities and differences of interaction techniques. It also helps reason about where to look for alternative input techniques and what properties they require.

   Design space visually organizes design alternatives

   (b) *Aids designers in considering alternative parallel designs and selecting the most appropriate mobile phone input technique for a particular interaction scenario.*

   Early in the design process, the design space can be used to generate ideas for parallel designs to comparatively test in a human-centered design process. As the designs mature, the best design becomes clear, or the strengths of the top designs can be merged to a unified design.

   Helps with parallel design

(c) *Allows future input techniques to be predicted.*

Helps identify gaps

Gaps in the design space are visually apparent, and can allow a designer to analyze which future input technique might fill the gap. This property was most apparent when examining the original iStuff hardware components in a design space to visualize the gaps of coverage of that early toolkit and identify where it could be extended.

2. *New interaction techniques for employing mobile phones as input devices in ubiquitous computing application scenarios.*

   (a) *The Sweep and Point & Shoot input techniques use the camera on the mobile phone as a sensor to enable interactions with large public displays.*

   New camera-based mobile input techniques

   The Sweep technique is a continuous, relative, indirect input technique that uses camera-based motion detection to allow a user to control a cursor on a remote display using 3 degrees of freedom (X, Y, θ). An analogy can be made to an optical mouse, in that motion is tracked through a camera view. The Point & Shoot technique is a discrete, absolute, direct input technique that allows the user to use the camera as a view finder to aim at and select their desired target. It uses tags in the environment to establish an absolute coordinate system on the remote screen.

   (b) *The Elope project shows how to fluidly combine the storage and processing of mobile phones with the input and output capabilities of an interactive workspace.*

   New RFID-based interactions

   Elope is a novel way to combine personal data on your mobile phone with high-fidelity input and output facilities in an interactive workspace. The idea takes advantage of the affordances of objects in the room (such as a presentation remote control) to represent room configurations and services.

   (c) *The REXplorer pervasive and mobile game for tourists employs mobile phones as a platform where users can interact with spirits (historical characters) distributed throughout an urban environment by casting spells (gestures created from waving the mobile phone through the air).*

   First mobile spell-casting interaction

   REXplorer is the first pervasive game to use spell-casting as an interaction metaphor, allowing the mobile phone to be used as gestural input based on the Sweep technique mentioned above.

3. *Architecture support for physical user interface input and mobile phone input techniques in ubiquitous computing application scenarios that:*

   (a) *Lowers the threshold for prototyping ubiquitous computing appli-*

cations that employ physical or mobile phone input while maintaining a high ceiling of prototyping activities.

The iStuff architecture makes it easy to use a variety of hardware components ranging from mobile phones to sensor network platforms to off-the-shelf commercial components as building blocks to construct novel ubiquitous user interfaces. A broad range of examples demonstrates the high ceiling of prototyping activities, and an evaluation demonstrates the low threshold of prototyping.

Radically simplifies construction of functional prototypes

(b) *Supports incremental integration, extensibility, and rapid configuration of input using the Patch Panel infrastructure.*

The Patch Panel is a systems approach to address issues of heterogeneity and interoperability in ubiquitous computing environments. It uses event intermediation to allow otherwise incompatible entities to communicate, even if they weren't intended to interoperate.

Supports incremental integration

(c) *Introduces several new programming interfaces for rapid prototyping ubiquitous computing interactions including a light scripting language and a visual programming environment.*

Quartz Composer uses a pipe-and-filter metaphor to rapidly prototype motion graphics. This dissertation shows that it is also useful to define physical prototypes. This interface is preferred in our user studies over an alternative textual programming environment. The textual programming environment was better suited for defining interactions modeled by finite state machines.

New interfaces support rapid prototyping

4. *Redefines the expressiveness of input devices so that it can be calculated using physical properties of the device instead of empirical thresholds.*

(a) *Demonstrates how to use expressiveness to structure the evaluation of prototype pointing devices to be able to make conclusions about how the device will function after further refinement.*

Expressiveness can be calculated using the resolution and sampling rate of the input device. As long as a pointing task during evaluation does not extend past the expressiveness of the input device, the Fitts' law regression can be used to make conclusions about future improvements of the input technique with respect to resolution and sampling rate.

Expressiveness as a design and evaluation tool

5. *Demonstrates how to apply player-centered iterative design to pervasive game development*

(a) *Illustrates a range of low-fidelity, and limited functionality prototypes that can be used to evaluate parts of the game earlier in the design process.*

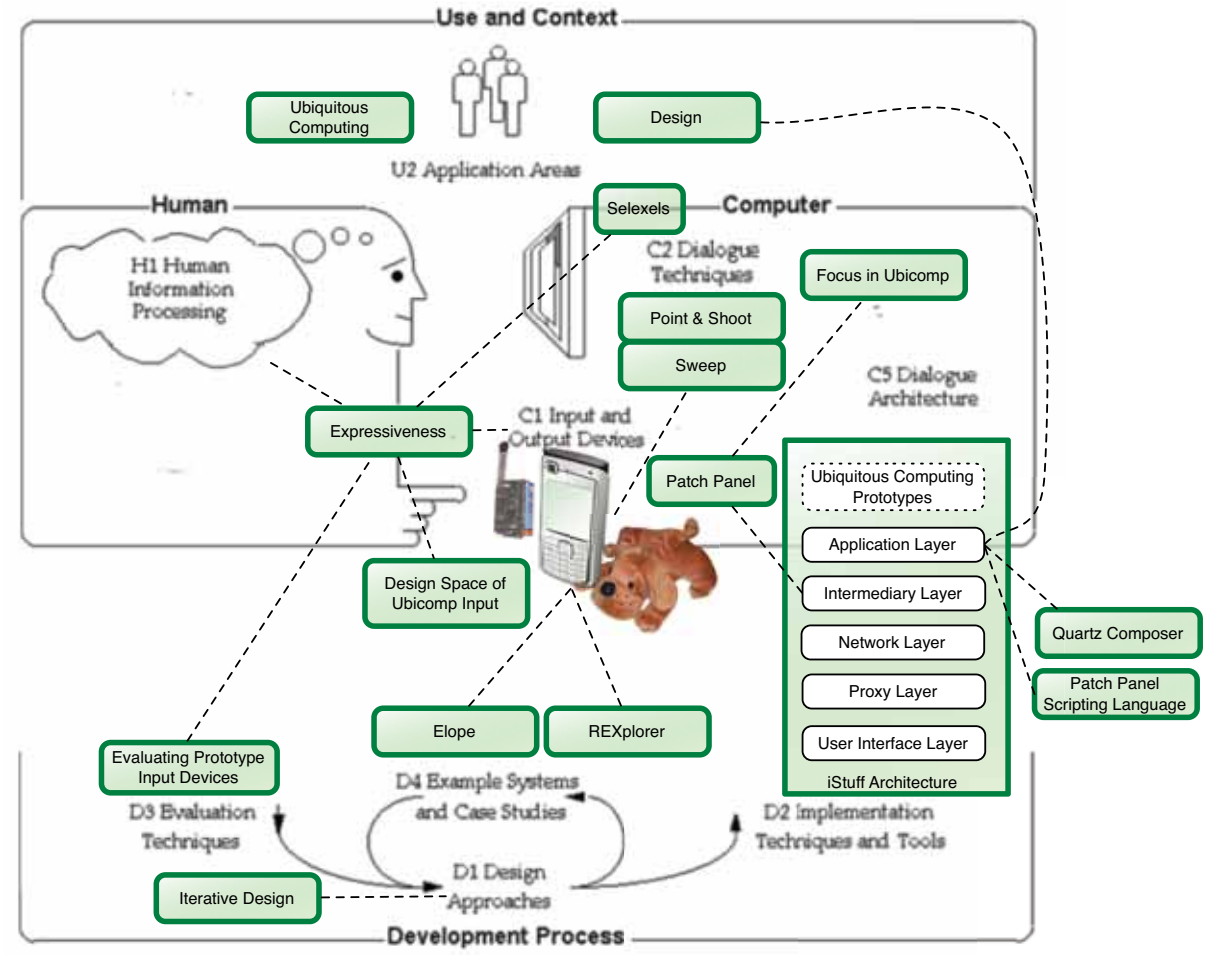


Figure 6.1: A map showing the contributions of this thesis mapped over the ACM's map of Human–Computer Interaction.

Bring theory to practice with real world examples

Iterative design is difficult to apply to ubiquitous computing applications. Through this example, we walk the reader through the use of iterative design from drawing board to deployment.

6.2 Future Work

Use toolkits to explore the design space

The iStuff architecture can be applied to explore the design space of ubiquitous computing applications. Because the iStuff architecture lowers the threshold for physical interface development and makes development available to a broader user base, there is an opportunity for application research. This architecture and resulting toolkits can be used for creative design, technology transfers, classroom educational scenarios, and longitudinal deployment and evaluation of these systems.

iStuff Mobile can be used to explore the gaps identified in the design space

of ubiquitous mobile input techniques. For example, using the mobile phone for more 3D position and orientation interactions by attaching appropriate 3D sensors.

Continue Quartz Composer extentions

Quartz Composer can continue to be extended to address any physical prototyping gaps in the programming environment. A more thorough evaluation that compares Quartz Composer to other rapid prototyping environments such as the statechart editor from d.tools, programming by demonstration with Exemplar, or other pipe-and-filter environments such as Max/MSP would benefit the toolkit community.

Further explore selection space pointing

Further exploration of pointing performance under selexels would be valuable. The user studies presented in this work only consider point cursors in selexel space, even though the active region spanned a screen region in display space. Confirmation is required that previous findings regarding fixed area cursors or dynamically expanding area cursors also apply to area cursors in Selection space.

Animation for selexel cursor motion

It would also be valuable to explore using animation to smoothen selexel cursor movement. Animation may introduce delay and create ambiguity in terms of the active selexel during a transition between selexel regions, however it may provide a smoother interaction experience for the user.

Toolkit support for adaptive selexel UI layout

Building on the expressiveness work, we are currently working on toolkit support of the selexel framework, allowing applications to dynamically adjust their selexel resolution based on the input device used. An early proof-of-concept of these ideas has been demonstrated by Jenabi (2006). This framework shows that it is possible to adapt a user interface at run-time to match the capabilities of an input device. More work is required to make this a robust application framework for other developers to depend on.

Evaluation of pervasive games

In our continued research on REXplorer, we are examining different evaluation strategies for pervasive games. We plan on performing a long-term study examining how usage patterns emerge over large sets of users. We also plan to examine the effectiveness of using a game simulator in a controlled setting to identify design flaws in a pervasive game. The controlled setting is not "ecologically valid", but is much cheaper to perform than a full-scale field study. The simulator also allows the developer to be location independent and not tied to their intended deployment location during development.

6.3 Closing Remarks

Ubiquitous computing has the potential to change the way we live, work, and play. In order to tap into this potential, we need to design applications that are useful, efficient, and enjoyable. The human-centered iterative design process has been effective in the desktop domain, but has remained elusive for ubiquitous computing because of a number of challenges. This

dissertation reduces the gap by contributing to each phase of the iterative design process, and demonstrates that iterative design can be an effective design strategy by showing a product that has been taken from drawing board to deployment.

Appendix A

iStuff Hardware Schematics

The original iStuff hardware was designed to use commercially available electronic components that were relatively easy to reconstruct for anyone with basic soldering skills. These designs are fairly elementary but are included here for completeness to allow other interested designers to recreate the original hardware set.

A.1 iButton

The iButton consists of a transmitter assembly activated by a button. When the button is depressed, the transmitter sends its identifying 8-bit number to a receiver. In software connected to the receiver, an action can be paired with the ID such that the button can perform a useful task. The identifying number is specified by setting data pins (D0 - D7) high or low according to the binary version of the number (1= high, 0 = low). For our iButtons, although not specified in the schematic, we have hardwired the appropriate data pins to high such that each iButton has it's own unique ID. The default state of a disconnected data pin is low.

The iButton circuit (see Figure A.1) consists of the following components.

- Ming TX-99 transmitter[1]
- HT640 encoder chip[2]
- $3V$ watch battery and holder
- Resistor (51 $k\Omega$)

[1] `http://www.rentron.com/remote_control/TX-99.htm`
[2] `http://www.rentron.com/remote_control/Holtek.htm`

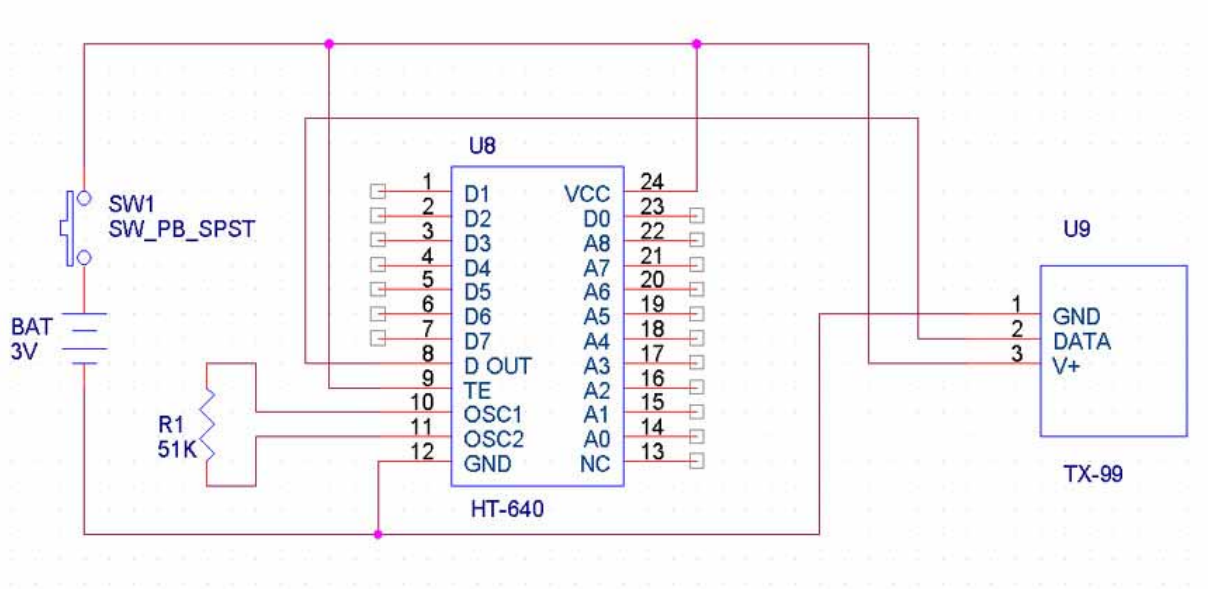


Figure A.1: The circuit diagram for the iButton.

A.2 iDog

The iDog was a slight variation of the iButton, where the button was replaced with a binary gravity sensor. The circuit was sewn into a stuffed animal such that when the dog was handled, it behaved as a pressed iButton.

A.3 iLight

The iLight is a portable Ming 300MHz RF receiver and linked actuator. It listens for preset ON and OFF numerical codes sent by Ming 300MHz transmitters, and when it receives one of these codes, the actuator is correspondingly turned on or off. The device is designed to control many different types of actuators (light, buzzer, vibrator, anything requiring simple on/off control), providing a standard Molex connection with one pin power and another, ground. We refer to this device as the iLight, because our actuator at the moment happens to be a bright LED. The ON code for the device is set by using an 8-bit dip switch. The OFF code is this same 8-bit dip switch number, only with the least significant bit inverted. While the iLight responds to its preset on and off codes, it is unaffected by other transmitted codes. Power on the iLight in our laboratory is provided by a 9V 300mA AC adapter, but the power source could easily be a battery. The circuitry requires a supply voltage of at least 5V to enable logic, however, a larger supply voltage may be needed depending on the actuator used.

The iLight circuit (see Figure A.2) consists of the following components.

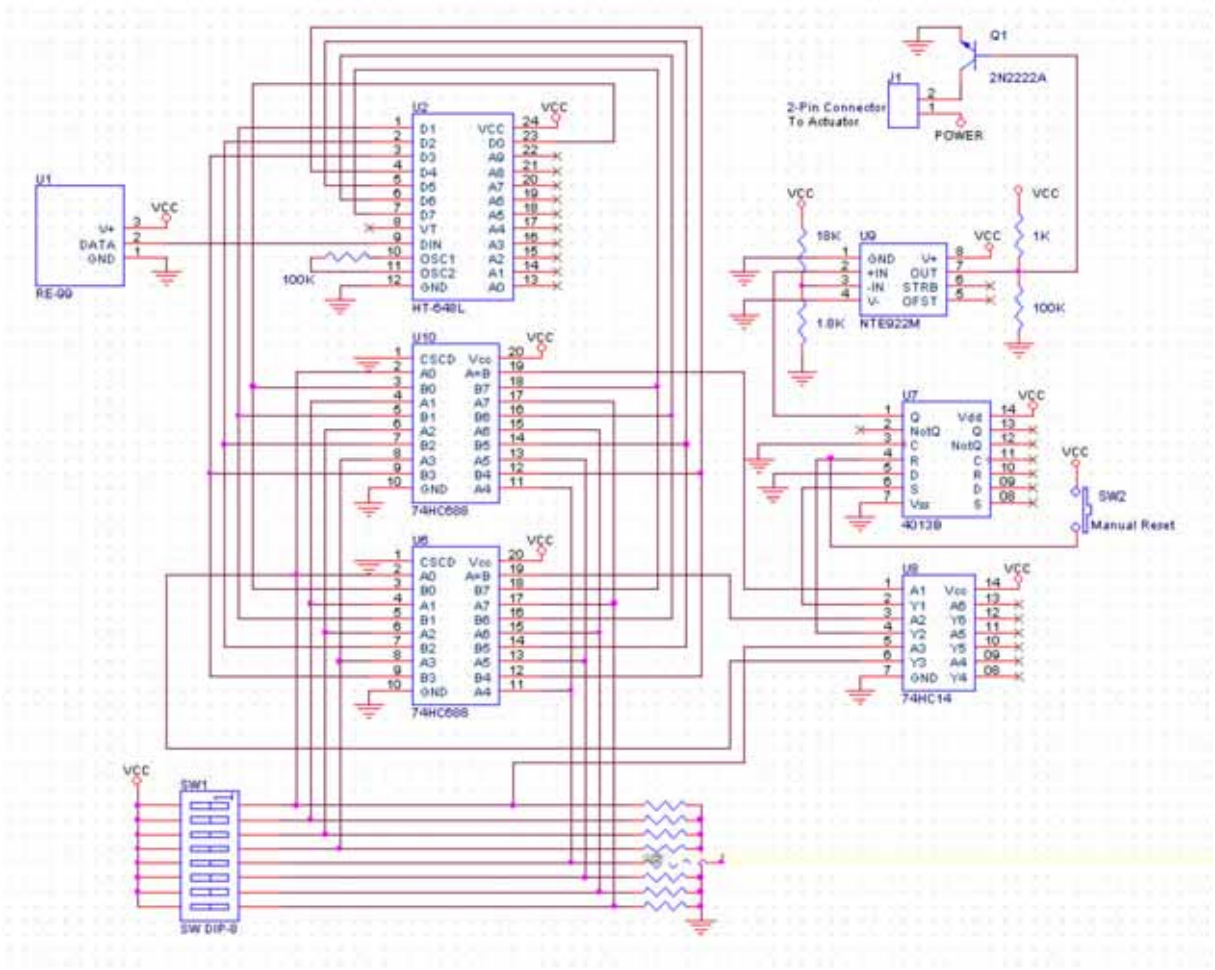


Figure A.2: The circuit diagram for the iLight.

- Ming RE-99 receiver[3]
- HT-648L decoder[4]
- 74HC688 8-bit comparator (Need 2)
- 8-bit dip switch
- 10 100$k\Omega$ resistors
- 74HC14 Schmidt-triggered hex inverter
- 4013B dual D flip-flop
- NTE922M voltage comparator
- 2N2222A transistor
- Molex connector of your choice
- Momentary push-button switch

[3] `http://www.rentron.com/remote_control/RE-99.htm`
[4] `http://www.rentron.com/remote_control/Holtek.htm`

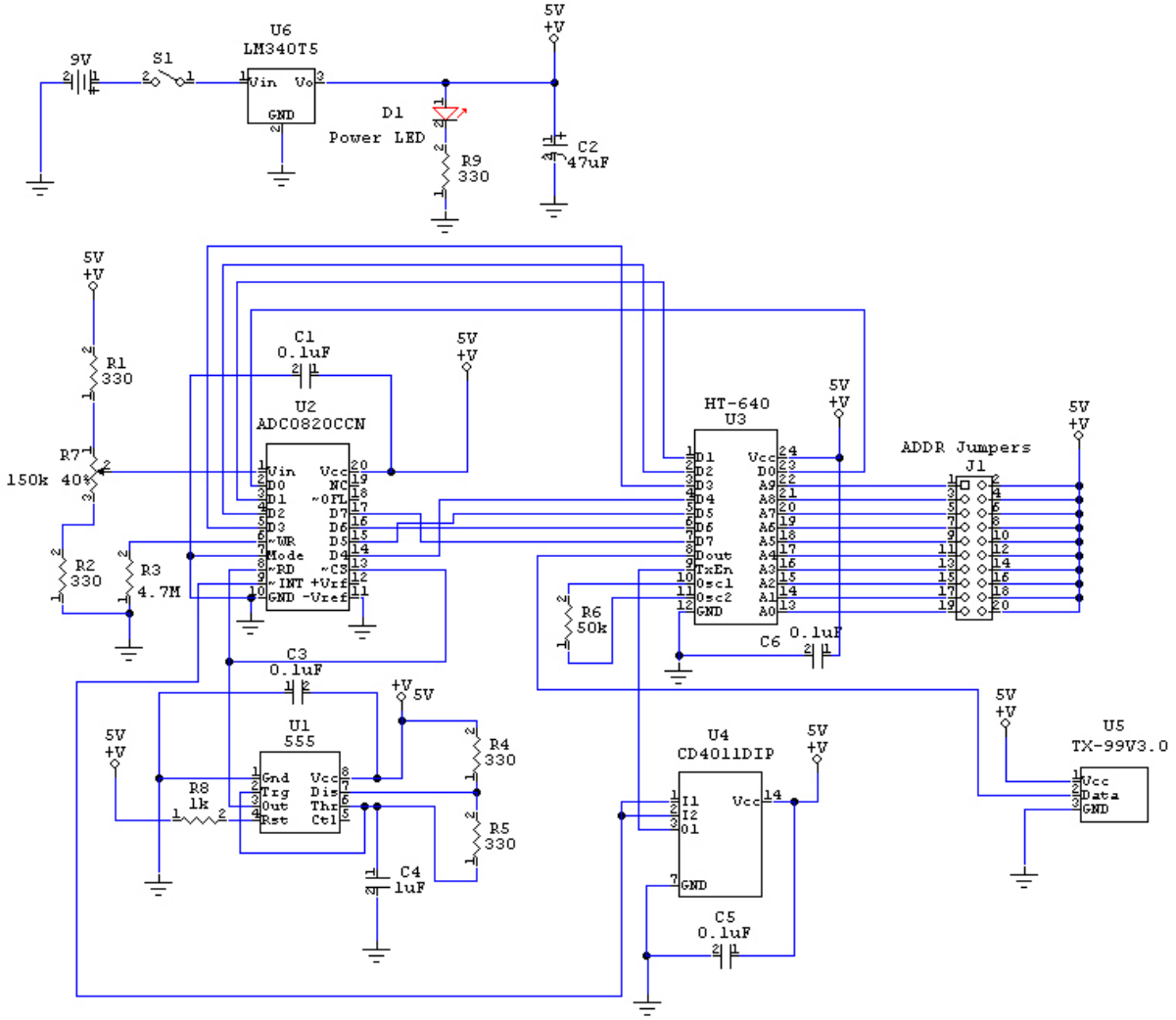


Figure A.3: The circuit diagram for the iSlider.

- $18k\Omega$ resistor
- $1.8k\Omega$ resistor
- $1k\Omega$ resistor

A.4 iSlider

The iSlider is a one dimensional input device that transmits data using the 300MHz Ming transmitter. The circuit (see Figure A.2) included of the following main components.

- Ming TX-99 transmitter[5]

[5] `http://www.rentron.com/remote_control/TX-99.htm`

- HT640 encoder chip[6]
- LED (to indicate when on)
- Sliding Potentiometer

A.5 iStuff Proxy Receiver

The receiver board (see Figure A.4) enabled communication between the various input devices (such as the iButton, or the iSlider) to a desktop computer. The construction included a Ming RE-99 receiver, a decoder chip, and an ActiveWire USB interface board. This receiver supports iStuff that broadcast using the Ming TX-99 transmitter.

- Ming RE-99 receiver
- ActiveWire USB card[7]
- HT-648L decoder
- LM340T5 5V power regulator
- 2N2222A transistor
- LED
- 120Ω resistor
- $33k\Omega$ resistor
- $100k\Omega$ resistor
- $9V$ power supply

A.6 iStuff Proxy Transmitter

The transmitter board (see Figure A.5) enabled communication between the desktop computer and various output devices (such as the iLight, or iBuzzer). The construction included a Ming TX-99 transmitter, an encoder chip, and an ActiveWire USB interface board. This transmitter can send information to iStuff with Ming RE-99 receivers.

- Ming TX-99 Transmitter
- ActiveWire USB card

[6] `http://www.rentron.com/remote_control/Holtek.htm`
[7] `http://www.activewireinc.com/`

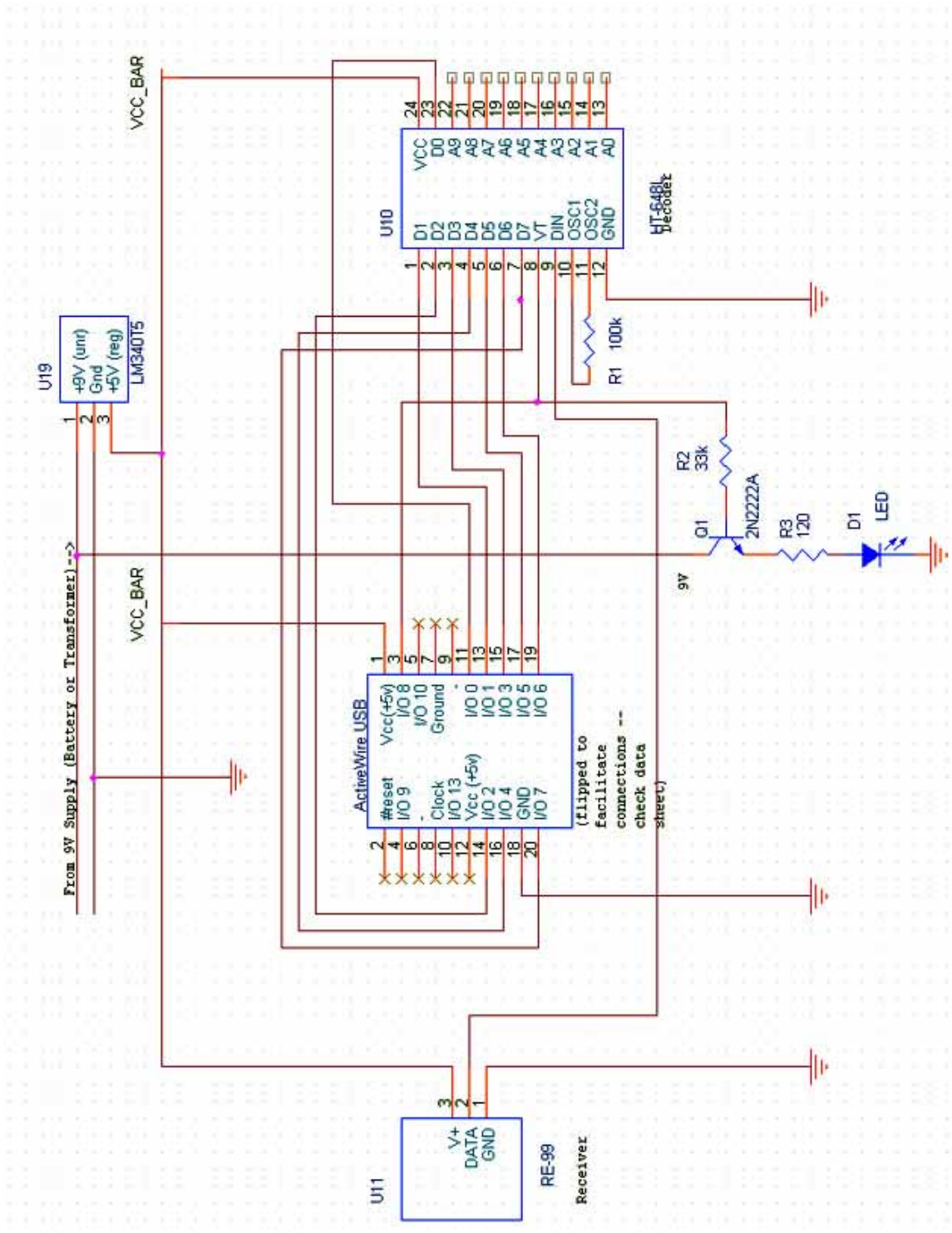


Figure A.4: The circuit diagram for the iStuff proxy reciever.

- HT-640 encoder
- LM340T5 5V power regulator
- 2N2222A transistor
- LED
- 120Ω resistor
- $33k\Omega$ resistor
- $100k\Omega$ resistor
- $9V$ power supply

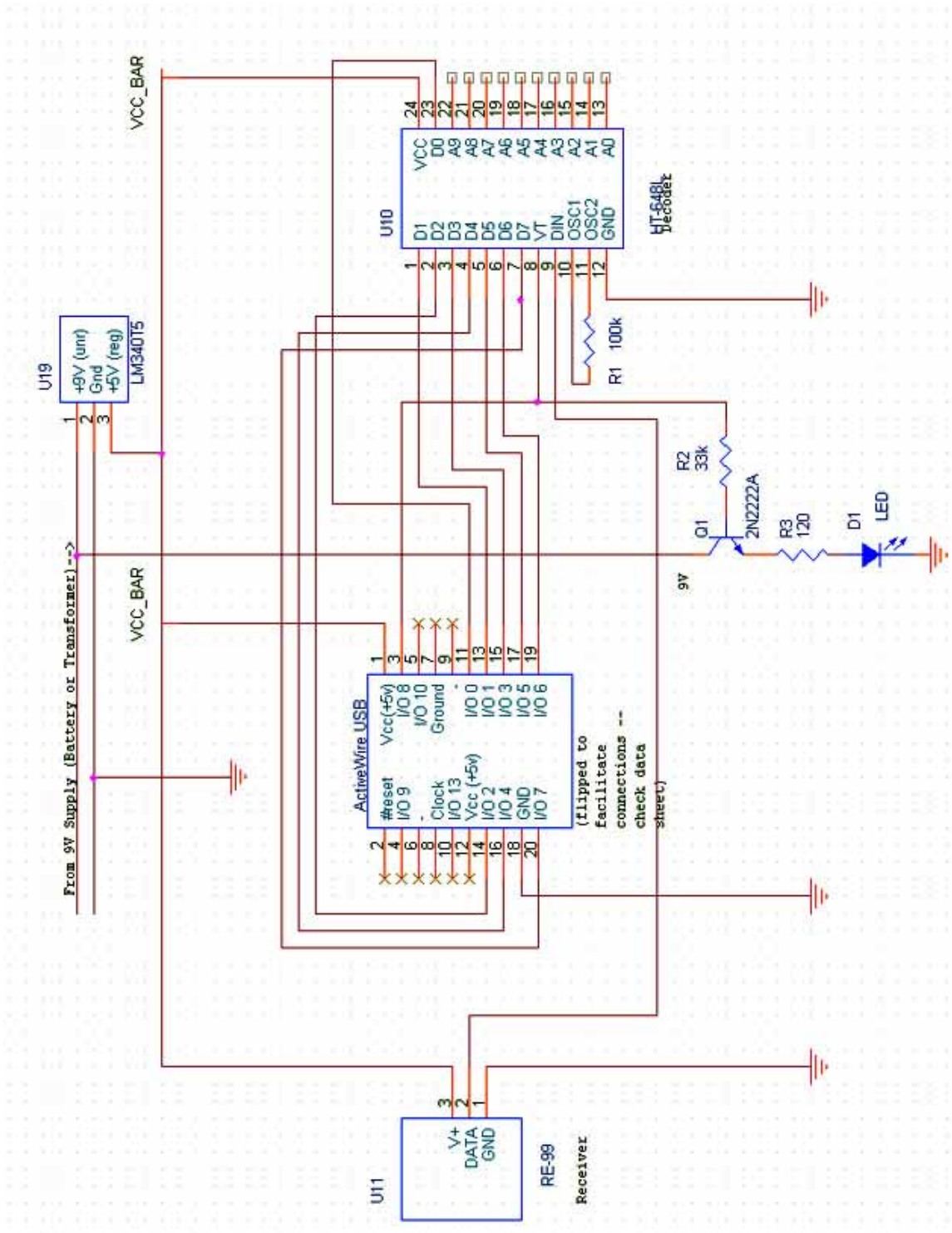


Figure A.5: The circuit diagram for the iStuff proxy transmitter.

Appendix B

iStuff Evaluation and Scenario Descriptions

User tests for the Patch Panel GUI and the Patch Panel Script language

Welcome to the user study of the iStuff project at the Media Computing Group and thank you for your participation. We are happy to give you an introduction into our latest development in the field of supporting the rapid prototyping process for ubiquitous environments: A new graphical interface for the *Patch Panel* and a command line wrapper responsible to facilitate the management of iStuff proxies, the *Proxy Manager*.

0. Important note

Important note on the test in that you take part for us: **It is not you to be judged or evaluated; it is our design of the software we hand to you!** So do not be afraid to criticize the development environment.
This lesson should include the benefit for you that you become familiar with the development suite and for us that we can justify a user evaluation and incorporate today's results of a user evaluation into our work.

The tasks you perform are recorded in terms of a screen capture and audio recording. Please do not feel offended by these techniques and just try to ignore them. We will not attempt to identify or judge your behavior with these recordings. They only represent a post-evaluation aid for us to see how you have derived a certain solution. You are not filmed explicitly. The iSight vision is turned off!

Figure B.1: Page 1 of description of the evaluation handed out to the participants - Introduction

1. Motivation

In order to give a motivation for the following scenarios we would like you to recreate certain design tasks. Please let your fantasy go a little beyond what you directly see and the context in which you perform your tasks.

Ithat you are part of a design team in a company that wants to produce a new type of mobile phones. The new design should include different sensors and explore new ways of interaction with the phone. Another research field lies in security systems as well as in controlling robotic assistance systems.

One week, your advisor enters your office and presents you different scenarios and asks you to evaluate them. The scenarios are presented in written form as you can see below. You take the papers he gives you and start thinking about different configurations and what types of input and output devices you will need.
With the help of the iStuff prototyping suite you want to build a first hardware prototype that can be shown to different test users. The results of these user studies should justify your current design or help to discover weaknesses or strengths that influence future designs. You know that your prototypes will look a little awkward but it is clear to you and also to your test users that the functionality matters, not the look of the device. If the interaction is well designed, a later product that integrates all the tested capabilities can be derived from that early design.

So, just go ahead and explore the possibilities of the *Patch Panel* and create whatever you think to be meaningful. And please remember: At that stage there is not right or wrong design, it has to be found out later. You just want to build something robustly working.

Figure B.2: Page 2 of description of the evaluation handed out to the participants - Motivation

Scenario 1: Control a multi screen presentation controlled via a mobile phone and Phidgets.

A presentation software running on two different machines should show the same set of slides. The screen of the left machine should display the last slide and the one on the right should show the current slide that is talked about. In order to control the slide transitions, the use of mouse and keyboard should be avoided. Instead, the presentations should be controlled via the presenter's mobile phone keys. An alternative to this design is the installation of pressure and touch sensors on the floor, the speaker has to step onto. A light sensor would also be possible.

Discover the event types of the other events through the Event Logger.
And also discover the mobile phone key values by examining the appropriate events on the Event Logger.

Scenario 2: Tilt to scroll (SmartIts sensor board)

Augment a mobile phone in the following way:
A SmartIts sensor boards is attached to the mobile phone.
When the phone is tilted upwards or downwards and a pressure threshold is crossed, the menu display should start scrolling in upward direction and downwards, respectively. Depending on the degree of tilting, the scrolling speed changes. In a first prototype, it should be differentiated between at least two scrolling speeds.

Figure B.3: Page 3 of description of the evaluation handed out to the participants - Scenarios 1 and 2

Scenario 3: Prototype a new iPod Shuffle using iTunes and Phidgets.

You are asked to build a new iPod Shuffle interface with a new generation of sensors. You decide to use Phidgets sensors to control the new version of the iPod Shuffle.
As a representation of the music player device, you use the iTunes application coming with your Mac OS distribution. An iStuff proxy that controls the application is available. Basic interaction such as Play/Pause, Next/Previous Track and the increase and decrease of volume are to be implemented in the scope of this scenario.

Scenario 4: Control motor with SmartIts and Phidgets).

An automatic balance control system for boats should be developed. Whenever the boat is tilting too much to the side, the motor should move a counter weight attached to it in order to balance it again.

Make use of the SmartIts Accelerometers and the Phidgets Servo Motor Controller for this scenario. As an extension, you could also make use of other extensions with Phidgets.

Figure B.4: Page 4 of description of the evaluation handed out to the participants - Scenarios 3 and 4

Appendix C

Post-participation Questionnaire

Post participation questionnaire

1. Would you say that the development capabilities with the Quartz Composer approach are better than with the script?

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

2. Could you imagine more scenarios where ubiquitous devices are connected and configured in the presented way?

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

3. How do you like the data flow metaphor of the Quartz Composer?

It was a strong help	It was useful	Don't know	It was no help	It absolutely made no sense to me
( )	( )	( )	( )	( )

4. Would you say that you used lots of the Quartz Composer original patches?

I used them a lot	I used them	equal to the new one	seldom	not at all
( )	( )	( )	( )	( )

5. The general concept of the iStuff project (Event Heap infrastructure, distributed exchange of information with events, etc.) was understandable.

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

6. How would you judge the future extensibility of the graphical approach?

Very extensible	extensible	don't know	hardly extensible	none at all
( )	( )	( )	( )	( )

7. I would prefer the **graphical approach** over the scripting approach

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

8. I would prefer the **scripting approach** over the graphical approach

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

9. Would you say the script approach is more flexible or powerful?

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

Figure C.1: Page 1 of the questionnaire handed out to the participants after the user test

10. What approach encourages you to go through many iterations?

Graphical Appraoch (Quartz Composer)	Scripting Approach
( )	( )

11. What would you like to be changed in future version?

12. What features would you add?

13. What patches were you missing during the prototyping process?

14. Additional comments: (Write whatever you think!)

15. How would you judge your former experience with Quartz Composer (1 = No at all, 5 = Quartz Composer expert)?

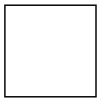


Figure C.2: Page 2 of the questionnaire handed out to the participants after the user test

Bibliography

Abowd, G. D. (1999). Software Engineering Issues for Ubiquitous Computing. In *Proceedings of the 1999 International Conference on Software Engineering.*, pages 75–84, Los Alamitos, CA, USA. IEEE Computer Society Press.

Abowd, G. D., Hayes, G. R., Iachello, G., Kientz, J. A., Patel, S. N., Stevens, M. M., and Truong, K. N. (2005). Prototypes and Paratypes: Designing Mobile and Ubiquitous Computing Applications. *IEEE Pervasive Computing*, **4**(4), 67–73.

Agilent Technologies (2004). ADNS-2610 Optical Mouse Sensor. *(http://cp.literature.agilent.com/litweb/pdf/5988-9774EN.pdf)*.

Akers, D. (2006). CINCH: a cooperatively designed marking interface for 3D pathway selection. In *UIST '06: Proceedings of the 19th annual ACM Symposium on User Interface Software and Technology*, pages 33–42, New York, NY, USA. ACM Press.

Apple Computer Inc. (1992). *MacIntosh Human Interface Guidelines*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Ballagas, R., Ringel, M., Stone, M., and Borchers, J. (2003). iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments. In *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 537–544, New York, NY, USA. ACM Press.

Ballagas, R., Szybalski, A., and Fox, A. (2004). Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments. In *PerCom '04: Proceedings of the 2nd annual IEEE Conference on Pervasive Computing and Communications*. IEEE.

Ballagas, R., Rohs, M., Sheridan, J. G., and Borchers, J. (2005). Sweep and Point & Shoot: Phonecam-based interactions for large public displays. In *CHI '05: Extended Abstracts of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1200–1203, New York, NY, USA. ACM Press.

Ballagas, R., Rohs, M., Sheridan, J., and Borchers, J. (2006). The Smart Phone: A Ubiquitous Input Device. *IEEE Pervasive Computing*, **5**(1), 70–77.

Ballagas, R., Memon, F., Reiners, R., and Borchers, J. (2007a). iStuff Mobile: Rapidly Prototyping New Mobile Phone Interfaces for Ubiquitous

Computing. In *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1107–1116, New York, NY, USA. ACM Press.

Ballagas, R., Walz, S. P., Kratz, S. G., Fuhr, C. O., Yu, E., Tann, M., Borchers, J., and Hovestadt, L. (2007b). REXplorer: A Mobile, Pervasive Spell-Casting Game for Tourists. In *CHI '07: Extended Abstracts on Human Factors in Computing Systems*, pages 1929–1934, New York, NY, USA. ACM Press.

Ballagas, R., Rohs, M., Sheridan, J., and Borchers, J. (2008). The Design Space of Mobile Phone Input Techniques for Ubiquitous Computing. In J. Lumsden, editor, *Handbook of Research on User Interface Design and Evaluation for Mobile Technologies (To appear)*. IGI Global, Hershey, PA, USA.

Bartlett, J. F. (2000). Rock 'n' Scroll Is Here to Stay. *IEEE Comput. Graph. Appl.*, **20**(3), 40–45.

Bass, L., Clements, P., and Kazman, R. (1998). *Software Architecture in Practice*. Addison Wesley.

Bates, J., Bacon, J., Moody, K., and Spiteri, M. (1998). Using events for the scalable federation of heterogeneous components. In *Proceedings of the 8th ACM SIGOPS European Workshop on Support for Composing Distributed Applications*, pages 58–65. ACM Press.

Baudisch, P., Cutrell, E., Hinckley, K., and Eversole, A. (2005). Snap-and-go: helping users align objects without the modality of traditional snapping. In *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 301–310, New York, NY, USA. ACM Press.

Benford, S., Seagar, W., Flintham, M., Anastasi, R., Rowland, D., Humble, J., Stanton, D., Bowers, J., Tandavanitj, N., Adams, M., *et al.* (2004). The Error of Our Ways: The Experience of Self-Reported Position in a Location-Based Game. In *Ubicomp '04: Proceedings of the 6th Int'l Conference on Ubiquitous Computing*, pages 70–87. Springer.

Benford, S., Magerkurth, C., and Ljungstrand, P. (2005). Bridging the physical and digital in pervasive gaming. *Commun. ACM*, **48**(3), 54–57.

Beyer, H. and Holtzblatt, K. (1998). *Contextual Design: Defining Customer-centered Systems*. Morgan Kaufmann, San Francisco, CA, USA.

Bier, E. A. and Freeman, S. (1991). MMM: A user interface architecture for shared editors on a single screen. In *UIST '91: Proceedings of the 4th Annual Symposium on User Interface Software and Technology*, pages 79–86. ACM.

Bjork, S. and Holopainen, J. (2005). *Patterns in game design*. Charles River Media, Hingham, MA, USA.

Blanch, R., Guiard, Y., and Beaudouin-Lafon, M. (2004). Semantic pointing: improving target acquisition with control-display ratio adaptation. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 519–526, New York, NY, USA. ACM Press.

Borchers, J. (2001). *A pattern approach to interaction design*. John Wiley & Sons, New York, NY, USA.

Buchenau, M. and Suri, J. F. (2000). Experience prototyping. In *DIS '00: Proceedings of the conference on Designing interactive systems*, pages 424–433, New York, NY, USA. ACM Press.

Buxton, W. (1983). Lexical and pragmatic considerations of input structures. *SIGGRAPH Comput. Graph.*, **17**(1), 31–37.

Buxton, W. A. and Sniderman, R. (1980). Iteration in the design of the human-computer interface. In *Proceedings of the 13th Annual Meeting of the Human Factors Association of Canada*, pages 72–81. HFAC.

Cao, X. and Balakrishnan, R. (2003). VisionWand: interaction techniques for large displays using a passive wand tracked in 3D. In *Proceedings of the 16th annual ACM Symposium on User Interface Software and Technology*, pages 173–182. ACM Press.

Card, S. K., English, W. K., and Burr, B. J. (1978). Evaluation of mouse, rate controlled isometric joystick, step keys and text keys for text selection on a CRT. *Ergonomics*, **21**, 601–613.

Card, S. K., Newell, A., and Moran, T. P. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, USA.

Card, S. K., Mackinlay, J. D., and Robertson, G. G. (1991). A morphological analysis of the design space of input devices. *ACM Trans. Inf. Syst.*, **9**(2), 99–122.

Carter, S., Churchill, E., Denoue, L., Helfman, J., and Nelson, L. (2004). Digital graffiti: public annotation of multimedia content. In *CHI '04: Extended abstracts of the SIGCHI Conference on Human Factors and Computing Systems*, pages 1207–1210. ACM Press.

Carter, S., Mankoff, J., Klemmer, S., and Matthews, T. (2007a). Exiting the cleanroom: On ecological validity and ubiquitous computing. *Human-Computer Interaction (To appear)*.

Carter, S., Mankoff, J., and Heer, J. (2007b). Momento: Support for Situated Ubicomp Experimentation. In *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 125–134, New York, NY, USA. ACM Press.

Cassinelli, Á., Perrin, S., and Ishikawa, M. (2005). Smart laser-scanner for 3D human-machine interface. In *CHI '05: Extended abstracts of the SIGCHI Conference on Human Factors and Computing Systems*, pages 1138–1139, New York, NY, USA. ACM Press.

Chaos Computer Club (2002). Blinkenlights. http://www.blinkenlights.de.

Chen, H., Perich, F., Finin, T., and Joshi, A. (2004). SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services*, Boston, MA. IEEE.

Consolvo, S., Roessler, P., and Shelton, B. (2004). The CareNet Display: Lessons Learned from an In Home Evaluation of an Ambient Display. In *Ubicomp '04: Proceedings of the 6th Int'l Conference on Ubiquitous Computing*, volume 4, pages 1–17. Springer.

Czerwinski, M., Robertson, G., Meyers, B., Smith, G., Robbins, D., and Tan, D. (2006). Large display research overview. In *CHI '06: Extended Abstracts on Human Factors in Computing Systems*, pages 69–74, New York, NY, USA. ACM Press.

Dahlbäck, N., Jönsson, A., and Ahrenberg, L. (1993). Wizard of Oz studies: why and how. In *IUI '93: Proceedings of the 1st International Conference on Intelligent User Interfaces*, pages 193–200, New York, NY, USA. ACM Press.

de Souza e Silva, A. and Delacruz, G. C. (2006). Hybrid Reality Games Reframed. Potential Uses in Educational Contexts. *Games and Culture*, **1**(3), 231–251.

Dix, A., Finley, J., Abowd, G., and Beale, R. (2004). *Human-Computer Interaction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Edwards, W. K., Newman, M. W., Sedivy, J., and Izadi, S. (2002). Challenge: recombinant computing and the speakeasy approach. In *MobiCom '02: Proceedings of the 8th Annual Int. Conf. on Mobile Computing and Networking*, pages 279–286. ACM Press.

Edwards, W. K., Bellotti, V., Dey, A. K., and Newman, M. W. (2003). The challenges of user-centered design and evaluation for infrastructure. In *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 297–304, New York, NY, USA. ACM Press.

Epstein, M. and Vergani, S. (2006). Mobile Technologies and Creative Tourism. In *AMICS '06: Proceedings of the 12th Americas Conference on Information Systems*. Association for Information Systems.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, **47**, 381–391.

Foley, J. D., Wallace, V. L., and Chan, P. (1984). The human factors of computer graphics interaction techniques. *IEEE Comput. Graph. Appl.*, **4**(11), 13–48.

Fullerton, T., Swain, C., and Hoffman, S. (2004). *Game Design Workshop: Designing, Prototyping, and Playtesting Games*. CMP Books, San Francisco, CA.

Gajos, K. and Weld, D. S. (2004). SUPPLE: automatically generating user interfaces. In *IUI '04: Proceedings of the 9th International Conference on Intelligent User Interface*, pages 93–100, New York, NY, USA. ACM Press.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gellersen, H., Kortuem, G., Schmidt, A., and Beigl, M. (2004). Physical Prototyping with Smart-Its. *IEEE Pervasive Computing*, **3**(3), 74–82.

Gosling, J., Rosenthal, D., and Arden, M. (1989). Windows System Architecture: History, Terms and Concepts. In *The NeWS Book*, chapter 3, pages 23–52. Springer-Verlag.

Gould, J. and Lewis, C. (1985). Designing for usability: key principles and what designers think. *Communications of the ACM*, **28**(3), 300–311.

Greenberg, S. and Fitchett, C. (2001). Phidgets: Easy development of physical interfaces through physical widgets. In *UIST '01: Proceedings of the 14th annual ACM Symposium on User Interface Software and Technology*, pages 209–218, New York, NY, USA. ACM Press.

Grossman, T. and Balakrishnan, R. (2005). The Bubble Cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 281–290, New York, NY, USA. ACM Press.

Hansen, W. (1971). User engineering principles for interactive systems. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 523–532. AFIPS.

Harrison, B. L., Fishkin, K. P., Gujar, A., Mochon, C., and Want, R. (1998). Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. In *CHI '98: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 17–24, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A. L., and Gee, J. (2006). Reflective physical prototyping through integrated design, test, and analysis. In *UIST '06: Proceedings of the 19th annual ACM Symposium on User Interface Software and Technology*. ACM Press.

Hartmann, B., Abdulla, L., Mittal, M., and Klemmer, S. R. (2007). Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 145–154, New York, NY, USA. ACM Press.

Heiner, J. M., Hudson, S. E., and Tanaka, K. (1999). The Information Percolator: ambient information display in a decorative object. In *UIST '99: Proceedings of the 12th annual ACM Symposium on User Interface*

Software and Technology, pages 141–148, New York, NY, USA. ACM Press.

Hewett, T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., and Verplank, W. (1992). *ACM SIGCHI Curricula for Human-Computer Interaction*. ACM Press, New York, NY, USA.

Hinckley, K. and Horvitz, E. (2001). Toward more sensitive mobile phones. In *UIST '01: Proceedings of the 14th annual ACM Symposium on User Interface Software and Technology*, pages 191–192, New York, NY, USA. ACM Press.

Hu, J., Brown, M., and Turin, W. (1996). HMM based online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(10), 1039–1045.

Hudson, S., Fogarty, J., Atkeson, C., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J., and Yang, J. (2003). Predicting human interruptibility with sensors: a Wizard of Oz feasibility study. In *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 257–264, New York, NY, USA. ACM Press.

Hudson, S. E. and Mankoff, J. (2006). Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. In *UIST '06: Proceedings of the 19th annual ACM Symposium on User Interface Software and Technology*, pages 289–298, New York, NY, USA. ACM Press.

Humble, J., Crabtree, A., Hemmings, T., Akesson, K.-P., Koleva, B., Rodden, T., and Hansson, P. (2003). "Playing with the Bits" User-Configuration of Ubiquitous Domestic Environments. In *Ubicomp '03: Proceedings of the 5th Int. Conf. on Ubiquitous Computing*, pages 256–263. Springer-Verlag.

Hummels, C. C. M. (2000). *An exploratory expedition to create engaging experiences through gestural jam sessions*. Ph.D. thesis, Delft University of Technology.

Ionescu, A., Stone, M., and Winograd, T. (2002). WorkspaceNavigator: Capture, Recall, and Reuse using Spatial Cues in an Interactive Workspace. Technical Report 2002-04, Stanford University, Computer Science Department, Stanford, CA, USA.

ISO (2000). Ergonomic requirements for office work with visual display terminals (VDTs) - Requirements for non-keyboard input devices. *ISO 9241-9*.

Jellinek, H. D. and Card, S. K. (1990). Powermice and user performance. In *CHI '90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 213–220, New York, NY, USA. ACM Press.

Jenabi, M. (2006). *Selexels: Adapting User Interfaces to Mobile Input Devices*. Master's thesis, RWTH Aachen University, Aachen, Germany.

Jiang, H., Ofek, E., Moraveji, N., and Shi, Y. (2006). Direct pointer: direct manipulation for large-display interaction using handheld cameras. In *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1107–1110, New York, NY, USA. ACM Press.

Johanson, B. and Fox, A. (2002). The Event Heap: A Coordination Infrastructure for Interactive Workspaces. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 83. IEEE.

Johanson, B. and Fox, A. (2004). Extending tuplespaces for coordination in interactive workspaces. *J. Syst. Softw.*, **69**(3), 243–266.

Johanson, B., Fox, A., and Winograd, T. (2002a). The interactive workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, **1**, 67–74.

Johanson, B., Hutchins, G., Winograd, T., and Stone, M. (2002b). Pointright: experience with flexible input redirection in interactive workspaces. In *UIST '02: Proceedings of the 15th annual ACM Symposium on User Interface Software and Technology*, pages 227–234, New York, NY, USA. ACM Press.

Kabbash, P. and Buxton, W. A. S. (1995). The "Prince" technique: Fitts' law and selection using area cursors. In *CHI '95: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 273–279, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Kao, R., Sarigumba, D., Kao, M.-C., and Sarigumba, Y. (2007). *BlackBerry Pearl For Dummies*. Wiley Publishing, Hoboken, NJ, USA.

Karat, C.-M. (1990). Cost-benefit analysis of iterative usability testing. In *INTERACT '90: Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, pages 351–356. North-Holland.

Karpov, E., Kiss, I., Leppänen, J., Olsen, J., Oria, D., Sivadas, S., and Tian, J. (2006). Short message dictation on Symbian Series 60 mobile phones. In *ICMI '06: Proceedings of the 8th international conference on Multimodal interfaces*, pages 126–127, New York, NY, USA. ACM Press.

Kelley, J. F. (1984). An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems (TOIS)*, **2**(1), 26–41.

Kelley, T. and Littman, J. (2001). *The art of innovation*. Doubleday Broadway, New York, NY, USA.

Kidd, C. D., Orr, R., Abowd, G. D., Atkeson, C. G., Essa, I. A., MacIntyre, B., Mynatt, E. D., Starner, T., and Newstetter, W. (1999). The aware home: A living laboratory for ubiquitous computing research. In *CoBuild '99: Proceedings of Second International Workshop on Cooperative Buildings*, pages 191–198. Springer.

Kindberg, T. and Fox, A. (2002). System software for ubiquitous computing. *IEEE Pervasive Computing*, **1**(1), 70–81.

Kjeldskov, J. and Graham, C. (2003). A review of mobile HCI research methods. In *Mobile HCI '03: Proceedings of 5th International Symposium on Human-Computer Interaction with Mobile Devices and Services*, Lecture Notes in Computer Science, pages 317–335. Springer.

Kjeldskov, J. and Stage, J. (2004). New techniques for usability evaluation of mobile systems. *International Journal of Human-Computer Studies*, **60**(5-6), 599–620.

Klemmer, S. (2004). *Tangible user interfaces: Tools and techniques*. Ph.D. thesis, University of California, Berkeley.

Klemmer, S. R., Sinha, A. K., Chen, J., Landay, J. A., Aboobaker, N., and Wang, A. (2000). Suede: a Wizard of Oz prototyping tool for speech user interfaces. In *UIST '00: Proceedings of the 13th annual ACM Symposium on User Interface Software and Technology*, pages 1–10, New York, NY, USA. ACM Press.

Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. (2004). Papier-Mâché: Toolkit Support for Tangible Input. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 399–406, New York, NY, USA. ACM Press.

Klopfer, E. and Squire, K. (2005). Environmental Detectives—The Development of an Augmented Reality Platform for Environmental Simulations. *Educational Technology Research and Development*.

Landay, J. A. (1996). *Interactive sketching for the early stages of user interface design.* Ph.D. thesis, School of Computer Science, Carnegie Mellon University.

Lee, J. C., Avrahami, D., Hudson, S. E., Forlizzi, J., Dietz, P. H., and Leigh, D. (2004a). The Calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In *DIS '04: Proceedings of the 2004 Conference on designing interactive systems*, pages 167–175, New York, NY, USA. ACM Press.

Lee, Y. S., Smith-Jackson, T. L., Nussbaum, M. A., Tomioka, K., and Bhatkhande, Y. (2004b). Use of product-interactive focus groups for requirement capture and usability assessment. In *Proceedings of the 48th Annual Human Factors and Ergonomics Conference*, pages 2461–2465, New Orleans, LA.

Leichter, J. S. and Whiteside, R. A. (1989). Implementing Linda for distributed and parallel processing. In *ICS '89: Proceedings of the 3rd Int. Conf. on Supercomputing*, pages 41–49, New York, NY, USA. ACM Press.

Li, Y., Hong, J. I., and Landay, J. A. (2004). Topiary: a tool for prototyping location-enhanced applications. In *UIST '04: Proceedings of the 17th annual ACM Symposium on User Interface Software and Technology*, pages 217–226, New York, NY, USA. ACM Press.

Licoppe, C. and Inada, Y. (2006). Emergent Uses of a Multiplayer Location-aware Mobile Game: the Interactional Consequences of Mediated Encounters. *Mobilities*, **1**(1), 39–61.

Liu, L. and Khooshabeh, P. (2003). Paper or interactive?: a study of prototyping techniques for ubiquitous computing environments. In *CHI '03: Extended abstracts on Human Factors in Computing Systems*, pages 1030–1031, New York, NY, USA. ACM Press.

MacIntyre, B., Gandy, M., Dow, S., and Bolter, J. (2004). DART: a toolkit for rapid design exploration of augmented reality experiences. In *UIST '04: Proceedings of the 17th annual ACM Symposium on User Interface Software and Technology*, pages 197–206, New York, NY, USA. ACM Press.

MacKenzie, I. and Soukoreff, R. (2002). Text Entry for Mobile Computing: Models and Methods, Theory and Practice. *Human-Computer Interaction*, **17**(2), 147–198.

MacKenzie, I. S. (1991). *Fitts' Law as a Performance Model in Human-Computer Interaction*. Ph.D. thesis, Univeristy of Toronto, Toronto, Ontario, Canada.

MacKenzie, I. S. (1992). Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, **7**, 91–139.

MacKenzie, I. S. and Ware, C. (1993). Lag as a determinant of human performance in interactive systems. In *CHI '93: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 488–493. ACM Press.

Madhavapeddy, A., Scott, D., Sharp, R., and Upton, E. (2004). Using camera-phones to enhance human-computer interaction. In *Ubicomp '04: Adjunct Proceedings of the 6th International Conference on Ubiquitous Computing (Demos)*. Springer-Verlag.

Mankoff, J. and Schilit, B. (1997). Supporting knowledge workers beyond the desktop with palplates. In *CHI '97: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 550–551, New York, NY, USA. ACM Press.

Matthews, T. (2005). Interviewing peripheral display designers and developers. Technical Report EECS-2005-19, University of California, Berkeley.

Mayhew, D. (1991). *Principles and guidelines in software user interface design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Meyer, D. E., Smith, J. E. K., Kornblum, S., Abrams, R., and Wright, C. E. (1990). Speed-Accuracy Tradeoffs in Aimed Movements: Toward a Theory of Rapid Voluntary Action. In M. Jeannerod, editor, *Attention and Performance XIII*, pages 173–226. Lawrence Erlbaum, Hillsdale, N.J.

Miyaoku, K., Higashino, S., and Tonomura, Y. (2004). C-blink: a hue-difference-based light signal marker for large screen interaction via any mobile terminal. In *UIST '04: Proceedings of the 17th annual ACM Symposium on User Interface Software and Technology*, pages 147–156. ACM Press.

Moggridge, B. (2006). *Designing Interactions*. The MIT Press, Boston, MA, USA.

Munson, M. (1998). System Support for Composing Distributed Applications Using Events. Diploma Dissertation, Cambridge University, Cambridge, UK.

Myers, B., Hudson, S. E., and Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, **7**(1), 3–28.

Myers, B. A., Stiel, H., and Gargiulo, R. (1998). Collaboration using multiple PDAs connected to a PC. In *CSCW '98: Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pages 285–294. ACM Press.

Myers, B. A., Bhatnagar, R., Nichols, J., Peck, C. H., Kong, D., Miller, R., and Long, A. C. (2002). Interacting at a distance: measuring the performance of laser pointers and other devices. In *CHI '02: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 33–40, New York, NY, USA. ACM Press.

Mynatt, E., Rowan, J., Craighill, S., and Jacobs, A. (2001). Digital family portraits: supporting peace of mind for extended family members. In *CHI '01: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 333–340, New York, NY, USA. ACM Press.

Nichols, J. and Myers, B. A. (2006). Controlling home and office appliances with smart phones. *IEEE Pervasive Computing*, **5**(3), 60–67.

Nielsen, J. (1993). Iterative user-interface design. *Computer*, **26**(11), 32–41.

Nielsen, J. and Faber, J. (1996). Improving system usability through parallel design. *Computer*, **29**(2), 29–35.

Norman, D. (2002). *The design of everyday things*. Basic Books.

Object Management Group (2004). The Common Object Request Broker: Architecture and Specification. revision 3.0.3.

Oh, I. and Suen, C. (2002). A class-modular feedforward neural network for handwriting recognition. *Pattern Recognition*, **35**(1), 229–244.

Olsen, Jr., D. R. and Nielsen, T. (2001). Laser pointer interaction. In *CHI '01: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 17–22, New York, NY, USA. ACM Press.

Olsen, Jr., D. R., Nielsen, S. T., and Parslow, D. (2001). Join and Capture: A model for nomadic interaction. In *UIST '01: Proceedings of the 14th annual ACM Symposium on User Interface Software and Technology*, pages 131–140. ACM Press.

Oviatt, S., Cohen, P., Wu, L., Vergo, J., Duncan, L., Suhm, B., Bers, J., Holzman, T., Winograd, T., Landay, J., Larson, J., and Ferro, D. (2000). Designing the user interface for multimodal speech and pen-based gesture applications: State-of-the-art systems and future research directions. *Human-Computer Interaction*, **15**(4), 263–322.

Partridge, K., Chatterjee, S., Sazawal, V., Borriello, G., and Want, R. (2002). TiltType: accelerometer-supported text entry for very small devices. In *UIST '02: Proceedings of the 15th annual ACM Symposium on User Interface Software and Technology*, pages 201–204, New York, NY, USA. ACM Press.

Patel, S. N. and Abowd, G. D. (2003). A 2-way laser-assisted selection scheme for handhelds in a physical environment. In *Ubicomp '03: Proc. of the 5th Int. Conf. on Ubiquitous Computing*, Lecture Notes in Computer Science, pages 200–207. Springer.

Patel, S. N., Kientz, J. A., Hayes, G. R., Bhat, S., and Abowd, G. D. (2006). Farther Than You May Think: An Empirical Investigation of the Proximity of Users to Their Mobile Phones. In *Ubicomp '06: Proceedings of the 8th International Conference on Ubiquitous Computing*, pages 123–140. Springer.

Pering, T., Ballagas, R., and Want, R. (2005). Spontaneous marriages of mobile devices and interactive spaces. *Commun. ACM*, **48**(9), 53–59.

Pietzuch, P. R., Shand, B., and Bacon, J. (2003). A Framework for Event Composition in Distributed Systems. In *Middleware '03: Proceedings of the 4th ACM/IFIP/USENIX Int. Conf. on Middleware*. Springer-Verlag.

Plamondon, R. and Srihari, S. N. (2000). Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(1), 63–84.

Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., and Winograd, T. (2001). ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Ubicomp '01: Proc. of the 3rd Int. Conf. on Ubiquitous Computing*, pages 56–75. Springer-Verlag.

Raskar, R., Beardsley, P., van Baar, J., Wang, Y., Dietz, P., Lee, J., Leigh, D., and Willwacher, T. (2004). RFIG lamps: interacting with a self-describing world via photosensing wireless tags and projectors. *ACM Trans. Graph.*, **23**(3), 406–415.

Ravindran, S., Smith, P., Graham, D., Duangudom, V., Anderson, D., and Hasler, P. (2005). Towards Low-Power on-Chip Auditory Processing. *EURASIP Journal on Applied Signal Processing*, **7**, 1082–1092.

Rodden, T. and Benford, S. (2003). The evolution of buildings and implications for the design of ubiquitous domestic environments. In *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 9–16. ACM Press.

Rohs, M. (2005a). Real-world interaction with camera phones. In *UCS '04: Proceedings of the 2nd International Symposium on Ubiquitous Computing Systems*, pages 74–89. Springer.

Rohs, M. (2005b). Visual code widgets for marker-based interaction. In *IWSAWC'05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems – Workshops (ICDCS 2005 Workshops)*, pages 506–513, Columbus, Ohio, USA.

Rudström, A., Cöster, R., Höök, K., and Svensson, M. (2003). Paper prototyping a social mobile service. In *MUM '03: Proceedings of the 2nd International Conference on Mobile and Ubiquitous Multimedia – Workshop on Designing for Ubicomp in the Wild: Methods for Exploring the Design of Mobile and Ubiquitous Services*. ACM.

Samberg, J., Fox, A., and Stone, M. (2002). iClub, An Interactive Dance Club. In *Ubicomp '02: Adjunct Proceedings of the 4th Int. Conf. on Ubiquitous Computing (Videos)*. Springer.

Schmidt, A., Aidoo, K. A., Takaluoma, A., Tuomela, U., Laerhoven, K. V., and de Velde, W. V. (1999). Advanced interaction in context. In *HUC '99: Proceedings of the 1st international Symposium on Handheld and Ubiquitous Computing*, pages 89–101, London, UK. Springer-Verlag.

Schmidt, R. A., Zelaznik, H. N., Hawkins, B., Frank, J. S., and Quinn, J. T. (1979). Motor-output variability: A theory for the accuracy of rapid motor acts. *Psychological Review*, **86**, 415–451.

Schön, D. and Bennett, J. (1996). Reflective Conversation with Materials. In T. Winograd, editor, *Bringing Design to Software*, pages 171–189. Addison-Wesley.

Schrage, M. (1999). *Serious Play: how the world's best companies simulate to innovate*. Harvard Business School Press, Boston, MA, USA.

Sheth, A. (1999). Changing Focus on Interoperability in Information Systems: from system, syntax, structure, to semantics. In *Interoperating Geographic Information Systems*, chapter 2, pages 5–30. Kluwer, Norwell, MA, USA.

Shneiderman, B. (1992). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Silfverberg, M., MacKenzie, I. S., and Kauppinen, T. (2001). An isometric joystick as a pointing device for handheld information terminals. In *GI '01: Proceedings of Graphics Interface 2001*, pages 119–126. Canadian Information Processing Society.

Snyder, C. (2003). *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Morgan Kaufmann, San Francisco, CA, USA.

Sotamaa, O. (2002). All the World's a Botfighter Stage: Notes on Location-Based Multi-User Gaming. In *Proceedings of the Computer Games and Digital Cultures Conference*, pages 35–45. Tampere University Press.

Sturman, D., Banavar, G., and Strom, R. (1998). Reflection in the gryphon message brokering system. In *OOPSLA '98: In Workshop Proceedings of the 13th ACM Conference on Object Oriented Programming Systems, Languages and Applications – Reflection Workshop*. ACM.

Taylor, R. N., Medvidovic, N., Anderson, K. M., Whitehead, Jr., E. J., Robbins, J. E., Nies, K. A., Oreizy, P., and Dubrow, D. L. (1996). A component- and message-based architectural style for gui software. *IEEE Trans. Softw. Eng.*, **22**(6), 390–406.

Thomas, S. (2006). Pervasive learning games: Explorations of hybrid educational gamescapes. *Simulation & Gaming*, **37**(1), 41.

Toye, E., Sharp, R., Madhavapeddy, A., Scott, D., Upton, E., and Blackwell, A. (2007). Interacting with mobile services: an evaluation of camera-phones and visual tags. *Personal and Ubiquitous Computing*, **11**(2), 97–106.

Truong, K., Huang, E., and Abowd, G. (2004). CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In *Ubicomp '04: Proceedings of the 6th International Conference on Ubiquitous Computing*. Springer-Verlag.

Ulrich, K. and Eppinger, S. (1995). *Product design and development*. McGraw-Hill, New York, NY, USA.

Villar, N., Gilleade, K., Raymundy-Ellis, D., and Gellersen., H. (2006). The voodooio gaming kit: a real-time adaptable gaming controller. In *ACE '06: Advances in Computer Entertainment*, New York, NY, USA. ACM Press.

Waldo, J. (2000). Alive and well: Jini technology today. *Computer*, **33**, 107–109.

Walz, S. P. (2005). Constituents of Hybrid Reality: Cultural Anthropological Elaborations and a Serious Game Design Experiment merging Mobility, Media, and Computing. In G. M. Buurman, editor, *Total Interaction. Theory and Practice of a New Paradigm for the Design Disciplines*, pages 122–141. Birkhäuser.

Wang, J., Zhai, S., and Canny, J. (2006). Camera phone based motion sensing: interaction techniques, applications and performance study. In *UIST '06: Proceedings of the 19th annual ACM Symposium on User Interface Software and Technology*, pages 101–110, New York, NY, USA. ACM Press.

Want, R., Fishkin, K. P., Gujar, A., and Harrison, B. L. (1999). Bridging Physical and Virtual Worlds with Electronic Tags. In *CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 370–377. ACM Press.

Want, R., Pering, T., Danneels, G., Kumar, M., Sundar, M., and Light, J. (2002). The Personal Server: Changing the Way We Think about Ubiquitous Computing. In *Ubicomp '01: Proceedings of the 3rd Int. Conf. on Ubiquitous Computing*, pages 194–209. Springer-Verlag.

Ware, C. and Balakrishnan, R. (1994). Reaching for objects in VR displays: lag and frame rate. *ACM Trans. Comput.-Hum. Interact.*, **1**(4), 331–356.

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, **265**, 94–104.

Wigdor, D. and Balakrishnan, R. (2003). TiltText: using tilt for text input to mobile phones. In *UIST '03: Proceedings of the 16th annual ACM Symposium on User Interface Software and Technology*, pages 81–90, New York, NY, USA. ACM Press.

Wigdor, D. and Balakrishnan, R. (2004). A comparison of consecutive and concurrent input text entry techniques for mobile phones. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 81–88, New York, NY, USA. ACM Press.

Worden, A., Walker, N., Bharat, K., and Hudson, S. (1997). Making computers easier for older adults to use: area cursors and sticky icons. In *CHI '97: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 266–271, New York, NY, USA. ACM Press.

Zwicky, F. (1967). The morphological approach to discovery, invention, research and construction. In *New Methods of Thought and Procedure*, pages 273–297. Springer, Berlin.

Index