Auto-generation of Individualized, Accessible User Interfaces from a Functionality-Input-Needs / User-Sensible-Input Model

By

J. Bern Jordan

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Biomedical Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2015

Date of final oral examination: November 30, 2015

The dissertation is approved by the following members of the Final Oral Committee:

Gregg Charles Vanderheiden, Professor, Biomedical Engineering
Robert G. Radwin, Professor, Biomedical Engineering
Justin Williams, Professor, Biomedical Engineering
Douglas A. Wiegmann, Associate Professor, Industrial & Systems Engineering
Bilge D. Mutlu, Associate Professor, Computer Science

Acknowledgements

The experience of getting a Ph.D. has been a challenging, yet rewarding experience for me. I would particularly like to thank my advisor, Gregg Vanderheiden, for his support, patience, guidance, and advice during this entire process. Without these, this dissertation and my work on other projects would not be what they are. Thank you for both pushing me and encouraging me explore.

I would also like to thank Brian, Britta, and Alec for their assistance with the experimental data collection. While I might have liked to have had more interactions with participants myself, your assistance made my experiment a stronger and better one.

My family, my family of origin, and my small group have all been very supportive of me during my studies. Thank you, everyone, for how you have helped me while in graduate school. In particular, thank you, Kelly, for your patience and support and your willingness to be flexible for the times when I was working at capacity. We make a good team.

Much of the work in this dissertation was carried out with funding from the National Institute on Disability Independent Living and Rehabilitation Research, U.S. Department of Education, grant number H133E080022 (RERC on Universal Interface and Information Technology Access). However, the contents do not necessarily represent the policy nor imply endorsement by the funding agencies.

Table of Contents

List of Figures

List of Tables

List of Appendices

List of Abbreviations

Abbreviation	Expansion
AUI	Abstract User Interface
API	Application Program Interface
AT	Assistive Technology
FIN	Functionality Input Need
Gen-*	Automatically-Generated Interface
GUI	Graphical User Interface
HTML	HyperText Markup Language
ICT	Information and Communication Technology
Mfr-*	Manufacturer-Created Interface
PAUI	Pluggable Alternative User Interface
PUC	Personal Universal Controller (project)
PDA	Personal Digital Assistant
UI	User Interface
USI	User-Sensible Input
WAI-ARIA	Web Accessibility Initiative's Accessible Rich Internet Applications
WCAG	Web Content Accessibility Guidelines
XML	eXtensible Markup Language

Abstract

People with disabilities often have difficulty using mainstream user interfaces because of a poor fit with their individual needs and constraints. The automatic generation of personalized user interfaces is a potential solution to these problems. In this dissertation, a two-sided, Functionality Input Needs and User Sensible Input (FIN-USI) model is described to make this practical. The Functionality Input Needs (FIN) side of the model describes what a system needs as input from the user and the manner in which it is needed. The User-Sensible Input (USI) side of the model describes the components of a basic, yet extensible abstract user interface model that can fulfill the FINs. The USI model was developed to be simple enough to be used both to model the interfaces for devices and for the automatic generation of user interfaces, while also covering more types of input than current models. In order to validate the model, three interface generators were created. These generators took USI-specified models of target devices and used them to automatically generate different self-voicing, mobile-device interfaces intended for people who are blind. These interface generators created interfaces using each person's preferred interactors for particular input tasks. One of the interface generators created novel interfaces that could be accessed by dragging a finger in a loop around the edge of a screen. The efficacy of the model and generators was then tested using 12 blind and 12 inexperienced, blindfolded participants. In the main user study, participants' performance, preference, and satisfaction were measured and compared on four interfaces: two interfaces that were manufacturer created and two that were automatically generated from each user's preferences. All usability measures were better significantly for the automatically-generated interfaces compared to the manufacturer-created ones, including a manufacturer-created interface specifically designed for people who are blind. Further case studies were done with three participants to compare additional auto-generated interfaces that were intended to be even simpler for them to use.

Chapter 1.
Introduction

Information and communication technology (ICT) and devices, software, and systems with ICT-like interfaces are widely and increasingly being used in the workplace, at home, and in the community. Most users interact with a device through the built-in interface provided by the manufacturer. Some devices may also have remote controls or applications available that can be used for control. These mainstream products and their user interfaces (UIs) are designed with particular types or groups of users in mind.

People with disabilities are often not considered in the design of these user interfaces, or the interfaces only account for some types, degrees, or combinations of disability. As a result, users with disabilities often encounter interfaces that present a variety of barriers to use. For example, small text on a screen may make a smartphone difficult or impossible for people with low vision to use. Controls on a clothes dryer may be unreachable for people who are in wheelchairs and have limited reach. The touchscreen on a typical information kiosk would be unusable by a person without vision. A remote control might be unusable because the buttons are small and hard to see for elders. Many of these interfaces are also too complicated for users with cognitive disabilities, cognitive decline, or people with short-term memory problems who find it difficult to learn new interfaces. Since devices with electronic interfaces are being integrated into all parts of life (education, transportation, employment, home health, daily living, etc.), not being able to use these interfaces and therefore these devices is rapidly creating a severe barrier to participation and living independently. However, these people might be able to use the same products if they had different, more accessible user interfaces that fit their individual needs and constraints.

There are many potential strategies for creating accessible interfaces for people with disabilities. Built-in interfaces may be designed using a universal/inclusive design process to try to accommodate as many users as is commercially feasible (Mace, 1998; Vanderheiden, 2000). Software and web UIs may be designed according to accessibility standards, such as the Web Content Accessibility Guidelines (WCAG) 2.0 (Caldwell et al., 2008), and application program interfaces (APIs), such as MSAA, UIA, IAccessbile2, AT-SPI, UIAExpress, NSAccessibility, UIAccessibility, and others (Gibson, 2007; Gonzalez & Reid, 2005; Watson & McCathie Nevile, 2015). If UIs are designed according to accessibility standards and provide accessibility through APIs, then there is a better chance that assistive technology (AT), such as screen readers, can interpret and present the interface in a user-sensible manner (e.g., in audio form to a person who is blind). Manufacturers may also create specialized, niche products with built-in interfaces geared towards particular types of users. A manufacturer may go so far as to create several alternative UIs for a product for different expected user populations. All of these strategies, if they are to well-address the needs of people with disabilities, require deep knowledge of accessible design and the resources necessary for development and testing with all of the different types, degrees, and combinations of the disabilities targeted.

Furthermore, all of these accessibility strategies focus on **groups** of people with one or more general disabilities. They are not personalized. This makes accessibility more difficult to extend to people who have needs and preferences that are uncommon or conflict with those of other users, even other users with the same general disability. As a result, products today that work well for people with a wide range of disabilities are rare, and even exceptionally accessible products leave out people on the fringes of the disability spectrum.

1.1. Pluggable Alternative User Interfaces

A potential future strategy for accessible UIs would be for products to allow the connection and use of pluggable alternative user interfaces (PAUIs). Products that support PAUIs would allow a personal device to be used that has the ability to automatically generate a UI tailored to an individual's needs and preferences without human intervention. With this approach, a person would use their PAUI device and connect to a device's interface socket, which is an abstract interface model that would specify all of the information necessary for the PAUI to automatically generate an interface (Vanderheiden, 2009; Zimmermann & Vanderheiden, 2005). A PAUI device could potentially take many forms: a smartphone with a screen reader, a tablet, a speech recognition system, a braille computer, a laptop computer with specialized input hardware, and other computing devices. From any one of these PAUIs with a user interface generator, a user could potentially control a wide variety of compatible targets that may include audio-visual equipment, home automation and environmental controls, appliances, office equipment, and more. If no manually-designed UIs fit the PAUI and the user's needs, a PAUI could automatically generate a suitable UI for the user.

There are three groups that need to coordinate efforts in order for PAUIs with automatic user interface generators to move from concept to reality.

- **Manufacturers** need to model the functionality of devices and allow PAUIs to connect.
- **Interface generator creators**, who might be developing PAUIs or AT, need to create interface generators for PAUIs that can use the model.
- **Users** need to have a PAUI device that they (and also potentially clinicians, family members, and others who might support them) have provided with the needs and preferences information necessary for creating interfaces that they can use.

1.1.1 User Interface Modeling

A robust user interface model must be at the foundation of the PAUI effort. A useful model needs to be simple, so it is easy to use by both manufacturers (to model the functionality and user interfaces of their products) and interface generator creators (so they can develop interface generators that cover all of the functionality that might be modeled). The model also needs to be broad so that it can be used to model and generate interfaces for the wide variety of functionalities and applications that users encounter.

Various models for the generation of UIs have been developed as standards or as the subject of research projects. Many of the research-oriented models suffer from being too complex for their potential benefit (Meixner, Paternò, & Vanderdonckt, 2011; Myers, Hudson, & Pausch, 2000). None have been used by industry in any meaningful way that would enable accessibility. Models that have been successfully tested with users, such as the Personal Universal Controller (J. Nichols, Chau, & Myers, 2007) and SUPPLE (Gajos, Wobbrock, & Weld, 2008) have tended to be simpler. Unfortunately, models that have been developed to this point have generally been focused on relatively limited types of interaction and input, such as input that can be handled by form-like interfaces. Many of these systems seem to have been created so that standard form-based graphical UI widgets could be used, rather than considering a broader range of what various systems might need as input.

1.1.2 Interface Generation

Having an interface model of target devices is only one step towards having an accessible PAUI. Interface generators take the model of the target interface (and potentially other inputs) and build the final interfaces that people can use.

In order to be most useful to people with disabilities who cannot use mainstream interfaces to control all of the devices and services they encounter, interfaces may be automatically generated to fit users' needs and preferences. The generation of interfaces should be fully automatic (after any required

personalization or initialization that an interface generator requires). A user should not have to wait for another person to be available to customize or finalize a user interface before use.

To date, the vast majority of UI generation projects have focused on graphical user interfaces (GUIs). Graphical UIs are very powerful, but they exclude people who cannot see well enough to read or operate the interfaces. There have been relatively few UI generation projects that have focused on users with vision impairment.

Past projects that have generated interfaces that could be used by people who are blind or who have low vision have had shortcomings that limit their potential real-world usage. Olsen, Jefferies, Nielsen, Moyes, and Fredrickson (2000) and J. Nichols et al. (2002) explored auto-generated speech-input interfaces. In order to improve recognition, both systems used constrained language recognition to which users must be trained. This may be more difficult for people to use and understand compared to natural language that people use when communicating with others. Speech-input interfaces have additional drawbacks that make them unsuitable as the only interface for non-visual users. Speech input does not allow for private input. Also, some people cannot speak clearly enough for systems to understand. Because speech recognition systems are not 100% accurate, other interfaces are necessary for people who cannot see. The MyUI project (Peissner, Häbe, Janssen, & Sellner, 2012) demonstrated multimodal interfaces that could provide text-to-speech to user who had difficulty seeing the interface. However, the UI pattern-based generation of these self-voicing interfaces is limited to supported applications. Patterns must be manually extended in order to cover a wider range of systems that users may wish to control.

1.2. The Contribution of this Work

As with other interface generation schemes, a PAUI would rely on abstract user interface (AUI) models that define the interface to be generated in abstract, device-independent terms. Currently available AUI models are inadequate either because they require too much effort to implement or because they do not cover all of the necessary functionality.

For my contribution, I developed and proposed a model that can potentially be used as a model for PAUIs. My new model, the FIN-USI model, is a combination model of the input that devices, software, and systems need in order to function (i.e., Functionality Input Needs; FINs) and user-sensible inputs (USIs) that can be used to fulfill all of those needs. In developing the model, I considered a wider range of systems and functionality than has been previously covered by other abstract UI models. The FIN-USI model and its development are detailed in Chapter 3.

To validate the FIN-USI model, I created three interface generators and a preference elicitation system for self-voicing interfaces that could be useful for people who are blind (Chapter 4). The interface generators were designed to be used on smartphones because of the portability and near ubiquity of smartphones, but the approach can be generalized to other types of PAUI devices. The different generated interfaces take gesture input or button input from users and provide speech feedback. The components of the generated interfaces go beyond the types of interfaces and interactions that are available today on mobile devices for users who are blind.

As part of the validation of the model and the basic interface generators, I performed testing with participants to compare manufacturer-created interfaces and the gesture-based automatically-generated interfaces (Chapter 5). In the interface testing, I tested blind participants who were familiar with screen readers and text-to-speech systems and blindfolded participants who have little or no experience and few strategies for using such self-voicing interfaces. This was done to test the efficacy of the approach on newly blinded users. In the testing, I also included a copier device with a manufacturer-created accessible interface that was specifically designed for their copier for people who are blind and using screen readers.

I also ran three case studies with additional automatically-generated interfaces that did not require touchscreen gestures (Chapter 6). The participants included in these studies had some difficult remembering or performing touchscreen gestures, and the new tested interfaces avoided these gestures.

Chapter 2.
Background & Related Work

Everybody must have a user interface that meets their needs or they will be unable to use it. Many people can adapt to a range of interfaces and modalities, but people with disabilities and many others have limitations. A person who is blind cannot see to read or use an interface that provides only visual output. Individuals with limited motor control cannot accurately use computer interfaces with only small controls. Others with cognitive disabilities may find it hard to understand an interface that has many options or buttons. To solve this problem, either methods and tools can be developed to help people adapt to each different interface, or interfaces can be adapted so they are usable by people with a wider range of abilities.

2.1. Current Strategies for Dealing with Inaccessible Interfaces

People who cannot use a product could use a variety of strategies to deal with an unusable user interface. If available, a person may simply avoid the problem and buy a different mainstream or specialized product that is usable to them. For example, blind users might buy thermostats with buttons rather than touchscreens if they are allowed to wire one into their apartment or office. A person with limited dexterity might purchase a universal remote with larger buttons if universal remotes are available for all of the products with which they have trouble. Secondly, people may develop workarounds and strategies that allow them to operate the ordinarily inaccessible product. For example, people who are blind often add sticky dots or puff paint to mark controls to make them easier to identify or locate if it is a physical control and not an on-screen button. A motivated person who is blind might memorize patterns of button pushes to get through the menus on an inaccessible device in order to use some its functionality

if they have those memory capabilities and prior familiarity with the device in front of them. As another strategy, people may use adaptive assistive technology (AT) to access a product if the product is compatible with their AT. Unfortunately, AT may add an additional layer of interface which may make a device more difficult and complicated to use than a built-in user interface as used by other people. For example, a computer screen reader is a spoken interface representation of a visual interface to a product. This layering of a spoken interface which is interpreting a visual interface to the product's functionality can be difficult to use and hard to understand.

2.2. Current Strategies for Creating Accessible Interfaces

Product manufacturers also have strategies that can be used to improve the usability of UIs across a range of persons and disabilities. They can design UIs through a universal design process, develop accessible software and web UIs, build specialized products, or provide multiple alternative UIs.

The first of these strategies for improving product usability across users is to design an interface using an approach known variously as universal design, inclusive design, or design for all (Clarkson, Coleman, Keates, & Lebbon, 2003), where the widest possible range of people and abilities are considered during the design process. With regards to commercial ICT, the universal design approach involves trying to make systems and services usable to as many people in as many situations as possible within commercial constraints (Vanderheiden, 2000). However, there are commercial, practical, and theoretical limits of the universal design approach. People with very specific needs, needs that conflict with other users, or needs that are impractical to meet in a commercial product, may still find themselves unable to use these products even if universal design is used in their development (Harper, 2007; Vanderheiden, 2009). There are always people with severe disabilities or combinations of disabilities who may only be able to use a product if it had a customized user interface: one that would be impractical and expensive to build into all products. Some universal design techniques may also fundamentally alter a

device (e.g., increasing the size of a smartphone to incorporate large buttons), which would make the device difficult to sell in the mainstream market.

A second strategy for improving the usability of a system is to create a software or web interface for the first or *target* device that can be displayed and interacted with on a second device such as a computer or smartphone. To be usable to a particular person there are three requirements for such a remote UI:

1. The software UI must be designed according to applicable accessibility standards and have an accessibility application program interface (API),
2. Any assistive technology (AT) hardware and software a user requires to use that API must be available to that person when they need to use it, and
3. The accessibility standards and APIs that are used by the software UI must be implemented well in the person's AT.

The technology of AT lags mainstream technology (Vanderheiden, 2008), so each software UI would have to be manually created using a "least common denominator" approach, using only the most widely-compatible UI features, or each UI would need to undergo expensive testing with a wide range of AT to ensure broad compatibility and correct functioning. Unfortunately for users, as noted above, AT may add an additional layer of interface which may make a device more difficult and complicated to use than a built-in user interface.

As a third strategy, manufacturers may create specialized, niche products with built-in interfaces geared towards particular types of users. Specialized products, such as simplified remote controls, large button calculators, vibrating alarm clocks, and talking timers, are generally niche products that do not have a large market. As such, specialized accessible versions of products are almost always more expensive (usually much more expensive) than their mainstream counterparts, or they may simply not be

used because the individual feels they are stigmatizing (Bichard, Coleman, & Langdon, 2007). Because of commercial unviability, they also would not exist for many products.

Finally, manufacturers may go so far as to create multiple, alternative UIs that are specifically designed for different expected user groups. This strategy can be very useful to people who fall in the particular groups targeted by the manufacturer, but there will always be people with uncommon or particularly severe needs that cannot be addressed by any of the alternative UIs provided by a manufacturer. This strategy is also difficult to scale. Having to develop, support, and maintain multiple interface versions is challenging and costly.

These four general strategies represent the current state of making interfaces accessible to people with disabilities. Implementing accessibility in these ways can be challenging and there is a shortage of people who are trained to do so (Vanderheiden & Jordan, 2012) and an ever expanding number of products with digital interfaces appearing in people's lives. Different solutions need to be found for accessibility to be extended to the widest range of people and products.

2.3. The Interaction of User and the Interface

When a person uses a device, software, or systems to perform a task, the interaction with the system can be modeled as an interaction between four components: the user, the user interface, the system functionality, and the task as seen in Figure 2-1. In this model, the user interacts with the user interface that in turn controls the system functionality in order to complete the task. The user may get feedback from the interface or through the progress towards the task itself.

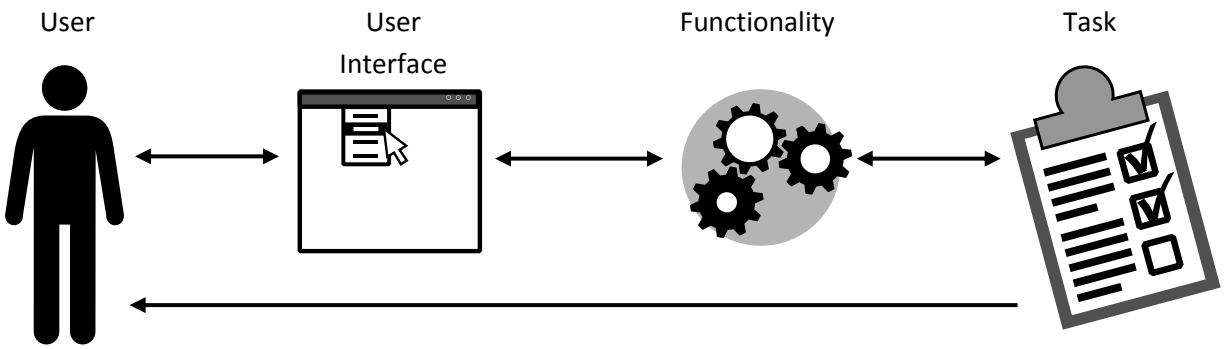


Figure 2-1. A model of interaction with ICT. In order to complete the task using ICT, the user must interact with the user interface, which controls the system functionality. The user may receive feedback about the operation from the interface or changes to the task.

A *user* has particular characteristics, skills, capabilities, and limitations that may help or hinder their usage of a particular user interface. For example, a person with color blindness would not be able tell the difference between particular colors on a color display. If color is used to convey information or differentiate controls, the person will not get this information. Another person may have poor motor control or weakness because of disability or ageing and may be unable to manipulate a particular type of user interface effectively. A person who is blind would not be able to use a purely visual interface, but would be more successful with an audio interface. Users' interface needs—what they need in order to perceive, understand, and operate an interface—must be met, or else they will not be able to effectively use the interface to complete the task.

The *user interface* is like a bridge between the user and the device functionality and is responsible for translating and relaying the commands and input from the user, and feedback from the system. User interfaces are specific to particular input and output modalities. User interfaces (even for the same function) can take on a variety of forms. For example, to move the pointer cursor on a computer one can move a mouse, roll a trackball, press arrow keys, touch a touchscreen, gesture on a graphics tablet, move a joystick, use a trackpad on a laptop computer, and so on. To access a file, one might double-click the

icon with a mouse, tap a file icon once on a touchscreen, type its filename at a prompt, or select the item from a list. For some particular devices, there are national, international, or *de facto* standards and common user interface patterns that users expect: people expect telephone keypads and computer keyboards to have characters arranged in particular ways, for example. Even with some standardization, user interfaces typically vary between devices that have the same functionality.

The system *functionality* is the output or outcome that can be done by the system for the user. The system needs particular input from the user in order to function as the user desires. The system does not care what users do or the modality they use to provide the information (e.g., if they turn dials, press buttons, type on a keyboard, or use a touchscreen). The system simply needs specific information to function as the user intends.

While not often apparent to the user, there is typically a separation between the interface and the system functionality. The separation between the interface and system functionality seen in the interaction model above is a common part of software architecture patterns. The Seeheim model (Pfaff, 1985) was an early generic software architecture that separated the user interface from the functionality core. This model was used as the basis for models and patterns such as the Seattle model (Lantz, Tanner, Binding, Huang, & Dwelly, 1987), the Lisbon model (Duce, Gomes, Hopgood, & Lee, 1991), the Arch model (Bass et al., 1992), Model-View-Controller (Krasner & Pope, 1988), Presentation-Abstraction-Control (Coutaz, 1987), Model-View-Adapter, Model-View-Presenter (Potel, 1996), Model-View-ViewModel (Smith, 2009), and other software architecture patterns. This separation of concerns is widely used in practice because its modular design allows for easier code reuse and developer specialization. In COUSIN, an early user interface management system, Hayes and colleagues (Hayes, Lerner, & Szekely, 1983; Hayes, Szekely, & Lerner, 1985) emphasized this separation to be able to automatically generate a UI from an abstract definition of the functionality. They found a number of advantages to such an approach: reduced effort for UI design, UIs with more user support, easier involvement of human factors

experts, more consistent UIs, and the ability to provide multiple UIs (Hayes et al., 1985). This separation between the interface and functionality also potentially allows for alternative user interfaces to communicate with the underlying system functionality.

2.4. Remote Control: A Promising Accessibility Strategy

Ubiquitous computing was a paradigm first developed and explored at the Xerox Palo Alto Research Center in the late 1980s (Weiser, 1991, 1993; Weiser, Gold, & Brown, 1999). In *ubiquitous computing* (also known as *pervasive computing*, *ambient intelligence*, or *everyware*) it is envisioned that users will be surrounded by devices with computing power, and that users would interact simultaneously with many devices and systems during everyday activities. There are many research centers around the world researching aspects of ubiquitous computing. One thread of active research related to ubiquitous computing involves the ability to access and operate a variety of target devices using mobile devices.

A number of systems that allow for remote control of products and environments have been developed and demonstrated. Hodes and colleagues outlined a system where users could connect to services around them with their mobile devices (Hodes, Katz, Servan-Schreiber, & Rowe, 1997). They prototyped their wireless system in a high-tech seminar room and implemented controls for audio/video equipment, lights, and printers. Several user interfaces for the system were custom programmed and could be used from different thin clients (Hodes et al., 1997). While the Hodes et al. system demonstrated remote control of a variety of devices, the system did not have the ability to automatically generate a user interface for new or unknown devices. Researchers at IBM (Eustice, Lehman, Morales, Munson, & Guillen, 1999) envisioned a universal information appliance that could serve as a user's personal assistant and portal to control technology around them. In their initial implementation they developed a networking system to communicate between devices and a universal information appliance on a personal digital assistant (PDA) that used a language called MoDAL (Eustice et al., 1999). MoDAL specified much of the

look and feel of the graphical user interfaces on PDAs, so there was little flexibility for the system to create alternate user interfaces. XWeb (Olsen et al., 2000) was a project at Brigham Young University that envisioned expanding the Internet to home appliances and other devices so they could be remotely controlled from any available web client or browser. They prototyped a method of Internet communication between devices and developed desktop computer, wall-sized interactive display, and speech interfaces for control (Olsen et al., 2000). ICrafter was a project at Stanford University for interactive workspaces, such as conference or class rooms; it interconnected devices like projectors, computers, document cameras, and conference phones and allowed them to be controlled from the user's own computer (Ponnekanti, Lee, Fox, Hanrahan, & Winograd, 2001). The Pebbles project at Carnegie Mellon University was a wide-ranging research program to research personal digital assistant (PDA) user interfaces and their interaction with other electronic devices. One of the projects to come out of the overarching Pebbles project was the Personal Universal Controller (J. Nichols et al., 2002). Nichols and colleagues (J. Nichols et al., 2002, 2003) have demonstrated graphical and speech interfaces on PDAs used to control audio/visual equipment, telephones, and video cameras using simulated devices on computers and actual devices using adapters. While not currently able to control any devices "out of the box" without adaptation, their work has shown that PDAs may be used successfully to control some types of devices and appliances. The FReCon, or "Fluid Remote Controller" is another universal controller prototype (Sanguinetti, Haga, Funakoshi, Yoshida, & Matsumoto, 2003). In their system, the researchers at Doshisha University demonstrated PDA controllers interacting with a television simulated on a computer (for ease of programming). The FReCon system required a user to first point the controller at the device to control before being able to control the device from anywhere within the room with a Bluetooth connection and control through web-based interfaces.

The idea of using one device to control another for accessibility began earlier than ubiquitous computing and research around separable user interfaces. In the late 1970s and early 1980s, keyboard

emulating interfaces and dual computer configurations were developed to allow people with disabilities to control computers with their own adaptive assistive technology (Rodgers, Kelso, & Vanderheiden, 1982; Vanderheiden, 1981; Vanderheiden & Kelso, 1982). Mouse-emulating keyboard techniques (C. Lee & Vanderheiden, 1987) and the SerialKeys connection for assistive technology (C. Lee, Novak, Schauer, & Vanderheiden, 1990) were developed later and built directly into Windows 95 and its successors. The application of this concept was extended with the development of the Total Access System (Scott & Gingras, 2001). The Total Access System used a special "accessor" interface which emulates keyboard and mouse functions and accessors were developed to provide both haptic and American Sign Language interfaces to a visual interface. A similar approach to accessibility was the universal remote console communication protocol (Vanderheiden, 1998). This approach was initially based on infrared communication, but later evolved to be independent of specific transport formats. Work on the protocol was combined with the accessor interface concept in the formation of the V2 standards working group of INCITS (International Committee of Information Technology Standards) and eventually led to the publication of the ISO/IEC Universal Remote Console (URC) standards, now in their second edition (ISO/IEC, 2014a – 2014f). The URC standards define a system that can discover and remotely control compatible target devices through interface sockets (Zimmermann & Vanderheiden, 2005). The URC standards do not prescribe a particular user interface or method of generating user interfaces for target devices. User interfaces may be generated on the fly or they may be defined for particular types of devices through hardcoded, rigid interfaces or more flexibly through presentation templates (ISO/IEC, 2014e; Zimmermann, Jordan, Thakur, & Gohil, 2013). These presentation templates may be provided by the target device's manufacturer or by third parties. Universal remote console work has been demonstrated for simulated audio/visual equipment (Zimmermann, Vanderheiden, & Gilman, 2002) and for medical devices including a simulated vital signs monitor, exercise ergo meter, and patient examination table (Danturthi, Shroff, & Winters, 2006; Shroff, 2005).

Today, the remote control of connectable devices is a fast-growing commercial market (Icontrol Networks, 2015). There are many home automation products available directly to consumers for security, lighting, environmental control, audiovisual equipment, and other devices. Many of these systems have their own dedicated applications that users can download to their mobile devices. Smart home software frameworks, such as HomeKit, Brillo, Weave, Smart Home Service, and others, are now being developed and utilized to potentially bring together disparate devices and systems and allow more integration on the user side. This connectedness has the potential to allow people with disabilities to use devices that they had not been able to use before. However, currently the interfaces used to control these smart devices are limited in their ability to accommodate people with widely varying disabilities.

2.5. Model-based Generation of Interfaces

Users need alternative UIs when the standard UI does not meet their needs. Alternative UIs can be handcrafted, but this is not a scalable solution because of the wide range of user needs and disabilities and the number of individuals needing alternative UIs. Through automatic, model-based UI generation, alternative UIs could be created and tuned for each person in an economic and scalable fashion.

2.5.1 Models

The CAMELEON reference framework classifies four different conceptual levels of a model-based UI generation process (Calvary et al., 2002, 2003):

- *Concept & Task Models* define the tasks a user is expected to perform and the domain objects with which a person will interact.
- *Abstract UI Models* define interface elements independently of specific interaction modalities.

- *Concrete UI Models* specify the interactors or widgets that will be used, which are targeted to a particular platform, environment, and modality. The concrete UI is not yet runnable and usable.
- The *Final UI* is the runnable user interface that can be executed as binary code or an interpreted language.

A user interface generator may take models at any of these levels and may incorporate other models and information about the user, the context of use, and the environment in order to generate the final UI. Fully automatic generation of UIs is challenging (Myers et al., 2000). Because of this, many of the model-based interface generation research programs have limited their work to semi-automatic generation of user interfaces (Meixner et al., 2011). This hand-tuning of interfaces is unacceptable as a scalable solution for providing access to people with disabilities. Most model-based user interface projects have focused only on mainstream users, leaving the creation of interface generators for people with disabilities as future work, with two notable exceptions. In the SUPPLE project, a constraint-based approach was used to lay out graphical user interfaces (GUIs) for people with physical disabilities (Gajos et al., 2008). In the MyUI project, a pattern-based approach was used to create UIs for older people interacting with specific applications on a television (Peissner et al., 2012).

Early model-based systems were developed to automatically generate interfaces because, at the time, creation of UIs was a particularly difficult process. Modern toolkits and visual editors were not yet available. With the Mickey system (Olsen, 1989), basic menus and dialog boxes were generated from specialized function signatures and comments in the application logic code. The Jade system (Vander Zanden & Myers, 1990) utilized a basic textual specification of seven types of inputs.

As frameworks were further developed, they generally became more complex. The MASTERMIND system (Szekely, Sukaviriya, Castells, Muthukumarasamy, & Salcher, 1996) was one of the first to include a task model, which was modeled in terms of goals and preconditions. Since then, most

model-based UI generation projects have included a task model or other higher-level model of the interface. The TRIDENT system (Bodart et al., 1995; Vanderdonckt & Bodart, 1993) relies on task models in the form of an Activity Chaining Graph. ConcurTaskTrees (Paternò, 2000) is a hierarchical task model that is included as a component of the TERESA (Mori, Paternò, & Santoro, 2004) and MARIA (Paternò, Santoro, & Spano, 2009) projects. The UsiXML project uses FlowiXML (Guerrero Garcìa, Vanderdonckt, & Gonzàlez Calleros, 2008) as a task model, which is itself an extension of ConcurTaskTrees. The task models are helpful in improving an interface by providing information that can be used in grouping and navigation (Montero & López-Jaquero, 2007). Producing task models also requires more effort, which may not be worth the investment for manufacturers. This complexity of models and interface generators is a disadvantage to the people who most need automatically generated interfaces: people with disabilities and specific needs that are not accommodated with mainstream interfaces.

There are some notable exceptions to the general trend towards increasing model complexity. The SUPPLE system represents abstract UIs with a combination of primitive input types and container types (Gajos, Weld, & Wobbrock, 2010). However, the primary focus of the SUPPLE project was on making graphical UI generation more flexible rather than specifically on the model behind UI generation. The Personal Universal Controller system (J. Nichols et al., 2002) defines interfaces in a similar way, with a limited set of input elements and a tree interface structure that can be adjusted based on dependency information. Both of these systems were tested with users and showed performance and other benefits over manufacturer and other one-size-fits-all UIs (Gajos et al., 2008; J. Nichols et al., 2007).

A widely used type of model at the Concrete and Final UI levels are GUI toolkits. Many GUI toolkits are available for different platforms and frameworks. Toolkit widgets may support input from multiple modalities (e.g., keyboard, mouse, and touch input), which can make them easier to use by people with disabilities. Toolkits for GUIs may also have built-in support for accessibility APIs. Assistive

technology, such as screen readers, can use these APIs in their functioning to present an alternative UI to users. Without accessibility APIs, using assistive technology can be more difficult or impossible. There are strategies, such as building and maintaining off-screen models that can be used (Schwerdtfeger, 1991), but these are more difficult and may only be available for the most commonly used applications. The strategy of basing an alternative interface on a graphical interface may result in layers of interface; for example, a person with a screen reader listens to and interacts with an audio representation of a graphical interface in order to control a system's functionality. This layering and indirection can lead to unnecessary complexity and confusion.

A number of interface models at different levels have been standardized. The HyperText Markup Language (HTML) is ubiquitous on the web and evolved from a language initially for the publication and exchange of documents to one that supports complex applications. Markup in HTML5 (Hickson et al., 2014) can be extended with scripts to enhance the type of applications and interactions that can be created, but this can lead to inaccessible interfaces if care is not taken. XForms (Boyer, 2009) offers a potentially more flexible way of specifying forms with richer processing logic than HTML, but is used much less in practice. In order to better support assistive technology in dynamic HTML, WAI-ARIA (Craig, Cooper, Pappas, Schwerdtfeger, & Seeman, 2014) was developed and can be used to provide compatible assistive technology with enhanced metadata about elements' roles and states. The URC standards define an interface socket that is defined in an XML language with a resource layer that defines different groupings and dependencies and uses XML Schema (Biron & Halhotra, 2004) data types. Although not a system for user interface generation, the Common Access Profile (ISO/IEC, 2009) uses a two-sided approach for assessing the matching quality of a system's capabilities with a user's needs. However, this approach is focused on modality-matching and does not accommodate individual user interface elements.

2.5.2 Interface Generation

In order to be effective universally, a PAUI device must have a user interface available for every product to be controlled. User interfaces may be manually designed or automatically generated from the data that the product communicates with the PAUI. While manually-designed user interfaces may be more effective for many users, it is impractical to design all user interfaces needed for every combination of user, PAUI, and target product. Because of this impracticality, researchers are working on the systems and algorithms to automatically generate appropriate user interfaces.

Many researchers have devised systems and algorithms to generate flexible graphical user interfaces (GUIs). As an early example, SYNGRAPH (Olsen & Dempsey, 1983) would automatically generate a Pascal program that would automatically generate on-screen menus and other user interface elements using text and basic formatting. Jade was another early interactive computer tool that generated GUI dialog boxes which could then be changed and tuned by a graphic artist (Vander Zanden & Myers, 1990).

With the Pebbles project at Carnegie Mellon University, methods were further developed to generate an appropriate GUI on a PDA that could be used to remotely control one or more devices (J. Nichols et al., 2002; J. Nichols, Rothrock, Chau, & Myers, 2006). The personal universal controller (PUC) could create a GUI by looking at the device's group tree (for grouping similar types of controls) and finding dependencies to develop panels of controls. The controller had a decision tree to decide in what panels controls should be placed and how much room each control takes up (J. Nichols et al., 2002). In a similar manner, a system called Huddle (J. Nichols, Rothrock, et al., 2006) could create an aggregate user interface for a system of interconnected devices (such as a DVD player which is hooked up to a television and an audio receiver which is in turn hooked up to surround sound speakers). Huddle required a flow-based model to be programmed (so the system could know the connections between devices) before it could create an aggregate user interface and separate user interfaces for each interconnected

product (J. Nichols, Rothrock, et al., 2006). User testing of the PUC system showed a reduction of errors and an increase in user speed, especially for interfaces that were generated to match an interface on which a person was previously trained (J. Nichols et al., 2007). The Pebbles GUI generation system did not take into account users' preferences, characteristics, or abilities when generating a user interface and may generate suboptimal GUIs for people with movement impairments and low vision.

Research into the SUPPLE system at the University of Washington extended the flexibility of GUI generation by automatically scaling user interfaces to a wide variety of screen sizes (Gajos & Weld, 2004). In SUPPLE, both the controller device and the user must be modeled to determine a cost function for the physical navigation movement between control widgets. Controls and their spacing are determined by optimizing the cost function given the constraints of the device (such as screen size and resolution). The SUPPLE system was extended to generate GUIs for people with low vision and movement disabilities (Gajos, Wobbrock, & Weld, 2007). In extending the model, SUPPLE++ had a module which measured and modeled users' pointing performance to optimize expected movement time. User testing with people with physical disabilities showed that participants generally performed better with the SUPPLE++ system than with other GUIs (Gajos et al., 2008).

User interface design patterns are common solutions to UI problems in particular contexts. They can be applied at different levels of an interface (Märtin, Herdin, & Engel, 2013): for example some patterns apply to the structure and layout of an entire application while other patterns may be a single widget or group of widgets that are a convenient way for entering particular types of data. Besides being convenient for designers who are manually creating interfaces, users benefit from common interaction patterns. Users become familiar with patterns as they use interfaces. This improves usability of interfaces. Because patterns are used in different applications and domains, users become accustomed to such patterns and their layout and behavior. While there is much current work on the development of standardized UI pattern languages and frameworks in support of UI generation (e.g., Deng, Kemp, &

Todd, 2006; Engel & Märtin, 2009; Wendler, Ammon, Kikova, & Philippow, 2012), the actual application of UI patterns to generated interfaces has been limited so far. The PUC project incorporated Smart Templates (J. Nichols, Myers, & Litwack, 2004), which effectively are patterns that can be applied to groups of GUI widgets. With Smart Templates, the PUC system could generate interfaces that conformed to common GUI conventions, such as a telephone keypad arranged with keys in an ascending manner in a 3x4 grid and player controls for media players in a particular arrangement and order. Support for specific Smart Templates was optional, because each template added additional burden for interface generator programmers. The more recent MyUI project made extensive use of design patterns for the generation of user interfaces (Peissner et al., 2012), as mentioned above. The MyUI project aimed to create dynamic interfaces that were accessible to people with disabilities and adapted to their abilities and situational context. The patterns were stored in a publically accessible repository and could be used as building blocks to define user interfaces (Peißner, Sellner, & Janssen, 2012). The improvement of the pattern repository was envisioned as a stepwise process—as a new application (such as e-mail or instant messaging) was developed, use-case-specific patterns could be identified and added to the repository. While the MyUI project demonstrated primarily visual interfaces that could accommodate elders and some disabilities, the project did not publish any results of user testing beyond anecdotes. Furthermore, such a pattern-based approach adds to the complexity of both models and generators. By their nature, UI patterns are specific to a particular modality, context, and interaction. Existing patterns tend to be visual, so there are few patterns for non-GUI interfaces that some people with disabilities need.

Some work has also been done to create non-graphical user interfaces for remote controlled devices that may be usable by some people with disabilities. Olsen et al. (2000) briefly described a prototype speech client for their XWeb system of interactive control that uses a widget-oriented approach. With the XWeb speech client, they simply wanted to show that it was possible to perform the basic tasks of tree traversing, value editing, and getting help. They admit that this speech interface approach may be

difficult to understand even though it does work. The Pebbles project also developed a voice interface generator that uses a constrained vocabulary which they discuss briefly (J. Nichols et al., 2002). The constrained vocabulary of Carnegie Mellon University's Universal Speech Interface was used for the Pebbles project, so users must speak with a particular syntax—for example, "set channel to WDUQ" or "volume status" (Harris & Rosenfeld, 2004). Training is required to use the system effectively. Neither the XWeb nor Pebbles speech systems were tested with people with disabilities.

Chapter 3.
The Functionality Input Need–User-Sensible Input (FIN-USI) Model

People with disabilities often find it difficult or impossible to complete tasks with user interfaces (UIs) that do not meet their needs or are not operable with their abilities. Automatic, model-based generation of user interfaces is potentially a powerful strategy for creating interfaces that fit each individual's needs and preferences.

In order to automatically generate interfaces for different target devices, systems, and software, an interface generator needs models of the target interfaces that need to be generated. Research-oriented models tend to be complex with multiple components and layers that could potentially allow for interfaces that are a better fit for a particular system, its functionality, and tasks. However, this potential usability enhancement comes at a high cost both to device manufacturers and to those who would create and personalize interface generators. Manufacturers who wish to create an abstract user interface (AUI) for their product to accommodate accessibility may have more effort when doing so with a heavy-weight model—there are simply more layers and elements with which to build the AUI. With more elements in the model to handle, interface generators in turn require both more generation rules and larger sets of UI widgets and interactors. For interface generators that are widely used (e.g., a UI generator for mainstream use by people without disabilities) the usability improvements of the generated interface may be worth it. However, for UI generators that are more specialized for people with severe, combined, and uncommon disabilities, a heavy-weight model and generator is problematic. It takes significant effort to develop many interactors that may be very specialized to particular users. Similarly, developing the rules and algorithms to be used by an interface generator is challenging and is even more so when the rules have to handle a heavy-weight model.

Models that have been successfully tested with users have tended to be simpler, light-weight models. Unfortunately, models that have been developed to this point (both light- and heavy-weight ones) have generally been focused on relatively limited types of interaction and input, such as input that can be handled by form-like interfaces. Many of these systems seem to have been created so that standard form-based graphical UI widgets (such as buttons, text input boxes, and checkboxes) could be used, rather than considering a broader range of what various systems might need as input.

What is needed is a simple model for user interface generation that covers a broader range of user input that systems need for operation. An ideal AUI model would have both broad coverage of those input needs and relatively few components and layers so that it is practical to implement by manufacturers, the creators of PAUIs, those who develop and customize interface generators. On the manufacturers' side, application developers need to be able to apply the model to existing functionality in the development of abstract user interfaces. A simpler model would require less expertise and resources. On the PAUI and interface generator side, those creating or customizing interface generators (which might include software developers, engineers, clinicians, therapists, family members, or even just the individual user) need a simple model so they know the essentials that must be implemented.

In this chapter, I describe such a new model, the Functionality Input Need–User-sensible Input (FIN-USI) model. The model has two parts: one side which models what input is required by functionality and the other side which is an AUI of input that meets those requirements.

3.1. Process of Constructing the FIN-USI Model

The FIN-USI model presented here was developed in two parts. The Functionality Input Need (FIN) side of the model was developed first in order to capture and classify the types of input that target devices, software, and systems need as input from the user in order to function. The User-Sensible Input

(USI) side is an abstract user interface model with a limited vocabulary of elements. The limited vocabulary of the USIs can be used to fulfill the functionalities' input needs.

To create a simpler, yet more comprehensive model than those that currently exist, I started by examining a variety of sources and literature. By better understanding the limitations and successes of other models, their best features could be incorporated into the new model. The model was developed in an iterative process. First a potential model was proposed and then real-world interfaces were modeled. When the model could not accommodate a real-world interface, the model was revised. The development and evolution of the FIN-USI model is described in more detail in Appendix A.

3.1.1 Literature & Standards

In an early attempt to categorize input, Foley, Wallace, and Chan (1984) proposed six elemental tasks for graphical computer systems:

- Select: Identifying object/s of interest from a set of alternatives.
- Position: Specifying coordinates in two or three dimensions.
- Orient: Specifying angles or orientations around one or more axes.
- Path: Specifying a time-dependent series of positions and/or orientations.
- Quantify: Specifying a single numeric value.
- Text: Inputting symbolic data, such as a string of characters.

These elemental tasks could potentially be accomplished through a number of different interface techniques. Their list of elemental tasks is not a list of perceptual tasks that people do on a computer, but is instead a way to think about and categorize input for graphical computer systems. Buxton (1986) realized that many of the elemental tasks could be broken down into hierarchies of subtasks. For example, Position and Orientation tasks can be broken into separate Quantify tasks for each dimension and axis of

rotation. This early work was focused on graphical computer interaction and not on more abstract forms of input that could be used in different modalities.

Other early taxonomies of input were developed that characterized input devices rather than the tasks one might perform. Buxton focused on continuous input devices and characterized their properties sensed and number of dimensions supported (Buxton, 1983). This work was later extended beyond continuous input to other devices and dimensions of input (Card, Mackinlay, & Robertson, 1990).

For creating interfaces for people with disabilities, a modality-independent approach, like that of Foley et al., (1984) is more appropriate. However, it is still important to consider the other early taxonomies of input because they generically describe the sensing of user motions and input. These device input taxonomies can categorize novel and modality-dependent input that may be used to provide input that is outside the form input paradigm for which most other AUI are designed.

The World Wide Web Consortium (W3C) has put forward several standards (known in W3C parlance as "recommendations") which can be used to specify user interfaces and user input. The HyperText Markup Language (HTML), the de facto language of the web, can be used to support everything from static documents to complex, web-based applications. The current version, HTML5 (Hickson et al., 2014), defines 26 UI components that can be used to input data in forms along with other components used for providing output to a user. Scripting languages can modify and extend the behaviors defined in HTML, so web-based interfaces can have UI elements outside of those defined in HTML that are custom-built for very specific purposes. XForms (Boyer, 2009), another W3C recommendation, is an abstract, device-independent format for the user input of data that allows for richer processing logic than does HTML. XForms only has nine different types of input UI components, but these input types can potentially take a wider variety of forms depending on the data type that a user is expected to provide and the space that is allocated on a screen. The WAI-ARIA W3C recommendation (Craig et al., 2014) aims to make web applications (especially those with custom elements) more accessible. It defines a set of roles

via a taxonomy that can be applied to HTML or other markup languages. These roles can be processed by assistive technology such as screen readers to improve access to interfaces that do not use native input elements. In WAI-ARIA, there are 12 roles that can be used for user input although the behaviors and display of elements associated with each role need to be explicitly defined by the application developer.

The W3C has also standardized a list of datatypes (Peterson, Gao, Malhotra, Sperberg-McQueen, & Thompson, 2012) for use in defining XML (eXtensible Markup Language) Schemas (which in turn are used to describe and model languages in XML). There are 44 built-in primitive and derived datatypes defined in the standard. These datatypes can be extended to form custom datatypes by union (combining two or more datatypes), list (by requiring several elements of a single datatype), or by restriction. The restrictions are defined along 14 constraining facets, which apply to some of the datatypes (e.g., an integer can be constrained by minimum and maximum values). This standard was particularly useful in the development of the FIN side of the model because it listed standard primitive and derived inputs that systems need.

The Mozilla foundation developed XUL (XML User interface Language; Mozilla, n.d.), which is an XML-based language for building user interfaces from a collection of UI widgets in applications such as Thunderbird and Firefox. In 2014, XUL had 28 input elements. User interfaces defined in XUL can also be extended using JavaScript and the XML Binding Language for defining custom widget types.

The TERESA (Transformation Environment for inteRactivE Systems representAtions) project is a semi-autonomous framework for generating interfaces for different platforms (Mori et al., 2004). Underlying the system are two XML schemas: one describes tasks and the other describes abstract user interfaces. The TERESA AUI model is composed of interconnected presentations which contain and group abstract interactors (Berti, Correani, Paternò, & Santoro, 2004). There are 11 interactors in the TERESA AUI model which allow for control and the selection and editing of values. The small number of AUI interactors helps reduce complexity and is of particular interest to my model development.

However, the overall TERESA framework involves a complex process because it requires designer involvement at multiple levels of abstraction in order to create a usable interface.

MARIA (Model-based lAnguage foR Interactive Applications) is the successor of TERESA with multiple enhancements to different modules (Paternò et al., 2009). The MARIA AUI model was expanded to 16 elementary input interactors. Data and event models were also introduced to improve the connections between different abstract interactors, but designer interaction was still required.

The User Interface Markup Language (UIML) is a standard for describing user interfaces proposed by OASIS (Helms et al., 2009). An interface specified in UIML is a structure which contains parts (various interface widgets) and their presentation and behavior. UIML provides a rich basis for describing interfaces, but it does not define a set of interactors that can be used across platforms. Instead the developer defines their own abstract UI widget classes and maps them directly to widgets in specific platforms and toolkits. Simon, Jank, and Wegscheider (2004) proposed a unified vocabulary for UIML which listed 10 basic, platform-independent interactive UI element classes. The architecture of UIML is mostly focused on the concrete UI level (Trewin, Zimmermann, & Vanderheiden, 2003), so interfaces described in UIML are not abstract enough to be readily translatable to various types of interfaces that people may need.

The User Interface eXtensible Markup Language (UsiXML; Limbourg et al., 2004) is a UI description language with the express goal of supporting features from other such description languages. UsiXML is complex, with many features, and allows for the specification of many different UI models at different levels, including task, domain, context, abstract UI, and concrete UI models and their interrelationships. The AUI model defines a few core concepts and relationships. For input, the AUI model defines 10 types of value inputs and also allows the input to be further classified by various attributes (UCL, 2013). While the complexity of UsiXML allows it to describe interfaces in many ways, the complexity is a double-edged sword, making it more difficult to apply to particular systems.

3.2. Goals for the Model

There were four goals as I was developing the model.

- Goal 1: *Allow for the creation of UIs that meet the needs of users.* All users have needs and preferences when interacting with UIs. People with disabilities may have different needs from others. A user interface generator must use the model and create an interface that meets these individual needs.
- Goal 2: *Model a broad range of input that systems need for proper functioning.* Users need to be able to control functionality of a variety of systems and devices and should not be limited to systems that only have form-like input requirements.
- Goal 3: *Require as few base components as possible.* Complex models are more difficult to implement for both manufacturers and PAUI developers. With a limited vocabulary of required inputs that are required, PAUI developers do not need to create as many interactors and UI generation rules for users who have unique input needs.
- Goal 4: *Allow for extensions to improve usability.* Expressiveness is limited with a small vocabulary. Optional extensions can enable richer interface generators that support a wider range of interactors that are tuned for particular types of inputs.

The user interface needs of all users fall into three broad categories (Vanderheiden, 2009). All users, including people with disabilities, must be able to perceive all information, must understand that information, and must be able to operate all controls necessary for all functionality. Interfaces generated via a suitable interface generator on a PAUI automatically cover the user needs of perception and operation (i.e., a generator should only use interactors that use a modality that the individual can perceive and interactors that the individual can operate). The user's ability to understand the input they are to provide, and how to do so, is a function of both the interface model and generated interface.

3.3. The FIN-USI Model

A user interface is a bridge between the user and the functionality they wish to control. The FIN-USI model is an abstract model of input that is provided through a user interface. Like a bridge, the FIN-USI model has two ends. The Functionality Input Need (FIN) end models the input needs of the functionality of the device, software, or system. For example, what does the tuner part of a television or the refrigeration functionality of a refrigerator need as user input in its most basic form? The FINs should not be conflated with what a user interface needs a user to provide. A user needs to be able to find and push the Channel Up and Down buttons on a specific remote control to tune the television, but the TV only needs to know the channel number (which is the FIN). With refrigerators, the user needs to find and twist a knob on the inside of one, or push buttons on another to set the temperature, but both refrigerators only need to know the settings and do not care how it is entered. If users cannot fulfill the FIN using an available user interface (because the available UI does not fit their needs), and no alternate interface is available to them, then they will be unable to use that functionality.

The User-Sensible Input (USI) end is a model of abstract, modality-independent interactors that a user needs to be able to perceive, understand, and operate in order to fulfill the system's FINs. The FIN and USI components of the model are related by transformation functions that translate user-understandable input into the system-understandable input that the device requires. In this way, USIs fulfill FINs. The double-ended aspect of the FIN-USI model is unique among interface models and accounts for situations where what a system needs is different from what a user needs and understands. The relationship between USI and FIN elements (where each element is an input component) is shown in Figure 3-1.

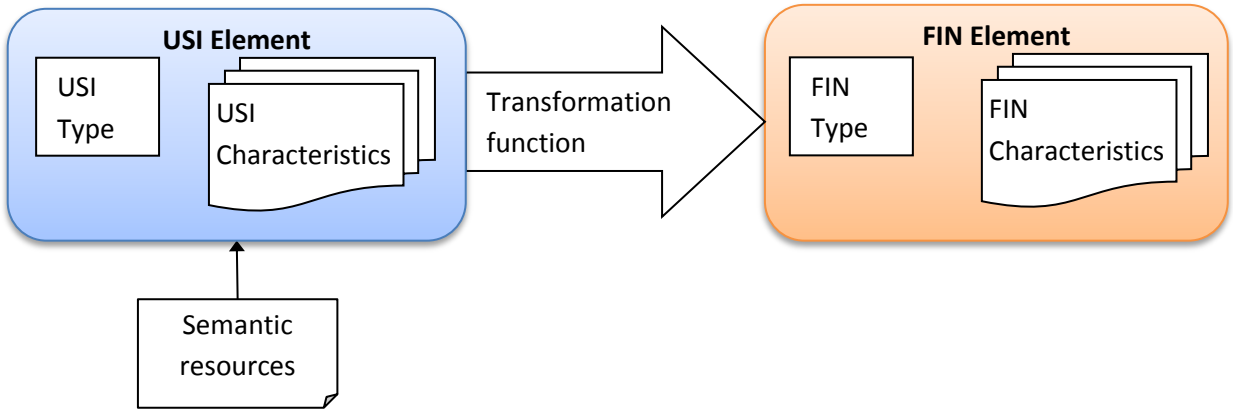


Figure 3-1. The relationship between a USI and FIN Element and their subcomponents. The value that a user inputs to a user-sensible input (USI) element is transformed to fulfill a functionality input need (FIN). Semantic resources (see Section 3.3.6) are applied to USI elements to make the purpose of the input understandable to users. An individual FIN or USI element is comprised of a type and any applicable characteristics.

Both the FIN and USI models have three major components:

- **Types:** The *FIN Types* are the raw data or other input that a system needs, whereas the *USI Types* are user-understandable forms of data and other input.
- **Characteristics:** The *FIN/USI Characteristics* define the manner in which the FIN/USI types must be provided.
- **Groups:** The *FIN Groups* identify data that must be grouped together for submission on the functionality end, while the *USI Groups* organize and provide context to the user input.

In short, the FIN/USI types are *what* a system needs and a user must provide, the FIN/USI characteristics are *how* the inputs are provided, and the FIN/USI groups are how the inputs are *organized and submitted*.

3.3.1 FIN Types

The FIN Types are a hierarchical organization of the types of input that functionality needs (see Figure 3-2). In practice, the model is extensible to accommodate custom data type values that a system might require.

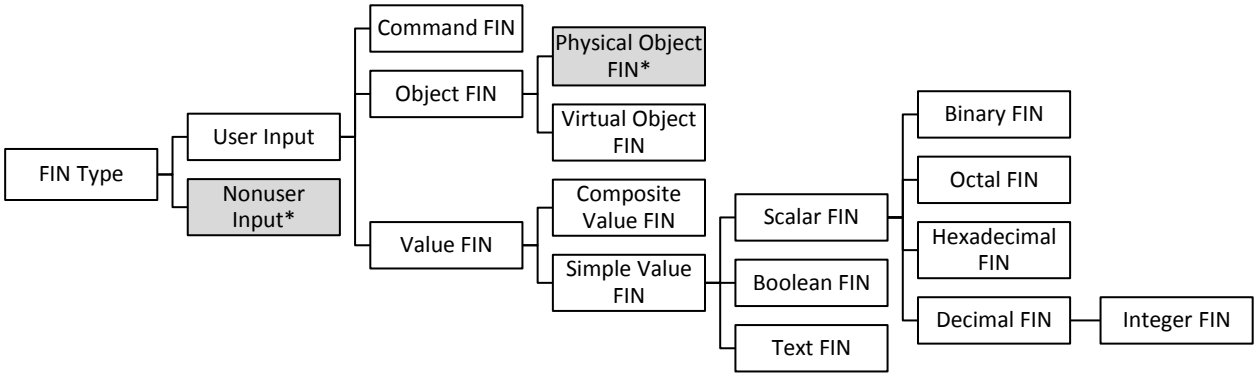


Figure 3-2. Hierarchical class diagram of the Functionality Input Need (FIN) Types. Classes that are not the focus of the paper (Nonuser Input & Physical Object Input) are marked with asterisks and shaded. They are included in the FIN model for completeness.

The input to a system may come from the user or from non-user sources (e.g., data from sensors, such as GPS, from network sources, or other channels). The User Input FIN type can be further broken down into Command, Object, and Value FIN types. With *Command FINs* a system requires a request or command to be issued by the user for the system to enact some functionality. With *Object FINs* a system requires physical (objects, environments, or events in the real physical world—for example, a copier needs paper) or virtual objects (data or files) for functionality. Physical objects (such as using a biometric scanner, loading paper into a copier, having a voice recorded for speaker recognition, providing printed text for optical character recognition and translation, or use of a specific physical control) have inherent barriers to some people with disabilities that cannot easily be ameliorated with software-based changes to an interface. *Value FINs* are variables or data that a system accepts as input and are among the most common types of input. These values are understandable to the system and may or may not be

understandable to users in their raw form. *Simple Value FINs* correspond to atomic data types that a system accepts, including Boolean, text, and scalar values. Composite Value FINs are comprised of other Value FINs. For example, an ISO-8601-format date-time FIN is a Scalar FIN with a representation like "2007-01-06T15:20:55-06:00". Another system might use a different date-time scale, such as a UNIX-epoch-date-time FIN (also a Scalar FIN) with a representation like "1168118455". As another example, systems that need the user to input a color might internally use different representations of color with different color models: for example, RGB, RGBA, HSV, HSL, CMYK, and others (which are all Composite Value FINs with multiple Scalar FINs along different dimensions) or color spaces such as Pantone, which can be described with ID numbers (which is a type of Text FIN). Value FIN types are practically unlimited because they must meet the specific needs of a system's functionality which may vary broadly between systems.

An interface generator based solely on FINs does not meet Goal 14 (because FINs may not be understandable in raw form to users) or Goal 3 (because there are a practically infinite variety of FIN types). Instead, it is proposed that an interface generator use the USI Types, which are limited in number, and which can be converted with transformation functions to fulfill the FIN Types.

3.3.2 USI Types

The USI Types are a hierarchical organization of types of basic input that are understandable to users (see Figure 3-3). It is restricted to user input that can be provided through software-based interactors. The USI Type model does not fulfill the Nonuser Input and Physical Object FIN Types because those needs cannot be met with the software-based interactors of PAUIs. The USI types can be used to fulfill FINs directly or with transformation functions (discussed below in Section 3.3.6).

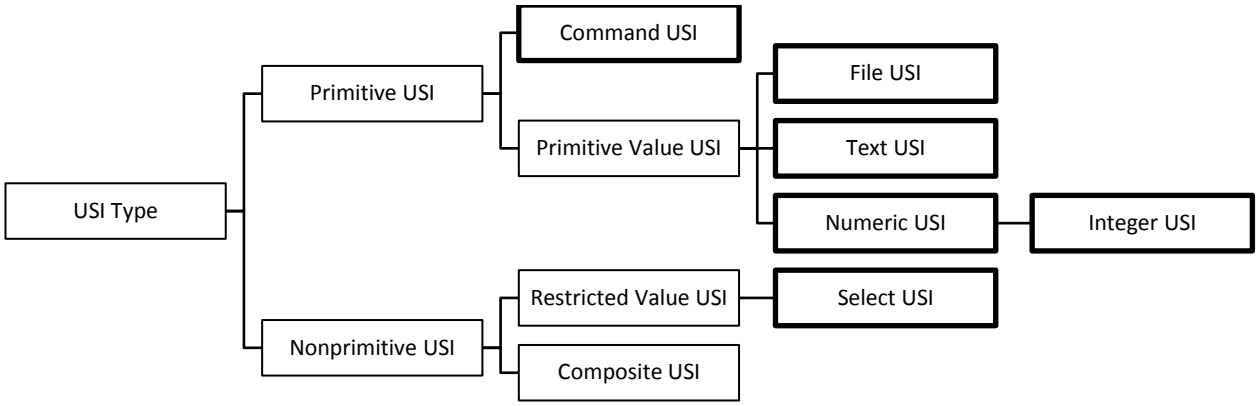


Figure 3-3. Hierarchical class diagram of the User-Sensible Input (USI) Types. Classes marked in bold boxes (Command, File, Text, Numeric, Integer, and Select USIs) are base USI types.

There are two categories of input in the USI model: Primitive and Nonprimitive USIs. *Primitive USIs* are atomic inputs that cannot be subdivided further into multiple user-sensible inputs. *Nonprimitive USIs* are composed of other Value USIs, and some are included as base USIs, which must be supported by all UI generators because they are very common in user interfaces.

A Primitive USI can be either a Command USI or a Primitive Value USI. The *Command USI* directly fulfills the Command FIN as a request or command issued by the user. In UIs, there are many ways by which users may issue a command: for example, they may click a button or menu item on the screen, press a shortcut key combination on a keyboard, make a gesture or swipe on a touchscreen, or enter a command at a prompt.

The Primitive Value USI class is further subdivided into three types of input: File, Text, and Numeric USIs. A *File USI* is a reference to a data file or an entity in a specific file that a system needs as input. The File USI directly fulfills the Virtual Object FIN. A *Text USI* is a string of character data that a system needs to function, and directly fulfills the Text FIN. A *Numeric USI* is any rational number that can be represented in decimal form. With appropriate transformations, Numeric USIs can be used to fulfill many types of Scalar FINs.

The USI Types are further extensible by creating *Nonprimitive USIs* in two ways: by specifying validation rules which restrict the values of one of the base types (Restricted Value USIs) or by defining Composite USIs which are composed of multiple Value USIs. A *Composite USI* is a group of related Value USIs that can be used to fulfill more complex FIN types that can be decomposed into individual atomic values. For example, a system may need a date in ISO 8601 extended format, such as "2009-04-04." While the date could be directly entered by the user in that form as a Text USI, in many cases, it would be better to use a Composite USI with the date split into three atomic Primitive Value USIs: year, month, and day.

Restricted Value USIs can be defined by restricting the values of Primitive Value USIs. There are many ways of restricting values (some of which are included in section 3.3.3.3 below), but one of the most important and common restrictions is enumerating all possible values so that users must select from that list—a Select USI. A Select USI can fulfill a Simple Value FIN directly (e.g., selecting a mode from a list) or through a name-value pair transformation function if that is more understandable to users (e.g., a user picks names from a list that are then transformed to the database ID numbers associated with each name). In mainstream UIs, there are many ways to make a selection, including choosing an option from a drop-down menu, clicking radio buttons or checkboxes on a form, keying a code for an item from a vending machine, or using a touchtone phone to choose an option in an automated voice menu system.

All value inputs, whether primitive or derived, must be submitted to the system. If the Value USIs are part of a Composite USI or Submission group (see section 3.3.4 below), then the individual inputs must be submitted as a group, not individually. The method of submission can be explicitly defined with a Submit USI or selected by an interface generator to fit the data being submitted, user preferences, and any task-related or other metadata that may be available. For example, when entering a temperature on a thermostat, some people may prefer that the temperature be submitted automatically when they have stopped adjusting it for 5 seconds, others may prefer the temperature to submit when they touch or

navigate elsewhere in the interface, and still others may prefer to submit the temperature by pressing an "OK" button or Enter key. Note that the users may have preferences for submission because they are unable to use one or more of the other approaches.

Interface generators must at a minimum support all of the base USI types. Interface generators may also optionally support some extended USI types with specialized interactors, especially for common types of entry with widely-used interface generators. For example, a Date Composite USI (which contains the primitive USIs: year, month, and day), could have its own interactor for some platforms and PAUIs. A specialized date-interactor might look like a calendar picker for some users and a set of drop-down menus for others (see Figure 3-4). Not all generators will have specialized interactors for a given extended USI, however. In these cases, users will use the interactors associated with the base USIs.

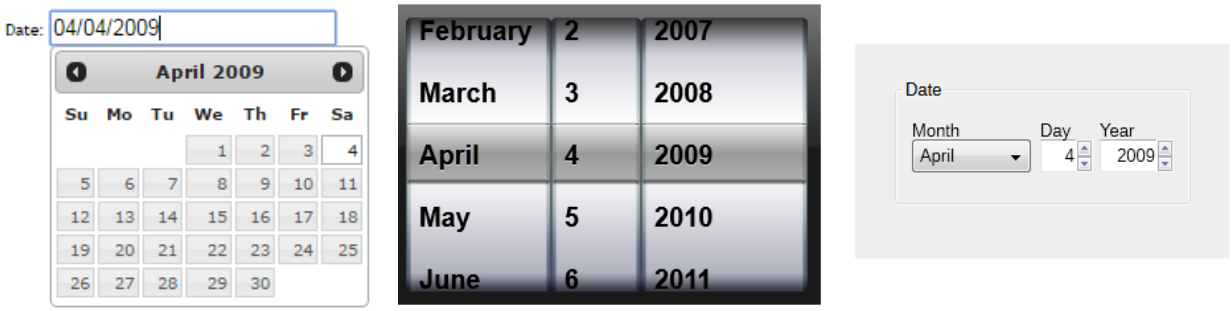


Figure 3-4. Three example interactors for a Date Composite USI: a calendar picker, a set of date barrel rolls, and a set of three basic input fields. The calendar picker and date barrel rolls are specialized interactors for the Date Composite USI. The three basic input fields might be the default type of interface that is generated based on decomposing the date into base USIs.

3.3.3 FIN & USI Characteristics

The FIN and USI models share the same characteristics, which relate to the *manner* in which input must be provided to the system in order to be accepted. The combination of the FIN Types and FIN Characteristics make up an individual FIN element (similarly with USIs).

3.3.3.1 Input Cardinality & Order

Input cardinality is the count of the number of individual object or value inputs that a system requires or allows. *Minimum cardinality* is the minimum number of values/objects that are required for valid input. FIN/USI elements with a minimum cardinality of 0 are optional. The maximum cardinality characteristic defines the maximum number of values or objects that can be input. For example, "Vote for up to 5 candidates," has a minimum cardinality of 0 and maximum cardinality of 5.

For multiple inputs, the order of the values may or may not be important to the system's functionality—this is the *order dependent* FIN/USI characteristic. For example, the order of input is important if users are supposed to rank their choices or provide a two-dimensional path, which is a series of XY values.

Typically the cardinality characteristics of a FIN and related USIs are the same, but there are some exceptions (e.g., a specialized Text-based USI, which is a text field that allows for any number of email addresses separated by a semicolon could be parsed by a transformation function into multiple email address FINs).

Typically, the same interactors can be used for both optional and required inputs. The only change practically necessary is labeling or otherwise designating a required input, which might be done with text, a symbol, or a change in how the interactor is presented (e.g., changing the color or the type of voice used to denote a required input).

The cardinality of input does not present an accessibility barrier to people except that providing multiple values requires more time. The type of input, whether single or multiple, does have a large effect on which interactors are appropriate. For example, many common GUI widgets only allow for single inputs, such as radio buttons, drop-down lists, spinners, and text boxes. To allow for multiple inputs, different GUI widgets are often used (e.g., checkboxes for multiple selections). However, it is frequently

possible to accommodate multiple inputs by providing multiple single-input interactors and/or a way to add additional single-input interactors (see Figure 3-5).

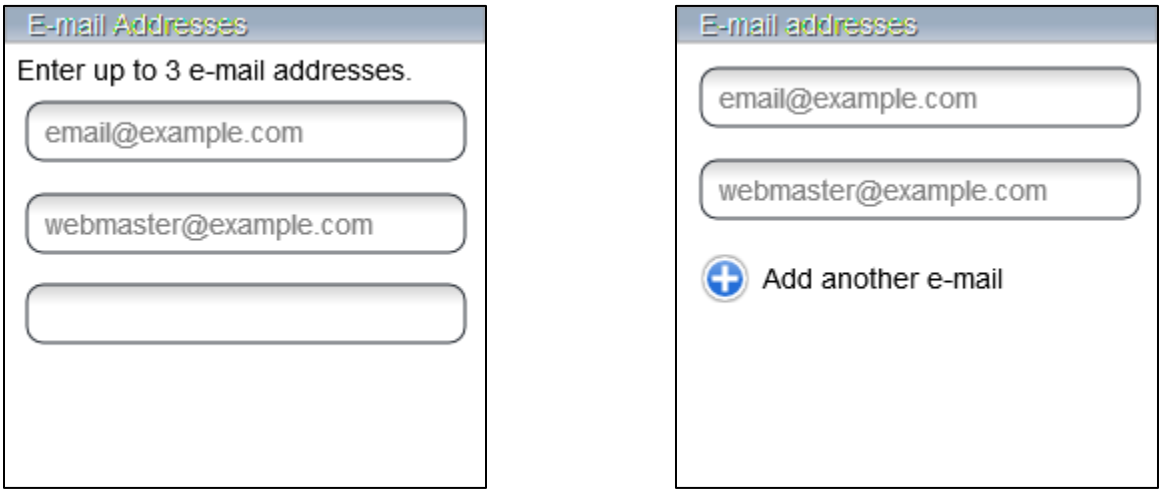


Figure 3-5. Two general strategies for allowing multiple input using only single-input interactors. On the left, an interface with three single-input fields (where the maximum cardinality is 3). On the right, an interface providing a button to add another single-input field.

3.3.3.2 Input Time Dependence

Time dependence is an important characteristic of input that can be applied to all FINs and USIs. Input is *time dependent* if the meaning or success of the user's input depends on when the input is submitted. If the same input submitted at a different time or for a different length of time, yields a different effect, it is time dependent. There are three types of time dependence:

- *none*: where there is no time dependence of the input.
- *momentary*: where only the moment of time at which the command is activated or value is submitted is important. For example during a teleconference a person might virtually "raise a hand" to signal that they have something to say. If one were to raise one's hand late, he or she might be much later in the queue of people who also have something to say.

- *continuous*: where values require continuous control or adjustment. For example, a person driving a vehicle with an automatic transmission needs to be able to continuously adjust at least three scalar values: the angle of the wheels, the pressure on the brakes, and the throttle.

The time dependence characteristic is the same between related FINs and USIs.

3.3.3.3 Input Validity

In order to function properly, a system needs values that are valid. There are many possible ways to validate data ranging from very simple rules to extraordinarily complex algorithms that rely on formulas that relate multiple input values and internal state variables. From the system's perspective, an invalid value simply needs to be replaced by a valid one, but things are more complicated from the user's perspective. Users need to know the location of invalid values, what they did wrong, and what types of values are actually expected. Note that validation properties may be different between a FIN and related USIs where there is a transformation between different FIN and USI value domains.

Validation can always be handled by notifying the user what inputs have invalid values with a helpful error message. However, it is better to prevent user input errors than to have the users fix errors after making them (Molich, Ede, & Kaasgaard, 2004). Many interactors prevent users from entering invalid data: for example, a slider widget does not allow a user to enter a value that is outside of a defined range. Despite the practically infinite number of methods of validation, there are some that are very common. Numeric values may be bounded by maximum and minimum values and further validated by conforming to a fixed step size between subsequent valid values. Text values may be validated by being compared to minimum or maximum string lengths and to regular expression patterns. Enumeration, which forms the Select USI base type, is a particularly common and important method of validation that allows for convenient interactors that list the available choices.

3.3.4 FIN & USI Groups

All but the simplest user interfaces have groups or container elements (such as screens, windows, dialog boxes, toolbars, tabs, and fieldsets) that are used to structure and organize the elements. Some functionality requires several inputs to be provided simultaneously—these are FIN groups. USI groups may also group simultaneous input, but they can also provide structure to make a UI potentially less cluttered and easier to understand and use. Groups can be nested to further organize the input.

Some functionality requires several inputs to be provided simultaneously—these are FIN groups. USI groups may also group simultaneous input, but they can also provide structure and context to an interface to make it potentially less cluttered and easier to understand and use. Groups can be nested to further organize the input. There are three types of grouping in the model: semantic, simultaneous, and submission groups.

A *semantic group* is a USI-only group that contains USI elements that are grouped in a logical manner. Semantic groups are not required by the functionality, but serve to organize USI elements, provide context, and aid understanding. For example, a media player might have a group of player controls and a group of volume/mute controls. As another example, logical groups might be used to organize USI light switches into separate rooms. By having a more branching interface with navigation between groups, semantic groups can be used to create an interface that fits better on small screens or for users who need less clutter. Alternatively, interfaces can be flat with many semantic groups displayed on a screen. Semantic groups should have a sensible label, which may be used as a menu link, title, or as the legend of a fieldset.

A *simultaneous group* is a set of time-dependent FIN or USI elements that must all be available to the user at the same time. The user must have access to all the elements of the simultaneous group in order to provide the input that a system needs. For example, a user must be able to use all of the primary driving controls of a car (steering wheel, brake pedal, throttle pedal, shifter, etc.) at the same time. Live

action games have similar simultaneous groupings of controls, where a person must be able to control many variables simultaneously.

Both simultaneous groups and semantic groups characterize elements that should be in the same interaction context (e.g., controls that must be on the screen at the same time or that make sense as a coherent whole). In contrast to the third group type (submission groups) simultaneous and logical groups do not require that the grouped inputs are submitted together.

A *submission group* is a set of non-time-dependent, value FIN or USI elements that must be submitted as a set to the system at the same time for validation, processing, or other reasons. A submission group may be a composite FIN/USI or may contain related elements. While all of the input of a submission group must be submitted to the system at once, the FIN or USI elements are not required to be in the same window or screen of a generated interface. An interface generator may create a wizard-style interface from a submission group: where a user fills in just a few pieces of information on each screen of the wizard, and then the data is buffered and submitted to the system at the end. Submission groups are a common feature on websites, for example, where a person might provide credit card and address information that a system needs in order to complete a transaction.

The individual elements of a submission group cannot be submitted individually but must be submitted with the group. The submission can be done with an explicit Submit Command or the interface generator can create an appropriate method of submission automatically (e.g., allow the Enter key or a specific touchscreen gesture to submit the submission group). The submission of a submission group may or may not be dependent on time.

3.3.5 USI Semantic Resources

The USI elements are input that a user understands. However, a user also needs to know the purpose and context of their input. The functionality does not need these semantics, but users need them to understand an interface. This information can be provided to users through semantic resources:

instructions, helpful error messages when invalid data is submitted, group titles, and input labels. These semantic resources are task specific and may also be specific to a particular language, culture, and output modality. Semantic resources for a USI element can potentially be provided in multiple versions, which might have different lengths and modalities (e.g., abbreviated text suitable for a small screen or dynamic braille; long, descriptive text; and icons as labels). An interface generator could choose the most appropriate resource from the multiple versions.

Labels and titles are a particularly important way to help users understand the current context and what input they are to provide. Labels are required in accessibility standards (e.g., Caldwell et al., 2008; ISO, 2008). Each USI element should have a label so the person knows the purpose of the element's input. A Composite USI must have a label for the composite and each Primitive Value USI (e.g., "Birthday" for a date Composite USI and then "Year," "Month," and "Day" for the primitive inputs).

3.3.6 Transformations: FIN-USI Relationships

A USI and a FIN may have a one-to-one relationship or there may be multiple FINs or USIs that are related to each other. A USI is sensible to users by definition; however, systems only understand the FIN-form of input, which may or may not be the same as the USI form. A transformation layer is necessary to convert one or more USIs to one or more FINs. It is envisioned that the transformation layer would be provided by the manufacturers, but third parties could potentially define alternative transformation functions from USIs to FINs. The transformation layer is not a new concept; in software, it is common to check, transform, and normalize user input into something that is useful to the system. The various types of relationships between FIN and USI types are diagrammed in Figure 3-6 and discussed below.

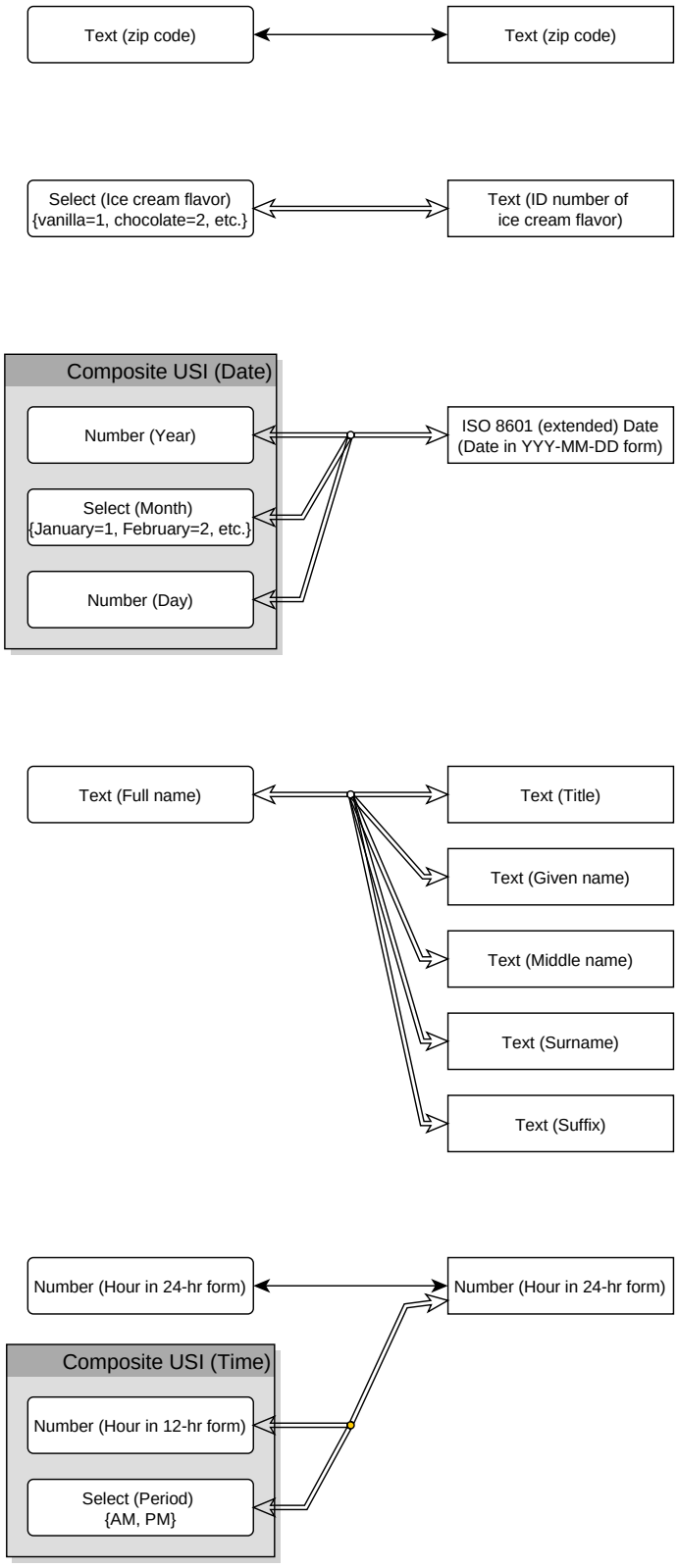


Figure 3-6. A diagram of various relationships between USIs and FINs. In this diagram, the USIs are on the left and the FINs are in the right column.

1. A direct relationship (identity transformation) between USI and FIN.
2. A transformation function required between a single USI and a FIN.
3. A Composite USI with multiple Primitive USIs that can be transformed to fulfill a single FIN.
4. A single Text USI that can be parsed and transformed to fulfill several FINs.
5. A FIN with two alternative USIs: the first is a direct relationship and the second is a composite USI.

When a FIN is sensible as is, then the FIN is a USI and the identity transformation function is used. For example, the number of copies a photocopier is to make is both a FIN and a USI. The USI would need an appropriate label to be sensible, but the data value is understandable to both the user and system.

Input from USIs may need to be transformed for several reasons. One major reason for a transformation function is that people may not understand the FIN form of value input they are to provide. For example, a system may need a country code in ISO 3166-1 Alpha-2 format (e.g., "us" for the United States and "ch" for Switzerland). While some of the two-letter country codes may be sensible to users, there are many that may not be sensible because they are abbreviations from different languages and may not be the most natural abbreviation. The USI in this case might be for the user to select from a list of country names, which are transformed to their two-letter code representation for the FIN.

Transforming a USI to a specific FIN form can be used to normalize the entry and prevent a user from making errors where their expectations are different from those of the system. With free input, where the user is not constrained to pick from a set of values, a user might not use the exact input form required. A user might type text in mixed case, enter a phone number or credit card with characters to separate number groups, or enter a decimal number with a different radix point (e.g., "3,14" or "3.14" or "3·14"). Normalization transform functions include converting all characters to upper- or lowercase, removing whitespace characters, removing punctuation character, and replacing the radix point character.

In some cases, it might be easy and intuitive for a user to provide a string of text input which can be parsed into multiple FINs. For example, a system might be able to parse an address entered in a multi-line free text field into several FINs: the street address, the city, state, and zip code for US addresses. Note that this particular approach should be used with caution (or as an alternative to more structured input) because for some people, text input is slower than other types of input.

Where a single FIN may be composed of multiple atomic values, then a many-to-one USI-FIN relationship may be appropriate and more sensible to users as a composite USI. The composite USI is comprised of multiple primitive USI. For example, a date of the form YYYY-MM-DD could be modeled as a "Date" composite USI with three primitive USIs: Year, Month, and Day. These three primitive USIs could then be transformed into the FIN form by concatenating the user input.

For some FINs, there may be several, alternative USIs (and their respective transformation functions) that are appropriate. The choice of alternative USI might depend on the needs or preferences of the user. For example, if a system needs a time in 24-hour format, some people may prefer a USI with 24-hour format (with an identity transformation) while others may prefer a composite USI with a 12-hour format value and AM/PM selection (that are then transformed into the FIN form). These alternative value inputs can provide flexibility to the user and interface generator. Different people may prefer different forms.

3.4. Application of the FIN-USI Model

The two-sided nature of the FIN-USI model makes explicit the separation between the input needs of systems' functionality and what the user must do to provide those inputs. The USI-side of the model may be used as a base vocabulary of modality-independent inputs for an abstract UI. The USI elements can straightforwardly be applied to typical form input widgets found in GUIs. The USI model can also be used to describe other types of input that do not fit into a form fill-in interaction style. See Table 3-1 for examples of the USI model applied to different types of input. Appendix B shows more detail about USIs relating to additional interactors.

Table 3-1. The USI model applied to various user inputs.

Input or Task	USI Type	Cardinality minimum	Cardinality maximum	Order dependent	Time dependent
Form input: Text	Text	*n*	1	-	none
Form input: Numeric	Number	*n*	1	-	none
Form input: single selection	Select	*n*	1	-	none
Form input: multiple selection	Select	n_1	n_2	false	none
Form input: Date (YYYY-MM-DD)	Composite: Number, Select, Number	*n*	1	-	none
Drawing program: Path of xy-coordinates	Composite: Number, Number	1	∞	true	none
Teleconferencing: Hand-raising functionality	Command	-	-	-	momentary
Live auction: Bidding	Number	1	1	-	momentary
Voting: Select up to 3 candidates	Select	0	3	false	none
Voting: Select & rank up to 3 candidates (rank voting system)	Select	0	3	true	none
Basic driving: simultaneous control over (1) wheel angle, (2) brake pressure, & (3) throttle.	Simultaneous group: Number, Number, Number	1,1,1	1,1,1	-	continuous (×3)

3.5. Inherent Task-Related Barriers to Access

It has been recognized that certain types of tasks cannot currently be made accessible to people across disabilities (Jordan & Vanderheiden, 2013). With current technology, we do not know yet how to create interfaces or interactors that allow access to these kinds of input.

Time dependence can be one of the main barriers to accessibility. With some input there is no known way for some people to provide the input in the expected time frame with the expected level of accuracy, especially when input is continuously time dependent. These barriers may be caused by a wide variety of factors that cannot easily be mitigated by changing an interface's interactors. Some people move more slowly than average because of physical disabilities. Those who have to use an interface in

which they navigate and step through options are typically slower than those who can directly select options or commands with a mouse or touchscreen. Some people might be using interaction styles that have their own time dependence (such as automatic scanning interfaces) and they cannot generate input at any arbitrary rate. Getting information through speech output or reading braille is inherently slower in many situations than looking at a screen and reading text.

Simultaneous groupings of inputs can also be a barrier to accessibility. These interactions are time-dependent and require a person to manipulate two or more controls at the same time. Some people can only easily manipulate one control at a time because of paralysis or other physical disabilities. Other people can only focus on one task and control at a time and would have difficulty trying to coordinate interactions with multiple controls.

3.6. Improvements over Existing Models

Existing models for the generation of user interfaces cover basic form and data entry interfaces well. However, they have poor coverage of other forms of input. The FIN-USI model was created to address this systematic shortcoming.

Table 3-2 is a comparison of several UI models. The USI model has fewer base input types than other models, which was a conscious decision (Goal 3). Having more input types is both a benefit and a liability: more differentiation potentially allows for fine control and tuning of interfaces to fit particular types of input, but it also increases the number of interactors that must be created in order to support the differentiated input. For people with uncommon disabilities and for those who need novel methods of interaction, it is likely that relatively few interactors will be available or be created.

Table 3-2. A comparison of selected interface models.

Key: **X** = complete support for the characteristic, / = partial support, 0 = no support

Model	Comments	Input Elements in Model	Cardinality Support	Order Dependence Support	Time Dependence Support
HTML5 (Hickson et al., 2014)	A few of the input elements in HTML are redundant and serve the same purpose.	26 input elements	/	0	/
WAI-ARIA (Craig et al., 2014)	WAI ARIA specifies non-abstract input roles that can annotate controls. States and properties can be applied to modify their features.	12 input roles	/	0	/
XForms 1.1 (Boyer, 2009)	XForms has a limited number of core input elements, but may rely on additional extended datatypes, which are not specified in XForms.	9 core input form control elements	/	0	/
XML Schema datatypes (Biron & Halhotra, 2004)	While not a standard for UI models, the XML Schema Datatypes are used in other specifications, such as that of URC (ISO/IEC: 24752:2014)	44 built-in datatypes	/	0	0
UsiXML AUI model (UCL, 2013)	UsiXML has many layers. Applicable abstract UI layer components are considered here.	13 abstract input interaction units	**X**	/	/
PUC (J. Nichols, Myers, Litwack, et al., 2004)	The PUC specification language was designed to support UI generation for appliances.	8 inputs	0	0	/
SUPPLE (Gajos, 2005)	The main focus of the SUPPLE project was not on modeling UIs but on generating GUIs from a declarative model.	7 primitive inputs	/	0	/
USI	A USI input element is fully specified by a base input (or composites) and its characteristics.	6 base inputs	**X**	**X**	**X**

The set of FIN-USI characteristics are a novel contribution of the model. Most other models allow for multiple selections, but, with the exception of UsiXML and limited HTML input elements, the

models do not allow for multiple inputs (i.e., cardinality maximums greater than 1) of arbitrary data. The order of multiple input values (the *order dependence* characteristic) may be important in some cases. Only the UsiXML model captured this consideration, but only for a specific array datatype. The time-dependence of an input is also an important consideration which may preclude some interactors and indicate others. The other models do not natively support timed input except implicitly with a command, trigger, or button input that could be used to invoke a command or submit values at a particular point in time. Timed input can potentially be added to some interfaces with scripting (e.g., JavaScript in HTML). However, this does not add these time dependencies to the UI model. Without these time dependencies in the UI model, a UI generator would be unable to choose interactors most appropriate to the needs of both the user and the system's functionality.

The two sides of the FIN-USI model also make explicit that users and systems may have different needs. Other models (except for the Common Access Profile) tend to take for granted that the input required by the functionality is already in a user-sensible form. In reality, systems may require input that is not user-understandable in raw form. This model underscores that transformation steps may be required for some input in order to be understandable to both users and systems.

3.6.1 Why not just swap UI widgets?

An interface generator based on the FIN-USI model would choose an interactor that suits the input needs of both the user and the functionality. This is a different approach than that which is commonly used today to adapt interfaces to make them accessible.

The current general approach to accessibility is to create an accessible interface element-by-element from an existing one (which is typically a GUI). Each element of the original interface is re-imaged into an equivalent one that is accessible to a user. For example, a radio button in a GUI would have an equivalent representation by a screen reader—a spoken radio button—where the screen reader

would read the name of the control, its role (“radio button”), its state (“unchecked”), and perhaps instructions that tell a person how to check that radio button.

This approach works well for a simple FIN that has a simple interactor that is a single widget. More complex FINs might be composed of multiple widgets, but this is where the element-by-element substitution approach may break down or be more awkward than it needs to be. The person is essentially using an AT interface in order to drive a graphical user interface in order to control the functionality. To illustrate this, take the example of a list builder interface shown in Figure 3-7.

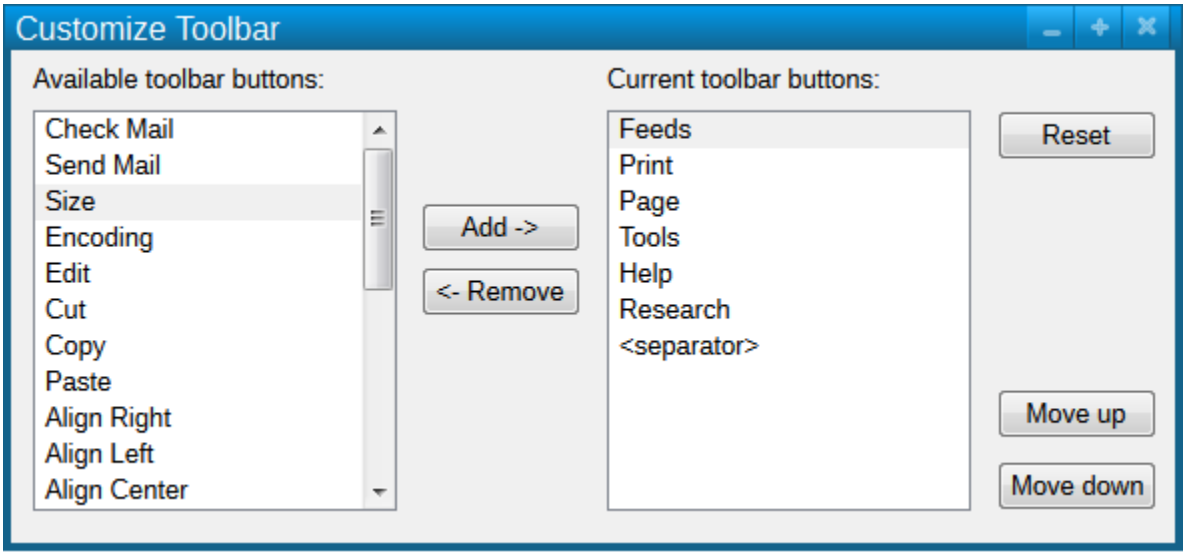


Figure 3-7. A List Builder interface, which is a common user interface design pattern.

A List Builder is a common user interface design pattern that is known by a number of other names: dual list, dual listbox, list shuttle, shuttle, swaplist, etc. A list builder has two lists in which a user can make a selection: a list of available selections and a list of current selections, which is being built by the user. Using an “Add” or “Remove” button, the user can move selected items from one list to the other. The user then uses the “Move up” and “Move down” button to reorder items in the current selection list. Different list builders may have additional buttons, such as one to move all items from one list to the other or buttons to move a selected item to the top or bottom of the current selection list.

Swapping individual elements of the list builder for accessible versions is straightforward to do (this is what AT would do), but ignores the purpose of the list builder. The functionality input need that the list builder fulfills is an ordered, multiple selection. There are other alternative interfaces that are suitable to fulfilling that need. For example, the selection and ordering could be done in two steps with an interface like that in Figure 3-8.

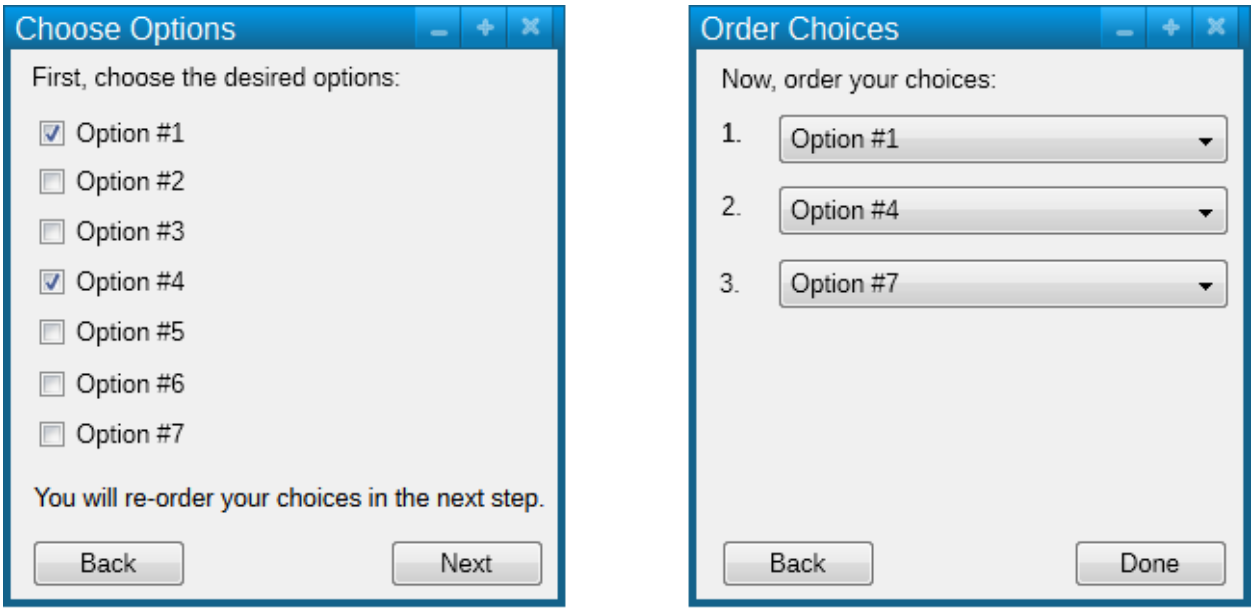


Figure 3-8. A two-step interface for making an ordered, multiple-selection from a list of options. With this interface, the person first selects the desired items and then orders them. On the left is the first dialog box with several choices that can be selected with checkboxes. On the right is the second dialog box that allows one to order the choices by selecting the desired choice for first, second, etc. from a set of drop-down selections.

As another alternative, a person might like to drag to select and sort items. With such an interface (see Figure 3-9), the user drags items from one list to another. The order of the items can also be changed by dragging.

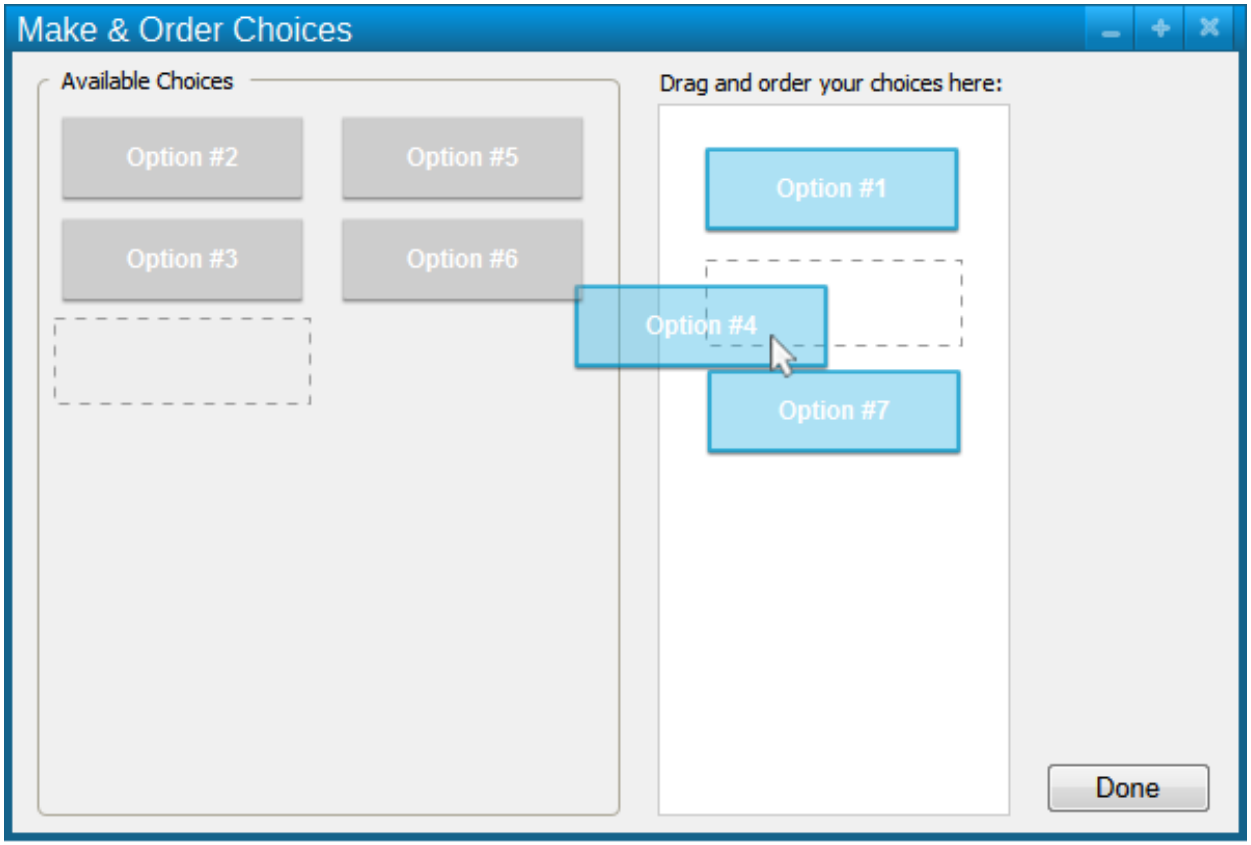


Figure 3-9. A drag-to-sort control that has two columns. The user drags the desired choice from the list of available choices to the second column. The order of the choices can also be adjusted by dragging within the second column.

By choosing interactors based on the functionality input needs, particularly where FINs are typically met with a pattern of widgets, an interface can be better fit to the user. This is likely to be better than providing an accessible interface by naïvely swapping individual interface elements.

3.7. Conclusion

There has been significant work done in the field of model-based UI generation and many models have been developed. However, many of these models are complex, which makes both the creation of product-specific abstract UIs and the creation of specialized UI generators more complex. This complexity of models and interface generators is a disadvantage to the people who most need auto-

generated UIs: people with disabilities and specific needs that are not accommodated with mainstream interfaces. The FIN-USI model is a simpler model of input that builds on existing work by proposing a set of characteristics that can be applied to different input types in order to support more paradigms of input. The USI model could potentially be extended to add additional properties or characteristics (see Appendix C for a potential list of additions) that may improve the usability of generated interfaces. The USI model might itself be used as a base abstract UI model for input to a system, or the concepts may be used in the development of new or revised abstract UI modeling languages.

Chapter 4.
FIN-USI Model Validation Part 1:
User Interface Generators

The first stage of validating the FIN-USI model is its ability to be used to create a user interface generator that could successfully generate alternate user interfaces that met the input needs of the devices. The second stage of validation would be to see whether or not these automatically-generated interfaces were in fact better than the original interfaces on the products. The ability to beat a special interface that was designed by the manufacturer specifically for the population used in the validation study was also evaluated in the second stage of the validation (Chapter 5).

4.1. User Interface Generators

Today, most user interfaces are hand-crafted for mainstream users. This allows for tight control over the look and feel of an interface, but also potentially excludes people who were not considered during the design process. The automatic generation of interfaces is a potential solution that could increase access to devices and systems for people with disabilities. An interface generator is the collection of algorithms that creates an interface from different inputs that relate to real-world domain objects such as the target device, controller device, and user.

An interface generator has to take in a number of different parameters as input, and then generate a user interface through algorithms that operate on and integrate these inputs. Required inputs depend on the interface generator, its features, and at what point it generates the interface. An interface generator that dynamically adapts to changing environmental conditions requires different inputs than a one-and-done interface generator that creates a few interfaces for predefined devices and mainstream users at development time. Interface generator input parameters can be collected into related parts that can be

modeled or profiled together. Figure 4-1 is a general overview of the interface generation process with the different input models/profiles that interface generators may need. These general input models and profiles are also described in more detail below.

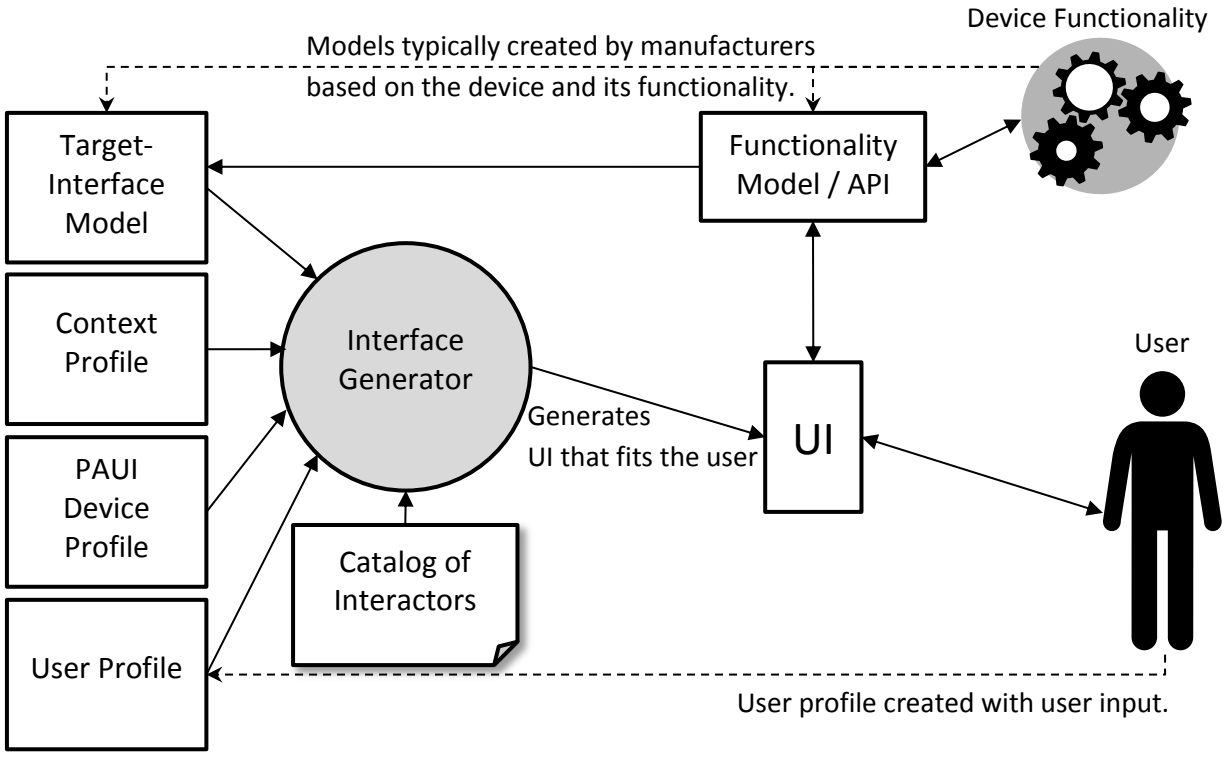


Figure 4-1. A diagram of the inputs an interface generator needs to create a personalized interface. Depending on the system, the input profiles and models may be split into subcomponents.

The **target-interface model** is one of the most important generator inputs because it describes in an abstract, non-modal manner the interface intended for people to use. This target-interface model is closely related to the **functionality model**, which describes the functionality of the system, its inputs, and outputs. Most systems and frameworks combine the target-interface and functionality models together into a composite model that abstractly describes the intended interface and the necessary APIs that need to be called to cause changes in the interface to be enacted upon and reflected by the device functionality. The FIN-USI model maintains the separation between the target-interface and functionality models with a

translation function that translates between input that is user understandable and input that in a system-understandable form. The target-interface model provides the interface generator with hints about the interface's labels, structure, and other information to make it more sensible to those who would use the generated interfaces. An interface generator may use these hints or override them if that would make more sense for a particular user.

Interface generators that create personalized interfaces require a **user profile** as input. User profiles may be created with a variety of approaches. A profile might attempt to model characteristics of particular abilities, such as the SUPPLE++ system, which modeled a person's pointing performance (Gajos et al., 2007). Another approach is to collect the needs and preferences of individuals into a profile rather than try to model the abilities of a person. Some generators can use information about previous usage of the device or similar devices to change the interface's layout (Gajos et al., 2010) or structure (J. Nichols, Myers, & Rothrock, 2006). Different approaches may be blended in constructing a user profile that may cover multiple dimensions of access and usability. More complex generators may support more features and nuances of user profiles.

Interface generators that are not specific to a particular PAUI device must have a **PAUI profile** of the characteristics and constraints of the interface device and platform for which the UI is being created. This PAUI profile might contain information about the PAUI's platform, resolution of the screen, the types of input that must be supported, what interactors can be presented, and other characteristics. Even PAUI-specific interface generators have a PAUI profile, but this profile is an implicit part of the interface generator itself in that the generator only creates interfaces that fit the PAUI.

Interface generators that create dynamic interfaces tailored to particular environmental, usage, or other contexts often include a **context profile** as input. Such interface generators often work in real time during the usage of the interface to adapt it to changing conditions. There are many potential changes that could be made due to changes in context. For example, systems might change the color scheme and

contrast in dark environments, increase the volume in noisy locations, or create a simpler interface when a person is driving a vehicle.

Many interface generators create interfaces using atomic subcomponents known as interactors. *Interactors* are the components or widgets with which a user interacts to get information from or provide input to the system. Some interactors are very generic, such as a button or a set of checkboxes, and can be used in many different circumstances. Other interactors are more specific to particular inputs and interaction styles, such as a visual calendar picker for selecting dates. Interactors are modality specific, relying on specific input mechanisms (such as a touchscreen, mouse, or keyboard) and providing feedback through specific modalities. Some interactors may have support for multiple modalities, but many others can only be used with very specific input and output devices.

A catalog or collection of these interactors can be used in the automatic generation of personalized user interfaces. The interface generator selects appropriate interactors from the interactor catalog that fits the context, target device, PAUI, and user.

4.2. Three Newly Developed UI Generators

Three user interface generators were developed as part of the project to validate the FIN-USI model. These were then tested with users to evaluate their efficacy and usefulness. These generators have similar inputs. Two target-interface models (one for a smart thermostat and one for a copier) were created with the FIN-USI model; these were inputs for all three generators. User profiles for each generator were created using each person's preferences for specific interactors that were collected as part of a preference elicitation session. None of the interface generators had an explicit PAUI model as input, because the generators were programmed for a specific PAUI. The three interface generators had very limited contextual adaptation to the user during the experimental sessions, so no context profiles were used. However, context adaptation could be a feature added to future generators.

The three user interface generators that were developed are called the List, Loop, and Button/Panel interface generators (see Figure 4-2). The List was designed to be immediately familiar to mobile device screen reader users, and the Button and Panel interfaces were designed to be familiar with screen reader users at ease with keyboard operation. The Loop interface was a novel interface type that was intended to improve efficiency, but was unfamiliar to users. All three generators created self-voicing interfaces designed for people who could not see well enough to use screen magnification or other visual strategies. The interfaces were created specifically for the PAUI device (iPod Touch) that was used in the user study with web standard technologies (HTML, CSS, and JavaScript). The research interest was related to users' usage, performance, and satisfaction with the generated interfaces, so the generated interfaces were not operationally connected to target devices. This operational connection and control has been well demonstrated by the PUC (J. Nichols et al., 2002), URC (Zimmermann & Vanderheiden, 2005), and other systems.

Figure 4-2. Photographs of List, Loop, and Panel interfaces. All of the interfaces were on an iPod Touch PAUI device. The List and Loop interfaces were operated with touchscreen gestures. The Panel interface has a plastic panel overlay that tactilely delimits were the five on-screen buttons are that are used to navigate and operate the interface. The Button interface (not shown) added physical buttons in the location of each hole in the plastic panel of the Panel interface, but the interface usage was exactly the same. Rulers in the figure are marked in cm.

The following algorithm was used for generating the different interfaces from the target-interface models. The generator would take each USI in order from the device model. For each USI, the interface generator would find the interactor from the catalog that best matched the Input USI and the user's interactor preference profile. The interface generator would try to construct an interface using the matching interactor. If the interactor could not be used in the interface because an exception was thrown when it was being instantiated, then a default interactor was used to construct that component of the interface. The generator used the labels and data associated with each USI. The generator did not try to reinterpret, rearrange, or translate any of the information about the USI.

The interface generators were also responsible for laying out the interface and adding interactors to the interface for operation. These operation interactors (such as "Back" or "Submit" buttons on screens,

pop up keyboards, etc.) were not specified in the device model but were added by the interface generator to make a usable interface. The List interface generator generally laid out the interface elements in a single column of screen-width interactors. With the Loop interface generator, all elements of the interface were arranged in a clockwise manner around the edges of the touchscreen to potentially make them easier and more efficient to find and access by dragging (see Appendix D). Both the List and Loop interfaces used gestures on the touchscreen for control (these are described in section 4.3). This gesture interaction was based partially on how people who are blind interact with the VoiceOver and TalkBack screen readers on iOS and Android devices respectively. Some of the other features of all three interfaces were based on EZ Access cross-disability access techniques developed at the University of Wisconsin-Madison (Law & Vanderheiden, 2000).

The Button/Panel interface generator supported two interfaces (the Button and Panel interfaces) that were used in a small case study. Instead of using gestures on the touchscreen, users interacted with the Button and Panel interfaces by tapping physical or virtual on-screen buttons respectively. This was akin to using a keyboard or other button set to navigate and use interfaces with screen readers.

4.3. Basic User Inputs & Gestures to the Interfaces

The three automatically-generated interfaces had a limited set or vocabulary of inputs that a person could do: navigating/moving through an interface, activating elements, changing an element's value, and moving to earlier and later pages of a screen. A wider vocabulary of inputs would be useful for some people because they could then directly make particular gestures to do other things with an interface. Both the VoiceOver (iOS) and Talkback (Android) screen readers have many more gestures that a person can use. However, for this study it was felt that a simpler input vocabulary was better for training and operation.

This section details the various modality-dependent physical inputs and gestures that users could make to control the three generated interfaces.

4.3.1 Navigating & moving the focus

Two methods of moving through the interface were implemented: stepwise linear navigation and direct pointing exploration. Both of these ways of moving through the elements of an interface are familiar to people who have used screen readers on modern touchscreen devices. Neither of these techniques is new. Navigation in a stepwise manner has been used by mainstream interfaces and early screen readers. One of the earlier examples of exploration by pointing on a touchscreen is the Talking Fingertip technique developed at the University of Wisconsin-Madison for use on accessible touchscreen kiosks (Vanderheiden, 1996). This exploration is also akin to how people who are blind use magnifiers in order to enlarge and read text visually (although no speech output is provided from magnifiers).

With *stepwise linear navigation*, users can make a gesture or press buttons in order to move the focus to the next or previous elements. When the focus moved, the new element was read through text-to-speech. All of the elements of the interface could be reached by navigation. On the Loop and List interfaces, quick rightwards and leftwards swiping gesture navigated to the next and previous elements respectively. This horizontal swiping gesture is used by both the VoiceOver (iOS) and TalkBack (Android) screen readers. On the Button/Panel interface, there were two dedicated on-screen buttons in a single row for navigating to the next and previous elements.

When using *direct pointing exploration*, participants drag, single-tap, or long press on the screen—when they touch an element that was unfocused, then that new element is focused and read to the user. Similar to VoiceOver and EZ Access features, this system made a sound effect (a quiet, white-noise hissing sound) when a user's finger was over blank areas of the touchscreen although the focus remained on the last-touched element. Unlike VoiceOver, there was no snap-to feature (where the focus sometimes moves to a close-enough element even if the person did not touch it) because it was felt that it might lead

to confusion about the actual size and position of on-screen elements. Direct pointing exploration was the default behavior for single-finger gestures that users made whenever the gesture was not recognized as any other valid form of input (i.e., swipes and double-taps). This behavior means that failed gestures often caused the focus to move, which can be disorienting, but is something to which users of touchscreen screen reader systems (such as VoiceOver) have grown somewhat accustomed. Direct pointing exploration was a feature that was only included in the List and Loop interfaces.

4.3.2 Activating elements

When focused, most of the interactors could be activated, which might lead to opening a menu or keyboard, going to a different screen, re-reading a non-interactive element, or other behaviors. Activation was accomplished by a double-tapping gesture anywhere on the screen for the List and Loop interfaces (as with both VoiceOver and TalkBalk screen readers) and by a dedicated button at the bottom of the screen on Button/Panel interfaces. VoiceOver also includes other gestures that can be used for activation that were not included in this system to keep things simple.

For these interfaces, most of the double-tap parameters were constants (see Table 4-1), but the double-tap rate (the allowed time between the two single taps of a double-tap) was a parameter that could be initially set and then automatically adjusted on the fly for each participant. The initial double-tap rate setting was done as part of the experimental protocol. The double-tap rate was limited to 250 – 1000 ms between taps. Throughout the use of the interface, the system would also look for “near double-taps”: two single taps that were within 50% of the current double-tap rate. The most recent five double-tap and near-double-tap rates were averaged and the double-tap rate was then reset if the average was more or less than 50 ms different from the current double-tap rate.

Table 4-1. Parameters and values used for recognizing double-taps as the activation gesture.

Parameter	Default Value	Comments
Distance	300 px (46.7 mm)	The maximum distance allowed between the two single taps that composed the double-tap. This value was made relatively large so that users did not have to try to tap really close together in order to have a recognized double-tap.
Tap Time	250 ms	The maximum time on the screen that either single tap could take and be considered as a valid single tap that might be part of a double-tap.
Double-Tap Rate	Variable (Constrained to 250 – 1000 ms)	The maximum allowed time between single taps that were to be considered together as a double-tap.
Wiggle	16 px (2.5 mm)	The maximum distance on the touchscreen that the user's finger could move or wiggle during a single tap.

4.3.3 Directly changing values

Some interactors allowed a person to change the values directly without having to activate it first. For example, a person might navigate to a volume control and increment its value by a few steps without having to open a menu or keyboard. With another interactor, a person might simply press up or down to change their selection easily to one of the other allowed values.

When a person had focused on an interactor that allowed this type of manipulation, they would quickly swipe up or down on the List and Loop interfaces or tap the dedicated Up and Down buttons on the Button/Panel interface. This is a gesture similar to the gesture used in VoiceOver to increment or decrement values for slider widgets (i.e., `<input type="range">` in HTML) or change values in pickers.

4.3.4 Scrolling & changing screens

All of the interfaces would automatically go to the next or previous part of the screen if a person navigated to an element that was not yet displayed on the screen. Users could also directly change to a different screen by using gestures on the List and Loop interfaces. On the List interface, users would make a three-finger swipe gesture upwards to go to a later part of the page (like pushing the page away)

and the reverse gesture to go to an earlier part of the page (like pulling the page)—these gestures are used in VoiceOver on iOS devices. On the Loop interface a person could get to later elements by dragging from the top left to top right corner of the screen (i.e., continuing a clockwise looping gesture at the top of the screen) and the reverse gesture to go back to earlier pages of elements. While making the Loop's dragging gesture, users could still move the focus to any elements that are at the top of the screens. The Button/Panel interface did not have a gesture or buttons for going up or down pages. Users just moved through the elements with the Next and Previous buttons.

4.4. Results

The FIN-USI model was successfully used to create three user interface generators that did in fact create alternate interfaces different from the manufacturer interfaces. The efficacy of these interfaces relative to the manufacturer interfaces was tested in Part 2 (Chapter 5).

Chapter 5.
FIN-USI Model Validation Part 2:
User Testing the Model-Based UI Generators

The automatic generation of user interfaces has been a topic of research for over 30 years (Hix, 1990; Meixner et al., 2011). Many frameworks and interface generators have been created, but there is a paucity of published user studies with generated interfaces.

This study aims to both evaluate the efficacy of the model-based UI generators on the usability of systems, and to add to the literature on auto-generated interfaces by including interface types and participant groups that before now have not been studied and reported on. Two interface generators that create self-voicing, gesture-based interfaces for mobile touchscreen devices were created. These interface generators were specifically designed for people who are blind. Blindfolded people were also included in the study as a proxy for people who are newly blind with no experience with screen readers and technology.

This chapter details the user experiment of these auto-generated interfaces compared to manufacturer-created interfaces that represent the current state of user interface for people who are blind.

5.1. Participants

Blind participants were recruited by making an announcement at a state blind convention, sending study advertisements to various local and statewide email lists that were of interest to people who are blind, and by having an e-mail message sent by a local vocational rehabilitation counselor to clients. Sighted participants were recruited through posters that were placed in stores and libraries in the local community. Participants completed a short set of screening questions over the phone. Participants were screened out of the study if they reported significant physical difficulty that would hinder their ability to

use a smartphone or if sighted participants felt that they could not wear a blindfold for at least 20 minutes at a time. The recruitment target was for 12 people who were blind and 12 people who were sighted. Participants were paid for travel expenses and at a rate of $15 per hour for their participation.

In total 12 people who were blind and 15 people who were sighted were initially recruited into the study. Three sighted individuals were dropped from the main study. The first two sighted individuals were dropped as run-in participants as the researchers identified problems with the experimental methods. These methods were changed before the rest of the sighted participants were run. The first sighted, run-in participant was only able to complete the first half of the Phase 2 tasks in the time that he had available on that day. For the second, run-in participant, a longer was session available, and he did all of the tasks with a lunch break in the middle. Fatigue effects and frustration were very apparent with this participant; by the end of the session, he was failing most tasks, even tasks that had been simple in the first replication. After this point, the experimental protocol was revised to allow for two sessions for Phase 2 for participants who needed it and the data from the first two sighted participants was discarded. One sighted participant was also screened out of this study because of observed difficulty during Phase 1, which took him 75 minutes (42%) longer than the next longest Phase 1 session. This participant was invited back later to be part of a case study which had additional interfaces available for testing. No participants were screened out because of disability or difficulty wearing blindfolds.

A total of 12 blind participants (6 female) with an average age of 37 (*SD*=12.0) and 12 blindfolded participants (8 female) with an average age of 37 (*SD*=15.5) completed both phases of the study. The blindfolded group had more racial (11 white, non-Hispanic and 1 Hispanic blind participants compared to 8 white, 3 black, and 1 Asian blindfolded participants) and educational diversity than the blind group (all 12 blind participants had education beyond high school, whereas 3 of the 12 blindfolded participants had a high school education or less). A few sighted participants reported disabilities that might have had an effect on their performance during the sessions: two participants had ADHD, one of

which also had a diagnosed nonverbal learning disorder. It was also observed that one of the blind participants had hand tremor, with the observed result that the iPod Touch would sometimes misinterpret the gestures that the participant actually intended.

5.2. Instrumentation

During the experiment, participants frequently interacted with an iPod Touch (Model ME643LL/A, Apple Inc., Cupertino, Calif) running iOS 8.2. The iPod Touch was connected to a computer and a mixer for recording and to a speaker to allow for louder and clearer audio. The iPod Touch was placed in a modified OtterBox (Fort Collins, Colo.) Defender Series case that had the screen protector removed for better responsiveness and a 1.9-mm-thick plastic piece added to block the top 4.3 mm of the screen with the iOS clock, icons, and notifications bar. For some parts of the experiment, participants used VoiceOver (a screen reader built into iOS) and for other parts of the experiment, the interface was programmed to provide speech output using the same voice as VoiceOver. Participants also interacted with the physical controls (Figure 5-1) of a smart thermostat (Model CT80, Radio Thermostat Company of America, San Francisco, Calif.) and a multifunction copier (Model X654de, Lexmark International, Inc., Lexington, Ky.). Participants used both the thermostat and iPod Touch on a table to make it easier to video record their interactions.

Figure 5-1. The thermostat and controls on the copier. Both systems had a touchscreen and physical buttons. Rulers in the photographs are marked in cm.

5.3. Methods

The main purpose of the experiment was to compare users' performance, preference, and satisfaction with manufacturer-created interfaces, which are the current state of technology, and automatically generated interfaces. The two devices in the study had their interfaces modeled using the FIN-USI model, which aims to be a simple model that can cover a broad range of applications and functionality. The two interface generators were basic: they used the FIN-USI model of the interfaces and then built an interface using the interactors that each participant preferred for each modeled input or output. There were no nuances or extra information included in the interface models or generators that could tune interfaces to particular tasks. In this experiment, four general interface types were tested, which for the purposes of this chapter are called the Mfr-Physical, Mfr-Item, Gen-List, and Gen-Loop interfaces (see Table 5-1 and Figure 5-2). (The prefix *Mfr-* denotes manufacturer-created interfaces and *Gen-* denotes automatically-generated interfaces.) When participants were introduced to the interfaces, they were called the Physical, Item, List, and Loop interfaces, respectively.

Table 5-1. The interfaces tested in the experiment.

Interface (Device)	Source	Description
Mfr-Physical (Both devices)	Manufacturer created	The physical interfaces on the tested devices. Both interfaces had physical buttons and touchscreens for control. Neither interface had speech output—only basic beeps and tones.
Mfr-Item (Thermostat only)	Manufacturer created	Native iOS application. It worked with VoiceOver (screen reader), but was not specifically designed for accessibility. The interface failed two WCAG 2.0 (Caldwell et al., 2008) provisions. • 2.4.2 "Focus Order" because some menus opened up in confusing locations earlier in the application's navigation order. • 2.4.6 "Headings and Labels" because labels for different elements were read as a single block rather than with each element.
Mfr-Item (Copier only)	Manufacturer created	A web-based interface specifically designed to be accessible and usable to people using screen readers on computers. It used very simple links and form fields, and met all level-AA WCAG 2.0 (Caldwell et al., 2008) provisions. It was specifically chosen for testing because it is an exemplar of accessibility among mainstream device manufacturers.
Gen-List (Both devices)	Automatically generated	Interface elements were generally laid out in a conventional list layout with one interactor or element per row. It used similar gestures as VoiceOver.
Gen-Loop (Both devices)	Automatically generated	Novel interface layout and gestures with interactors and elements arranged around the edges of the screen in a clockwise direction. It was designed to be used by dragging around the edges of the screen (although it could also be used with many of the same gestures as the Gen-List interface) to quickly find elements.

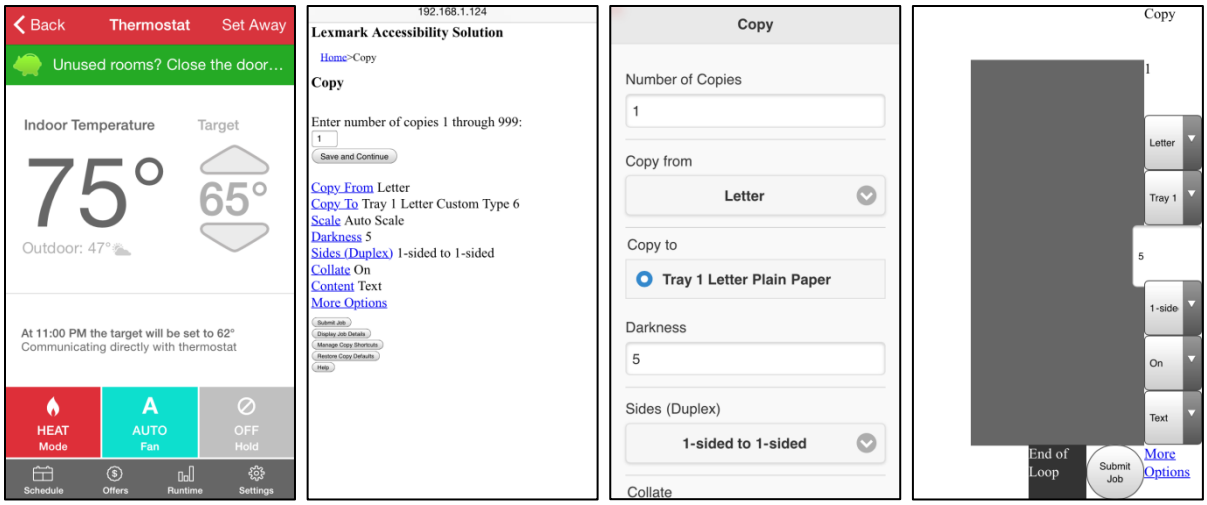


Figure 5-2. Screenshots of tested interfaces. The first and second interfaces are manufacturer-created (Mfr-Item) interfaces for the thermostat and copier respectively. The third interface is a representative, auto-generated Gen-List interface for the copier. The fourth interface is a representative, auto-generated Gen-Loop interface for the copier, where the elements start in the upper right corner and continue clockwise around the edges of the screen.

The experiment was in two phases. In the first phase, the participant's preference for interactors was elicited. The second phase was a comparative study of the manufacturer-created interfaces and automatically-generated interfaces using the participant's Phase 1 preferences. Sighted participants wore a blindfold while training and performing tasks. The interfaces were covered when participants took breaks from the blindfold.

Participants were told at the outset of the study that the researchers were comparing different interfaces for people who were blind. They were told neither the hypotheses of the experiment nor that they were choosing interactors and having customized interfaces generated for them. Furthermore, the experimenters who interacted with the participants during the study sessions were also not told the hypotheses of the experiment and were familiar only with the procedures required to carry out the data collection part of the experiment (see Appendix E for questionnaires and Appendix F for scripts).

Before the experiment was conducted, the significance (α) level of 0.05 was chosen for all statistical tests.

5.3.1 Phase 1: Preference Elicitation

After participants arrived for the first session, a consent form and demographic questionnaire were administered. The volume and rate of the iPod's text-to-speech output was then adjusted. The speech rate was incrementally slowed until the participant could correctly repeat five consecutive, randomly-ordered phrases that the system spoke from a list of phrases that the system was likely to say during the latter phase of the experiment. Finally, before the training session, the participant double-tapped the screen a number of times to record and set their baseline double-tap rate (constrained to be between 250 – 1000 ms).

Participants then went on with training on the three touchscreen interface styles (Mfr-Item, Gen-List, and Gen-Loop) on the iPod Touch. All three of the introductory training interfaces (see Figure 5-3) had a set of on-screen buttons, one for each letter of the alphabet.

- With the Mfr-Item training interface, which utilized iOS's VoiceOver feature, the alphabet buttons were arranged in a grid with 3 buttons per column. VoiceOver would step through them in alphabetical order when the horizontal swiping gestures were used.
- With the Gen-List training interface, the buttons formed a single column, like a list.
- The Gen-Loop training interface had alphabet buttons arranged around the periphery of the screen.

The order of the three training interface was randomly ordered and balanced between participants. Participants were instructed how to navigate (i.e., move the focus around) the self-voicing interfaces by stepping (making short right or left swiping gestures) or by dragging their finger around the interface. Participants were told how to change pages (three-finger swipes up or down on the Mfr-Item and Gen-List interfaces and a continuing looping gesture on the Gen-Loop interface) and activate items (double-tapping anywhere on the screen when an element is highlighted). Finally to ensure that participants had a basic understanding of the three interface styles, they were instructed to find and activate randomly

named on-screen buttons until they had activated five buttons correctly in a row. After training on all the iPod self-voicing interfaces, participants ranked them according to preference and commented on them.

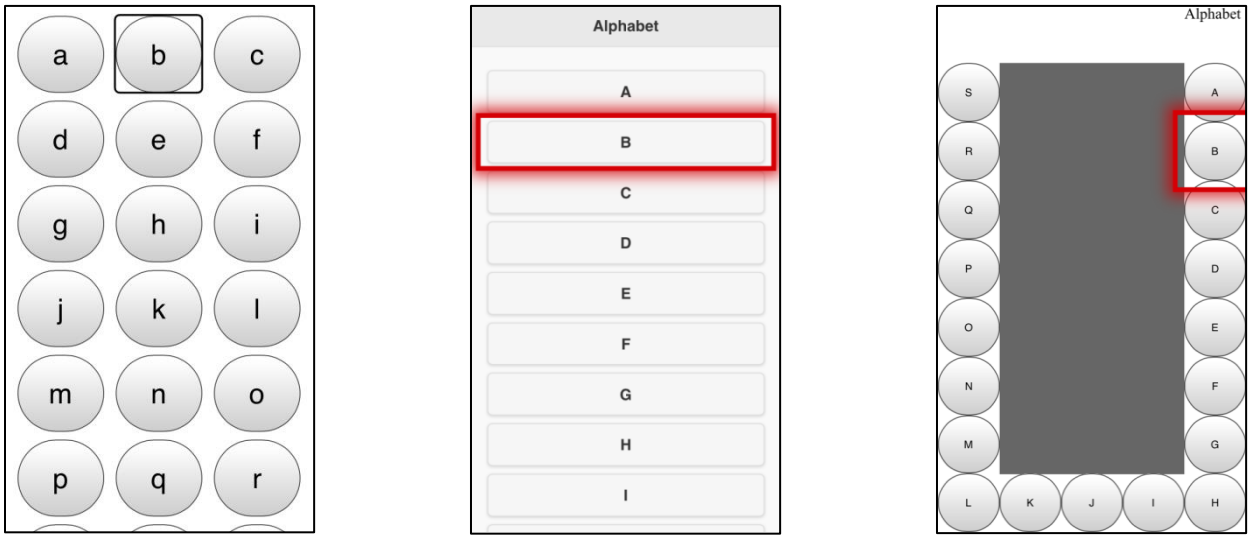


Figure 5-3. Phase 1 training system screenshots showing the arrangement of alphabetic characters of the Mfr-Item, Gen-List, and Gen-Loop interfaces (from left to right). The entire alphabet does not fit on a single screen in any of the arrangements.

The last and longest part of the Phase 1 session was eliciting the participants' preferences for particular interactors for specific types of input. Preference elicitation interfaces were constructed using the same two types of interactors of interest (for example, two dropdown menus or two sets of radio buttons). For each preference elicitation task, participants had to navigate past the first interactor on the screen to get to the second interactor, which they would need to manipulate. Participants would start by trying a random pair of interfaces with the same task and choose their favorite (ties were allowed). Their favorite interface from the prior pair was then paired with a random challenger interface, and they were then to try and rank those two interfaces. This pairwise comparison process was repeated until the favorite interface of all of the available ones was identified. Then the entire process was repeated for the next task and interface style. Participants started with the Gen-List interface and performed three sets of tasks in order: select-one-from-two, followed by select-one-from-seven task, and then numeric entry task. They

then repeated the three sets of tasks and interactor comparisons with the Gen-Loop interface. Participants were also asked to score each interactor after their first use it on a 7-point scale with anchors of 1 = “terrible” and 7 = “excellent.” All of the interactors tested are listed and described in Table 5-2.

Table 5-2. The interactors from which a person could choose during Phase 1. Each participant's favorite interactors were used to build personalized interfaces for Phase 2.

Interactor Name	Description	Tested Interfaces	Applicable Tasks
Picker	(Based on iOS VoiceOver representation of an HTML `<select>` element.) The set of choices was represented by a dropdown menu. A value was chosen by through a multistep process: (1) double-tapping the dropdown menu to open a picker at the bottom of the screen, (2) tapping in the picker to focus it, (3) swiping up or down to change the value stepwise, and (4) finding and double-tapping the picker’s done button which was in the top right corner of the picker.	Gen-List	Select-1-of-2, Select-1-of-7
Link Select	(Based on the Copier’s web interface.) Each choice was represented by a link that was the label followed on the same line by text, which described the current value. A value was chosen by double-tapping the link, which opened a separate page with a menu of links. Double-tapping a menu link made the choice.	Gen-List	Select-1-of-2, Select-1-of-7
New Page Select	A value was chosen by through a two-step process: (1) double-tapping the button on the screen to open up a menu on a separate screen and (2) finding and tapping on the desired menu button to make a choice.	Gen-Loop	Select-1-of-2, Select-1-of-7
Hybrid Select	Allowed for two methods to selecting a value: (A) Swiping up or down to change the value stepwise or (B) through a menu that behaved just like the New Page Select above.	Gen-List, Gen-Loop	Select-1-of-2, Select-1-of-7
Compact Radio Buttons	A set of radio buttons, with one for each choice that was checked or unchecked. The group label for the set of radio buttons was appended to the value of each radio button. A value was chosen by activating and thus checking the desired radio button.	Gen-List, Gen-Loop	Select-1-of-2, Select-1-of-7
Header Radio Buttons	A set of radio buttons preceded by a header, which would let the user know the group label associated with the radio buttons that followed. A value was	Gen-Loop	Select-1-of-2, Select-1-of-7

Interactor Name	Description	Tested Interfaces	Applicable Tasks
	chosen by activating and thus checking the desired radio button.		
Switch	The switch allowed a person to toggle between two values by double-tapping it.	Gen-List, Gen-Loop	Select-1-of-2
Cycle Button	The cycle button allowed a person to cycle through a set of values by double-tapping multiple times until the desired value was found.	Gen-List, Gen-Loop	Select-1-of-7
Range Input	(Based on iOS VoiceOver representation of an HTML slider `<input type="range">`) Changing a value was accomplished by swiping up or down, each swipe changing the value by a step, as many times as necessary.	Gen-List	Numeric
Keyboard	(Based on iOS VoiceOver representation of an HTML number input field `<input type="number">`) Changing a value required a multistep process to type a new number: (1) double-tapping to open the number-and-symbol keyboard at the bottom of the screen, (2) finding and double-tapping the delete key to clear the field, and (3) finding and double-tapping keys to type characters. The keyboard would close if the person touched or focused on other interactive screen elements that were above the keyboard.	Gen-List	Numeric
Keypad	Changing a value required a multistep process to type a new number: (1) double-tapping to open a numeric keypad at the bottom of the screen (but which also prevented access to the rest of the screen), (2) finding and double-tapping keys to type characters, and (3) finding and double-tapping the keypad's "Done" button in the lower right corner.	Gen-List	Numeric
Swiper Keypad	The interactor had two methods for changing a value: the person could (A) type as with the Keypad interactor above or (B) swipe up or down to change the value stepwise (similar to the Range Input interactor).	Gen-List	Numeric
Typer	Changing a value required a multistep process to type a new number: (1) double-tapping to open a loop numberpad, (2) finding and double-tapping keys to type characters, and (3) finding and double-tapping the keypad's "Done" button.	Gen-Loop	Numeric
Swiper Typer	The interactor had two methods for changing a value: the person could (A) type as with the Typer interactor	Gen-Loop	Numeric

Interactor Name	Description	Tested Interfaces	Applicable Tasks
	above or (B) swipe up or down to change the value stepwise.		
Slider	Changing a value required a multistep process to select a new number: (1) double-tapping to open a slider with a column of buttons on the right side of the screen, (2) dragging/sliding along the edge of the screen to find the desired number, and (3) double-tapping to choose that number.	Gen-Loop	Numeric

If there were two or more interactors tied for favorite for a given task, then the system automatically chose the interactor to be used for the auto-generated interfaces. The system's choice was based on the participant's tied interactors and a listing of interactors in a preference order determined before the study by the researchers.

5.3.2 Phase 2: Comparative Interface Testing

The main purpose of Phase 2 of the experiment was to gather performance, preference, and satisfaction data for comparing the four interfaces on the two devices. Phase 2 was split into two sessions for the few participants who were slower: one session for the initial factorial experiment and the second session for its replication.

Each Phase 2 session started with setting the three personal parameters as in Phase 1: volume, speech rate, and double-tap baseline. Afterwards, participants tried a short gesture practice session to become familiar again with the basic gestures used in the experiment: swiping right and left to navigate, double-tapping to activate, swiping up and down to change values, and three-finger swiping up and down to change pages. In the gesture practice sessions, participants were told a randomly-ordered task they were to accomplish (e.g., "Next", "Activate", or "Earlier Page"), and they would respond by making a gesture. Participants had to get four of each gesture correct the first time they were cued before they could continue to the training.

Participants were also trained on the five interfaces that they were to use: the copier's Mfr-Physical interface, the thermostat's Mfr-Physical interface, and the Mfr-Item, Gen-List, and Gen-Loop interfaces on the iPod Touch. For the three iPod Touch interfaces, participants were trained using the preference elicitation interfaces and tasks (from Phase 1) with their favorite interactors (the Mfr-Item training interface used the Link Select and Keyboard interactors, which is what would be used on the copier's Mfr-Item interface). The five interfaces were trained in random order. After completing the training, participants were administered a questionnaire verbally before continuing with the factorial experiment.

The bulk of the data gathered was during the factorial experiment (Device [2] × Interface [4] × Task(Device) [5]) where participants tried every combination of Interface and Device with a set of five device-specific tasks (see Table 5-3). Participants would start with a random device (either the copier or the thermostat) and try each of the four interfaces (Mfr-Physical, Mfr-Item, Gen-Loop, and Gen-List) before moving on to the other device. The order of the interfaces was counterbalanced for each participant using a Latin Square. Participants were told to perform each task "as quickly and as accurately as possible." Participants were given no more than 2 minutes to complete a task, after which point the participant was told to move on and the task was recorded as a failure. After each task, participants verbally answered the Single Ease Question (SEQ; Sauro & Dumas, 2009). After each of the iPod-based interfaces, participants were verbally administered the System Usability Scale (SUS) questionnaire (Brooke, 1996). The SUS was modified to have 7-point Likert-scale items instead of the original 5-point items. This modification was made because it was found in pilot testing that participants became confused with switching between 5- and 7-point scales and because there is evidence that there are no significant changes between scaled responses of 5- and 7-point scale questionnaires (Dawes, 2008) and a 7-point SUS scale may provide more accurate measures than the original scale (Finstad, 2010). Participants were

asked to rank and make comments on all four interfaces after trying all of them for a given device. All of the questionnaires used in the experiment are in Appendix E.

Table 5-3. The device-specific tasks participants attempted on the different interfaces. Simple tasks were chosen so that participants would have higher rates of success. The asterisks denote tasks that would require the use of the (non-voicing) touchscreen on the Mfr-Physical device in order to complete the tasks.

#	Copier Tasks	Thermostat Tasks
1	Make a single copy.	Raise the target temperature 2 degrees.
2	Make a darker copy. *	Set the target temperature to 70 degrees. *
3	Make 5 copies.	Turn the fan on. *
4	Make a 2-sided copy from a 2-sided original. *	Change the mode to cool. *
5	Set the Content setting to “Photograph.” *	Set the target temperature to 75 degrees and turn hold on. *

Depending on the time, participants would replicate the factorial experiment in the same session or come back for a second Phase 2 session. To reduce frustration, participants did not replicate the tasks on the Mfr-Physical interface that were *a priori* considered to be inaccessible because they required the use of non-voicing touchscreens for success. Participants were allowed breaks whenever they felt they needed them and also took breaks between devices.

5.4. Results

The mean Phase 1 session length (including breaks) for blind participants was 124 min. (*SD*=39) and that for blindfolded participants was 137 min. (*SD*=33). The difference between Phase 1 session times between groups was not significant; t(22)=0.880, *p*=0.388. A total of 9 blind participants completed Phase 2 in one session with a mean length of 163 min. (*SD*=23); 9 blindfolded participants also completed Phase 2 in a single session (*M*=184, *SD*=22). The difference in single Phase 2 session times was not significant (t(16)=1.933, *p*=0.071). Of the 3 blind participants who had two Phase 2 sessions, the two

sessions were 205 min. (*SD*=18) and 128 min. (*SD*=23) long, respectively. The 3 blindfolded participants who had two Phase 2 sessions had mean session times of 160 min. (*SD*=10) and 132 min. (*SD*=35), respectively.

5.4.1 Observations on Gestures

During the Phase 2 sessions, it was observed that participants had different styles of navigation that they settled on. Some participants (3 blind and 2 blindfolded) swiped right and left most of the time to navigate. For these five participants, the Gen-List and Gen-Loop interfaces would have had similar experiences as their layouts did not matter when swiping to navigate. Many participants (3 blind and 6 blindfolded) primarily used dragging gestures to move through the elements of both the Gen-List and Gen-Loop interfaces—even though these participants dragged on both interfaces, their experiences between the two would have been different because of the different layouts. Many participants (6 blind and 3 blindfolded) used a mixed approach where they would switch between dragging and swiping gestures. All of these nine mixed-approach participants used more swiping navigation gestures to step through Gen-List interfaces and more dragging gestures on Gen-Loop interfaces. One blindfolded participant used a tapping approach to navigation, where she would tap the screen, wait, listen, and then tap again to look for other things. This tapping approach was not particularly effective as this participant had the lowest completion rate (30%) on the Gen-List and Gen-Loop interfaces of all participants.

Nearly all participants made at least a few gestures on the iPod interfaces (Mfr-Item, Gen-List, and Gen-Loop) that did not quite meet the gesture thresholds required to be interpreted as intended. Sometimes this would have no effect, but frustratingly often these missed gestures would be interpreted as an explore-by-touch gesture. This often caused the system's focus to move elsewhere, which was disorienting. This gesture misinterpretation happened with both intended swipe and intended double-tap gestures.

Making a misinterpreted gesture was somewhat more problematic on the Gen-List interface than on the Gen-Loop interface for participants. For Gen-List interfaces, interactors cover most of the screen since the interactors are the width of the screen. With Gen-Loop interfaces, the center of the screen does not have any interactors because all of the interactors are along the edges of the screen. The center of the screen was a "safe area" in a sense because gestures confined to the middle of the screen would not cause a change of focus. However, misinterpreted gestures that strayed outside of the Gen-Loop interface's safe area caused the focus to change, as with the Gen-List interface. Horizontal swiping gestures of those who stepped through the Gen-Loop interface seemed to be more problematic than vertical swiping gestures. This observation is sensible since the iPod Touch was locked in portrait mode and the safe area is taller than it is wide.

During the training of Phase 1 and gesture practice of Phase 2, it was observed that a few of the participants had difficulty reliably making particular gestures on the iPod interfaces. The blind participant who had visible tremor had particular difficulty with making double-taps that the system would reliably detect. The double-tap detection of the Gen-List and Gen-Loop interfaces was very tolerant of the distance between subsequent taps. However, this participant's double-taps were more "draggy" than those of other participants, with each component tap forming a short drag on the screen, which resulted often in a change of focus rather than the intended activation. This participant also strongly disliked and had some difficulty with the three-finger up and down swipes for changing pages. A blindfolded participant also had noticeable difficulty with the three-finger vertical swipes because her fingers were crooked after surgery on her hand. She had to contort her arm, hand, and fingers in order to get three fingers to touch and swipe across the iPod's touchscreen. After completing this study, both of these participants were invited back for a separate case study which studied additional interface types.

During Phase 2, participants would sometimes double-tap after they had already swiped to change an interactor's value. For the swipe-to-change interactors, this double-tap opened up a menu or numeric

keyboard. Participants who did this often reacted with some frustration that they had already set the value but now had to either re-set the value or close the new screen or keyboard. Some participants mentioned that they knew that they did not need to double-tap to set the value, but that it was a habit.

While data was not specifically collected on the relative performance of the swipe up and swipe down gestures, observers noticed that the swipe-down gesture was more reliable than the swipe-up gesture for many participants. This may be because the swipe-down gesture can be done quickly by index finger flexion (i.e., bending or curling the finger) while the swipe-up gesture was often done with full hand or arm movement.

During the experiment, the double-tap rate ended up being adjusted dynamically for 5 out of 12 blind and 5 of 12 blindfolded participants. It is not known how much help the dynamic double-tap adjustment was in practice as there was no control group or interface for comparison.

5.4.2 Phase 1 Results

In order to get each participant's favorite interactors for the automatic, model-generated Gen-List interfaces, participants tried all 5 available interactors for the Select-1-of-2 task, then all 5 available interactors for the Select-1-of-7 task, and then all 4 interactors for the number input task. Ties were allowed in the preferences. Participants also scored each interactor the first time they tried one on a 7-point scale anchored with 1= "terrible" and 7 = "excellent." These data are shown in Figure 5-4. There was a moderate Spearman's rank correlation coefficient of 0.442 between the 7-point interactor scale scores and the weighted favorite score for each interactor.

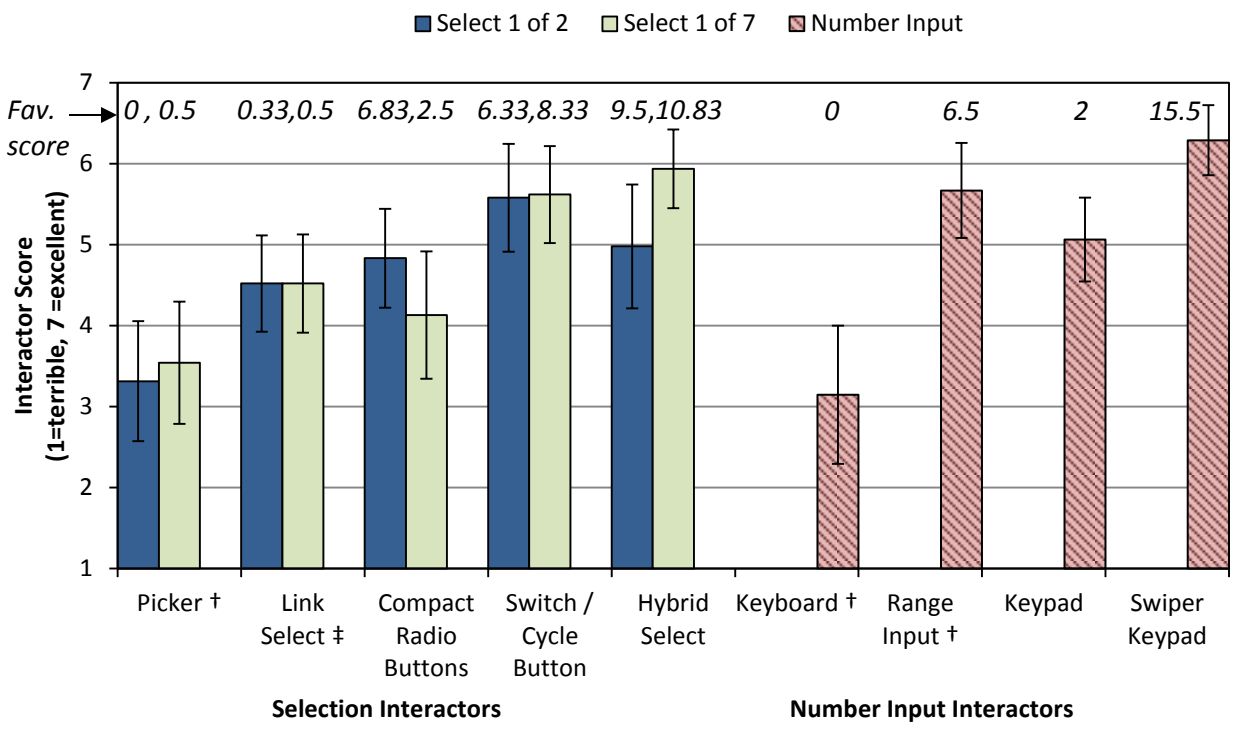


Figure 5-4. Interactor and fractional favorite scores for Gen-List interface interactors. The bars show the average score that participants gave each interactor on a 7-point scale with 95% CI error bars. The numbers in the top row of the chart denote the fractional favorite score of each interactor (i.e., how many people chose each interactor as a favorite with fractional scores for participants with tied interactors).

† Interactor closely based on native iOS VoiceOver screen reader interactors.

‡ Interactor closely based on those used in the copier's accessible web-based (Mfr-Item) interface.

For the two selection tasks (1-of-2 and 1-of-7) on the Gen-List interface, 7 participants chose the same interactor, 12 participants chose different interactors, and 5 participants had ties for at least some of the same interactors.

Several of the Gen-List interface interactors were closely based on pre-existing interactors that are used in the copier's accessible Mfr-Item interface or the native iOS VoiceOver screen reader. Participants favored the new interactors created for this study over the pre-existing interactors for all three preference elicitation tasks (see Table 5-4).

Table 5-4. Exact binomial tests of new interactors being favored over pre-existing interactors.

Task	Probability of success (i.e., favoring new interactors) due to chance	Number who favored new interactors (*k*)	Number who did not have ties between new & pre-existing interactors (*n*)	Significance (*p*)
Select 1 of 2	3 / 5	22	23	< 0.001
Select 1 of 7	3 / 5	23	24	< 0.001
Number input	2 / 4	15	19	0.010

Participants then chose favorites and scored the Gen-Loop interface interactors in a similar manner as with the Gen-List interactors. This preference data is shown in Figure 5-5. The Spearman's rank correlation coefficient between the 7-point scale scores and the weighted favorite score for each interactor was 0.285.

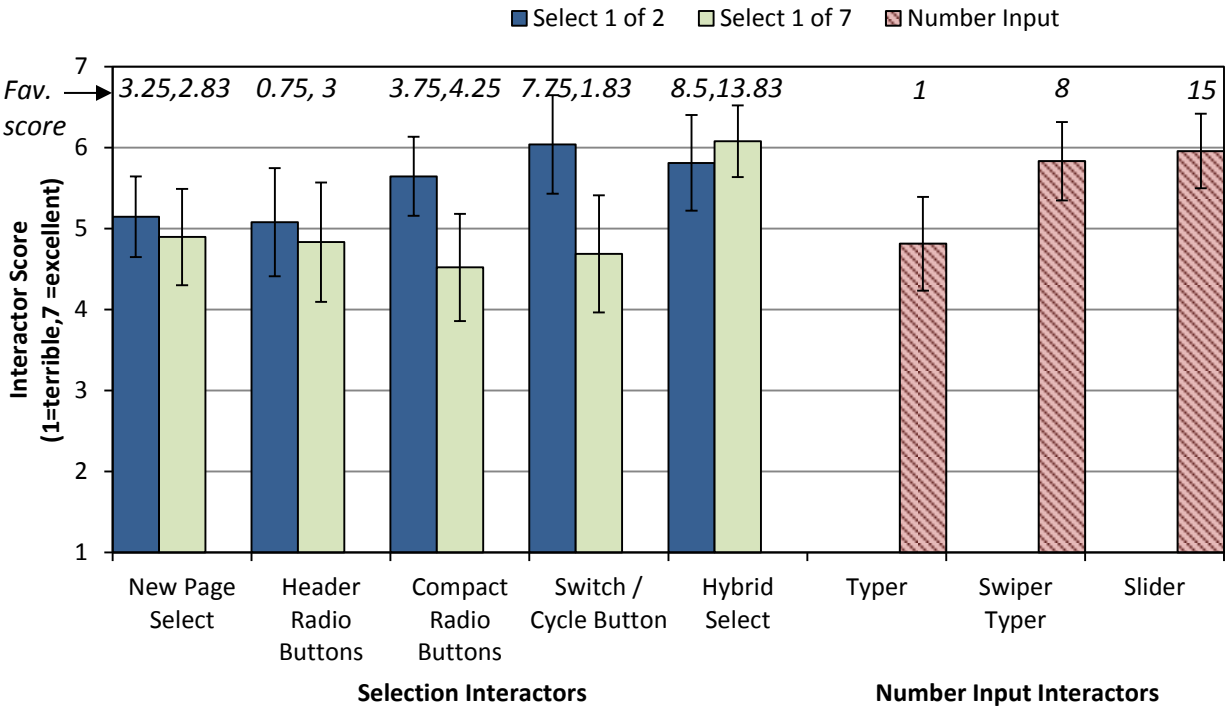


Figure 5-5. Interactor and fractional favorite scores for Gen-Loop interface interactors. The bars show the average score that participants gave each interactor on a 7-point scale with 95% CI error bars. The numbers in the top row of the chart denote the fractional favorite score of each interactor.

For the two selection tasks (1-of-2 and 1-of-7) on the Gen-Loop interface, 4 participants chose the same interactor, 15 participants chose different interactors, and 5 participants had ties for at least some of the same interactors.

Some of the Selection interactors were essentially the same between the Gen-List and Gen-Loop interfaces, with the same gestures used to select a new value. Between the Gen-List and Gen-Loop interfaces on the Select-1-of-2 tasks, 7 participants chose the same interactor, 12 participants chose different interactors, and 5 participants had ties for at least some of the same interactors. For the Select-1-of-7 tasks, 8 participants chose the same interactor, 10 participants chose different interactors, and 6 participants had ties for at least some of the same interactors between the Gen-Loop and Gen-List interfaces.

5.4.3 Interface Preference Results

Participants were asked to rank the interfaces according to preference at seven points during the two phases of the study. For analysis, each separate ranking can be treated as a binomial experiment where a "success" is defined as both automatically-generated interfaces (Gen-List and Gen-Loop) being ranked higher than both of the manufacturer-created interfaces (Mfr-Physical and Mfr-Item). These results for the two groups are shown in Table 5-5.

Table 5-5. Statistical tests of rankings of generated vs. manufacturer interfaces. This table shows the exact binomial probabilities of both of the automatically-generated interfaces being ranked higher than both of the manufacturer-created interfaces for both the blind and blindfolded groups.

When ranking occurred	Prob(success) by chance	*k / n* blind	Sig. (p) blind	*k / n* blindfolded	Sig. (p) blindfolded
Phase 1 after trying alphabet training interface	1/3	3 / 12	0.819	6 / 12	0.177
Phase 2 after trying training tasks on all interfaces	1/3	7 / 12	0.066	10 / 12	< 0.001
After trying all four thermostat interfaces the first time	1/6	12 / 12	< 0.001	11 / 12	< 0.001
After trying all four thermostat interfaces the second time	1/6	11 / 12	< 0.001	12 / 12	< 0.001
After trying all four copier interfaces the second time	1/6	10 / 12	< 0.001	10 / 12	< 0.001
After trying all four copier interfaces the second time	1/6	10 / 12	< 0.001	11 / 12	< 0.001
At the end of the experiment comparing all three iPod interfaces	1/3	12 / 12	< 0.001	12 / 12	< 0.001

Where *k* is the number of successes and *n* is the number of participants.

The fractional rankings for the interfaces at the different time points are shown in Figure 5-6. Non-parametric one-sided Wilcoxon signed rank tests on the participants' rankings at each time point showed that the Gen-Loop interface was favored over the Gen-List interface the first time the copier tasks were done (Blind group: z=1.67, p=0.048; Blindfolded group: z=2.61, p=0.005), the second time the thermostat tasks were done (Blind group: z=1.76, p=0.039; Blindfolded group: z=2.53, p=0.006), and at the end of the experiment for the blindfolded group only (z=2.02, p=0.022). Non-parametric one-sided Wilcoxon signed rank tests also showed that the Mfr-Item interfaces were ranked better than Mfr-Physical interfaces (with α=0.05) except for the Blindfolded group when ranking the copier interfaces the second time (z=1.63, p=0.052).

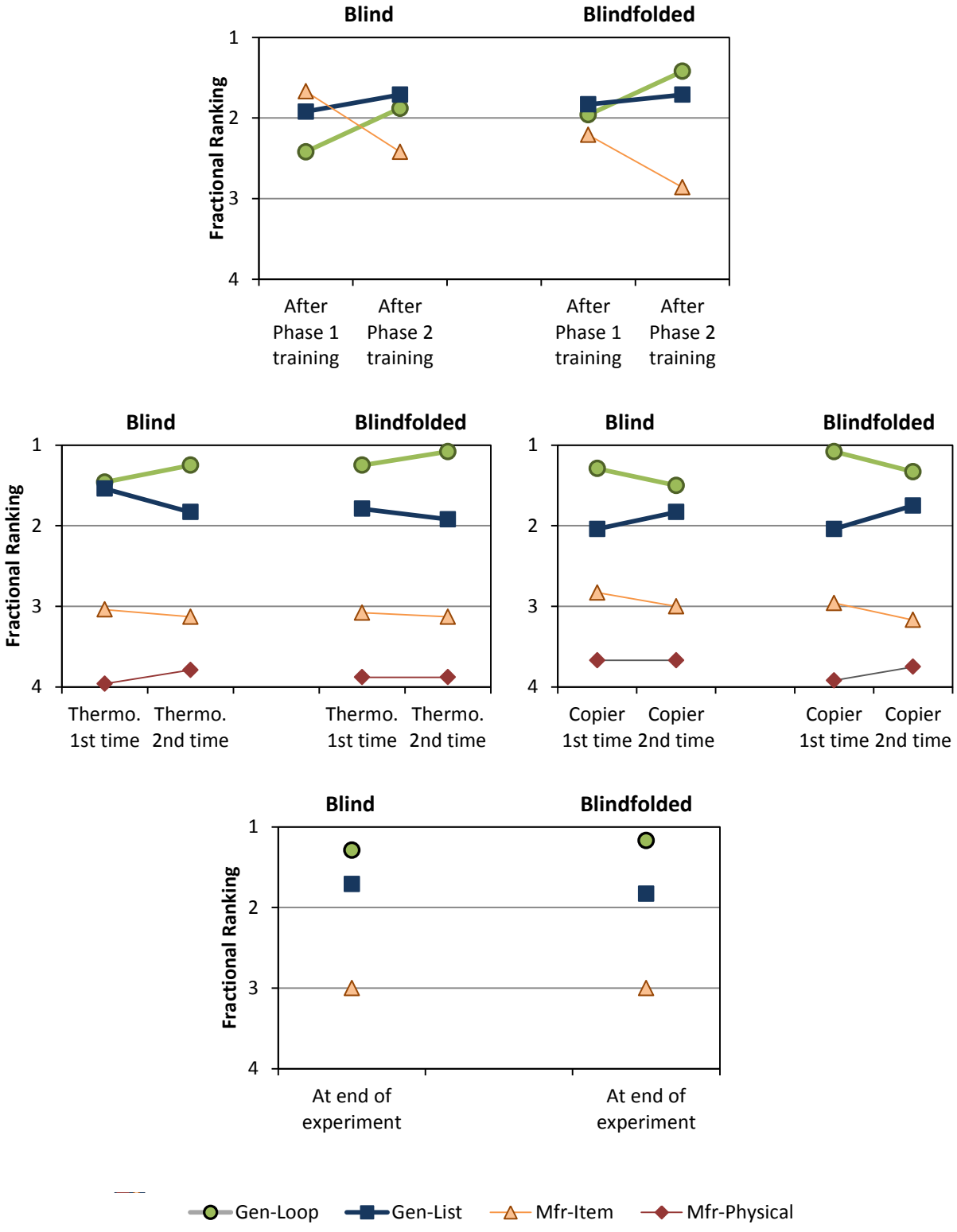


Figure 5-6. Fractional rankings of the three or four interfaces at different points by group.

Participants' rankings of the two auto-generated interfaces (Gen-List and Gen-Loop) were further compared using a General Estimating Equations analysis with a cumulative multinomial logit link function. The significance of each factor in the full factorial GEE model is shown in Table 5-6 with two groups (blind & blindfolded), two interfaces (Gen-List & Gen-Loop), and four measurement occasions when particpants were trying tasks on the copier and thermostat. The Gen-Loop interface had a median rank of 1 (IQR = 1), which was better than the Gen-Loop interface's median rank of 2 (IQR = 1).

Table 5-6. The significance of the factors in the full factorial GEE model of the rankings of the Gen-List and Gen-Loop interfaces.

Source	Wald Chi-Square	df	Significance (p)
Group	0.001	1	0.971
Interface	7.969	2	0.005
Occasion	14.357	3	0.002
Group × Interface	0.584	1	0.445
Group × Occasion	5.131	3	0.162
Interface × Occasion	6.986	3	0.072
Group × Interface × Occasion	0.995	3	0.802

5.4.4 Phase 2 Results

A number of different measures were collected as participants performed the various tasks. Success/failure, successful task time, and Single Ease Question (SEQ) responses, were collected for each task. The System Usability Scale (SUS) questionnaire was verbally administered after participants had completed all of the tasks on a particular interface. The Spearman's rank correlations between these measures were significant ($p < 0.001$ for all correlations) and are shown in Table 5-7.

Table 5-7. Spearman's rank correlation coefficients for the four Phase 2 study measures

	Rank-transformed performance [4]	**Success/fail** [4]	**SEQ** [3]	**SUS** [3]
Rank-transformed performance [4]	1			
Success/fail [4]	-0.902	1		
SEQ [3]	-0.760	0.720	1	
SUS [3]	-0.616	0.613	0.810	1

[3] Measures were collected for the three iPod interfaces (Mfr-Item, Gen-Loop, & Gen-List).

[4] Measures were collected for all four interfaces (Mfr-Physical, Mfr-Item, Gen-Loop, & Gen-List).

Each measure was analyzed separately for significance of the Interface factor and any Interface-factor interactions using methods appropriate for each type of data.

The success/fail data was binary data and was analyzed using Generalized Estimating Equations (GEE) with a binomial logit link function. GEE is a marginal model (also known as a population-averaged model) that extends the standard logistic regression model from the Generalized Linear Model (GLM) approach to correlated measures. The binomial GEE technique allows for correct inferences of repeated measures binomial data (J. Lee, Herzog, Meade, Webb, & Brandon, 2007).

The performance data was rank transformed so that the timing data and success/fail data were on a comparable scale. The rank-transformed performance data was then analyzed using the Repeated Measures ANOVA on partitioned data, which has been a suggested non-parametric-like analysis of non-normal data (Conover & Iman, 1981). The data was partitioned by device because such analyses may be confounded by non-null interaction effects (Toothaker & Newman, 1994) .

The Single Ease Question (SEQ) results in categorical data with integer responses ranging from 1 – 7. Generalized Estimating Equations with a cumulative multinomial logit link function was used to analyze the data because ordinal data is non-normal data that should not be treated as interval data without good reason.

The System Usability Scale is a well-studied Likert-scale. It is typically analyzed using typical normal-distribution-based statistical techniques (Sauro & Lewis, 2012). For this study, repeated measures ANOVA was used.

The results and statistical analyses are given in the sections below.

5.4.4.1 Success/Failure data

The success/failure data was analyzed using generalized estimating equations (GEE) with a binomial logit link function. A full factorial model with Group (2) × Device (2) × Interface (4) × Replication (2) was run and then the most non-significant terms were iteratively removed from subsequent GEE models until a parsimonious model was found with the lowest Corrected Quasi-likelihood Information Criterion (final model QICC = 1656.137). The terms of the model and significance tests of their effects is shown in Table 5-8.

Table 5-8. Factors in the final GEE model of the success/fail data and their significance.

Source	Wald Chi-Square	df	Significance (p)
(Intercept)	1.516	1	0.218
Group	9.991	1	0.002
Device	10.589	1	0.001
Interface	214.503	3	< 0.001
Replication	18.593	1	< 0.001
Group × Device	2.460	1	0.117
Group × Interface	18.918	3	<0.001
Device × Interface	24.228	3	< 0.001

All of the main effects were significant at a 0.05 α-level. The Group, Device, and Interface factors were also involved in two-way interactions. Pairwise post hoc tests were conducted on the significant interactions using the sequential Sidak procedure.

The Group × Interface interaction (see Figure 5-7) was significant. Between the blind and blindfolded groups on the Mfr-Item interfaces, blind participants had a success rate 0.44 points

(0.20 – 0.67 95% Wald difference CI) higher than that of the blindfolded participants ($p < 0.001$). There were no other significant differences between the groups on particular interfaces. For both groups, the Mfr-Physical and Mfr-Item interfaces had significantly lower success rates than the Gen-List and Gen-Loop interfaces (both with $p < 0.001$). For the blindfolded participants, the Gen-List interfaces had a success rate that was 0.60 points (0.43 – 0.77 difference CI) higher than on the Mfr-Item interfaces. For blind participants, the Gen-List interfaces had a success rate that was 0.36 points (0.21 – 0.50 difference CI) higher than on the Mfr-Item interfaces. Differences between success rates on the Gen-List and Gen-Loop interfaces within each group were not significant.

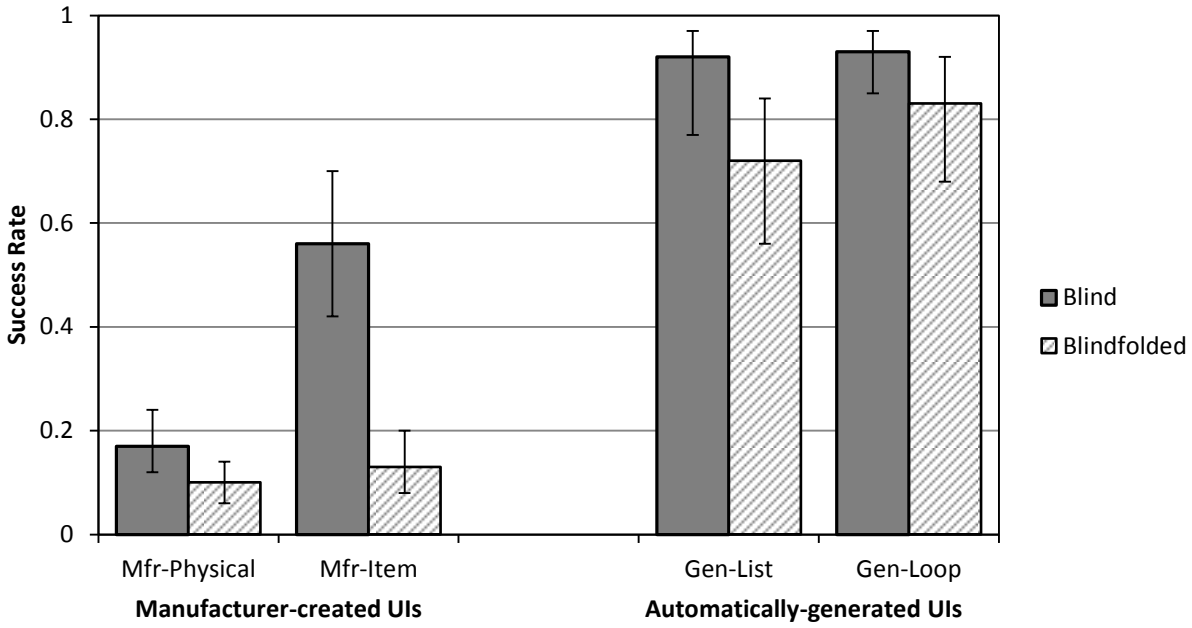


Figure 5-7. The estimated marginal mean success rates for the Group × Interface interaction. Error bars show 95% Wald confidence intervals for the marginal estimates.

The Device × Interface interaction (see Figure 5-8) was significant. With both devices, participants were more successful on the Gen-List and Gen-Loop interfaces than the Mfr-Physical and Mfr-Item interfaces. On the copier, the Gen-List interface had a success rate 0.42 (0.22 – 0.63 95% Wald difference CI) points greater than on the Mfr-Item interface ($p < 0.001$). Similarly, participants using the

thermostat were successful 0.61 (0.40 – 0.81 difference CI) points greater when using the Gen-List interface than when using the Mfr-Item interface ($p < 0.001$). Differences between the Gen-List and Gen-Loop interfaces on particular devices were not significant.

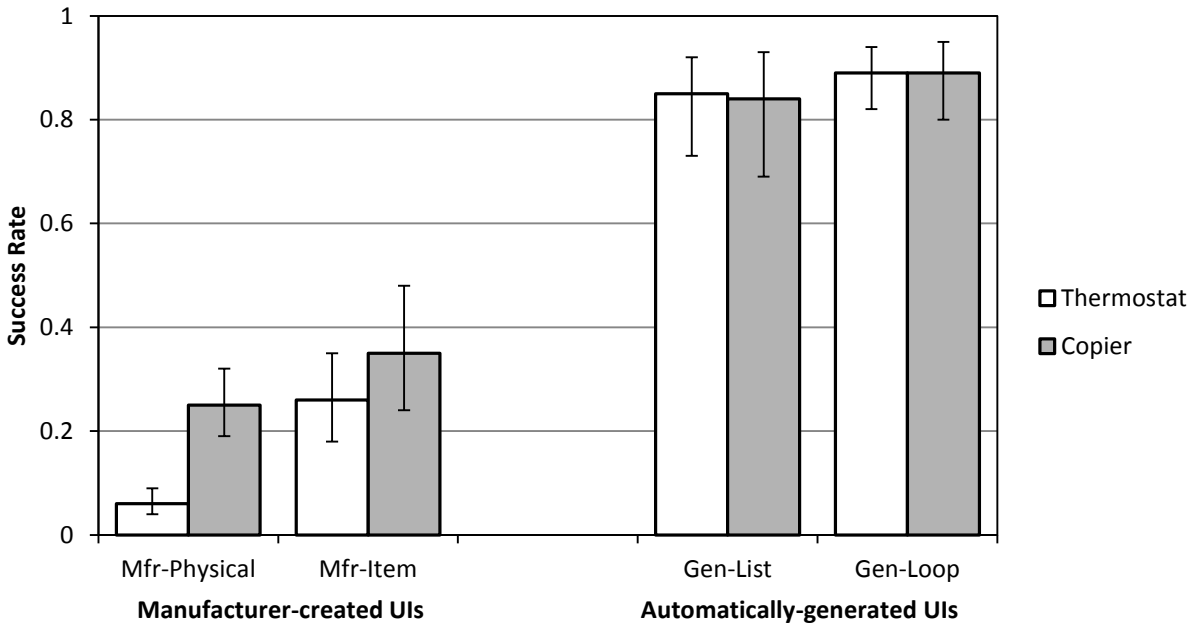


Figure 5-8. The estimated marginal mean success rates for the Device × Interface interaction. Error bars show 95% Wald confidence intervals for the marginal estimates.

5.4.4.2 Performance data

The success/fail data is folded into the performance data, because task times were only recorded for successful tasks. The statistical results of the success/fail data showed that people were more successful with the two automatically-generated interfaces (Gen-List and Gen-Loop) than on the manufacturer-created interface (Mfr-Physical and Mfr-Item). For this reason, performance differences were only tested between the Gen-List and Gen-Loop interfaces. The data from the two devices were partitioned and analyzed separately. The distribution of the copier data is shown in Figure 5-9 and the distribution of the thermostat data is shown in Figure 5-10.

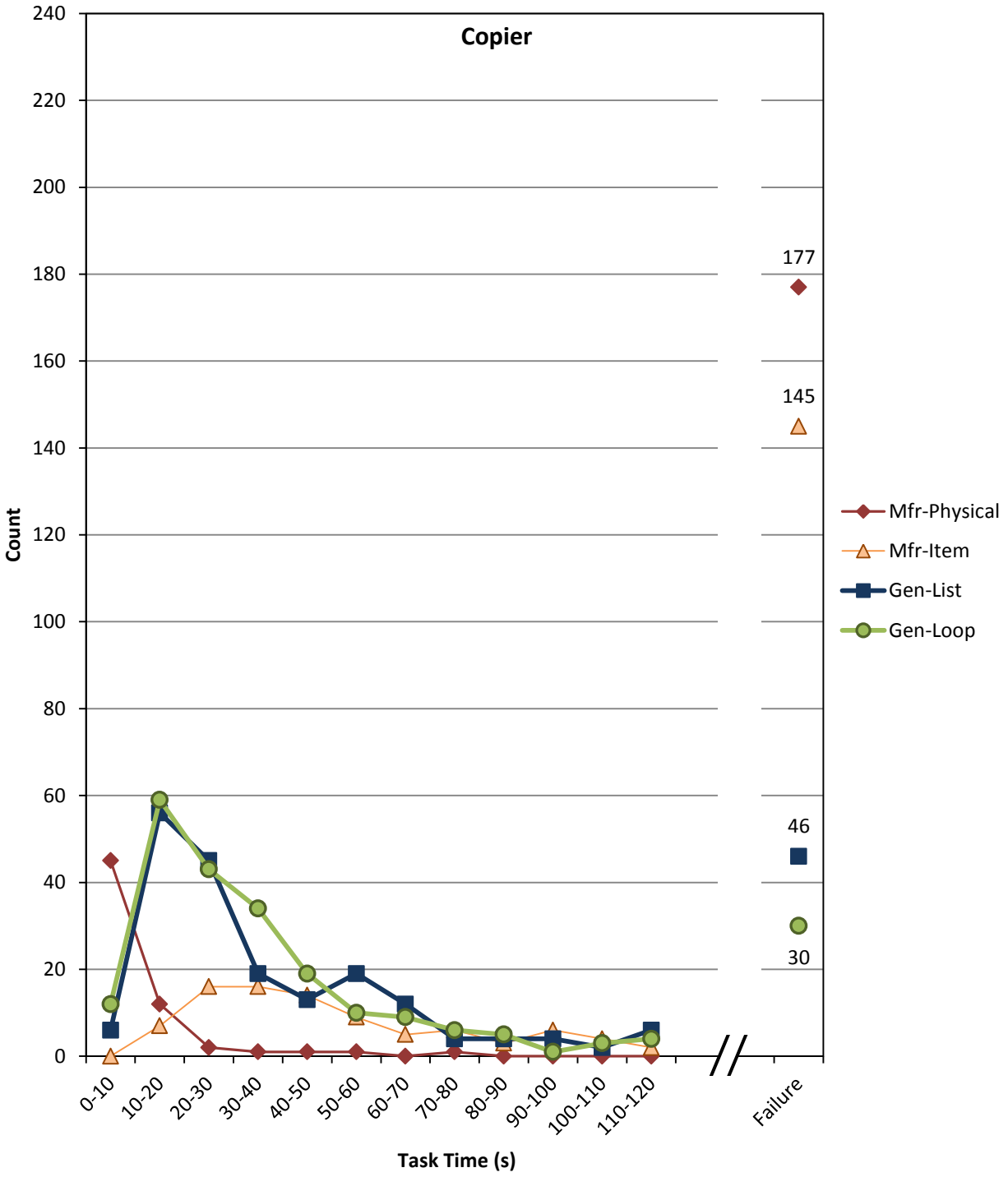


Figure 5-9. A histogram showing the distributions of successful task times and the number of failures for the copier device.

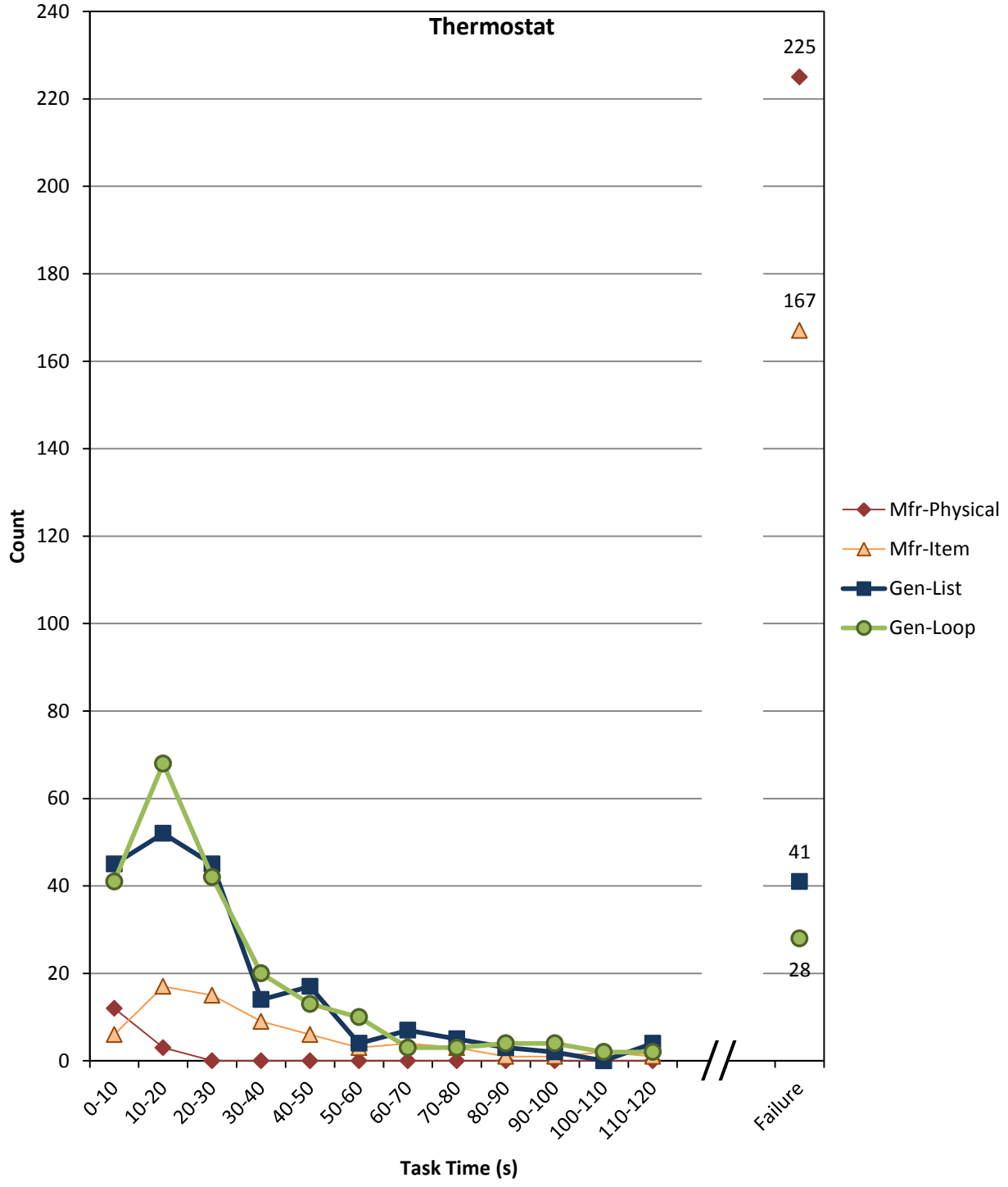


Figure 5-10. A histogram showing the distributions of successful task times and the number of failures for the thermostat device.

The cumulative distributions of the performance data in Figure 5-11 and Figure 5-12 show more clearly the trends between the interfaces.

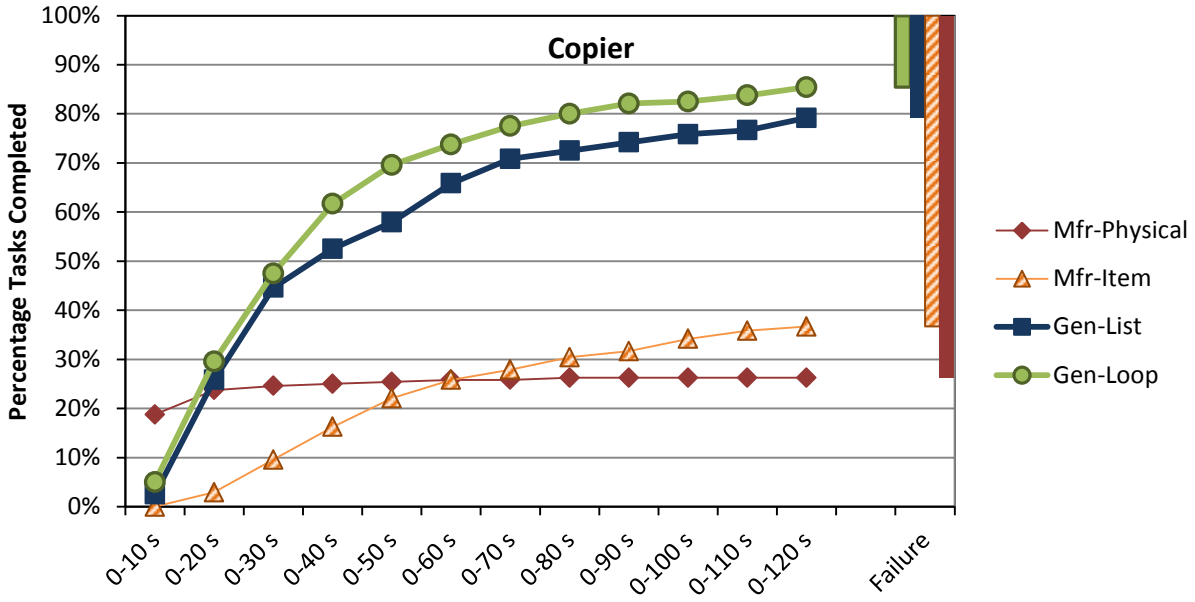


Figure 5-11. Cumulative distribution of the tasks completed on the copier.

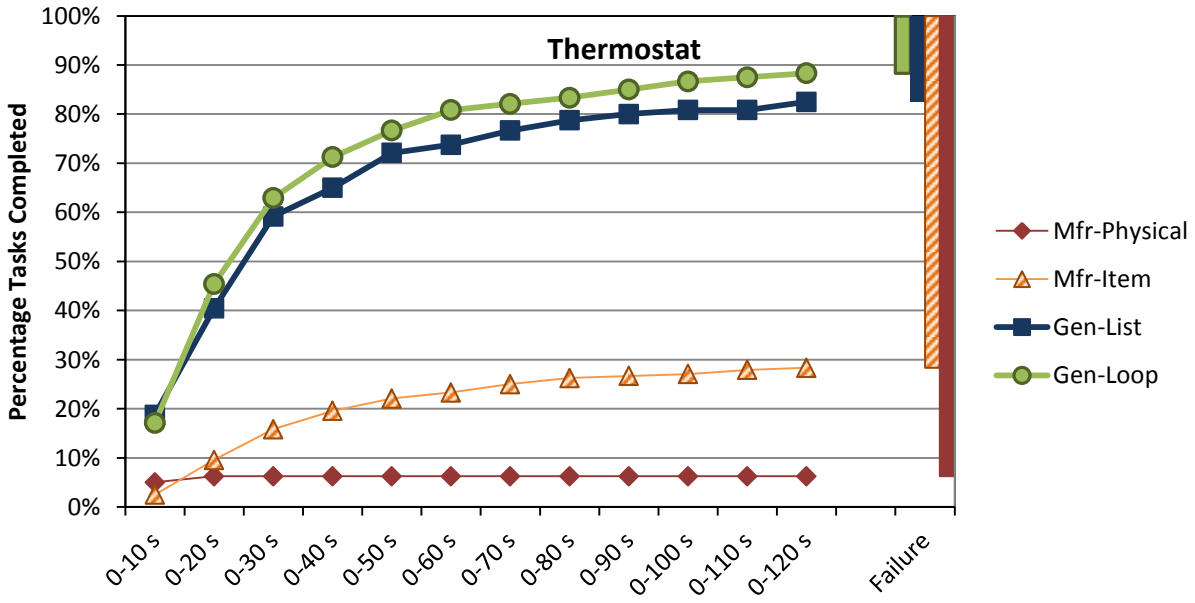


Figure 5-12. Cumulative distribution of the tasks completed on the thermostat.

To test the differences in performance, the performance time and failure data were converted to a comparable scale, which was done by rank transforming all of the data. Through the rank transform, all failures were given tie rankings. Repeated measures ANOVA was then used on the ranked performance data to analyze the two devices separately (full RM-ANOVA tables are included in Appendix G).

For the copier, the Interface main effect (F=3.346, df=1) was not significant (p=0.082). The only significant interaction involving the Interface factor was the Interface × Task (F=3.382, df=4, p=0.013) interaction, which is plotted in Figure 5-13. Sequential simple effect Sidak post hoc tests indicated that participants performed significantly better on the Gen-Loop interface on Tasks #4 (mean rank difference = 52.457, 17.198 – 87.715 95% difference CI, p = 0.005) and #5 (mean rank difference = 50.989, 11.045 – 90.934 95% difference CI, p = 0.015) on the copier as compared to corresponding tasks on the Gen-List interface. The differences between the interfaces on the other three tasks were not significant.

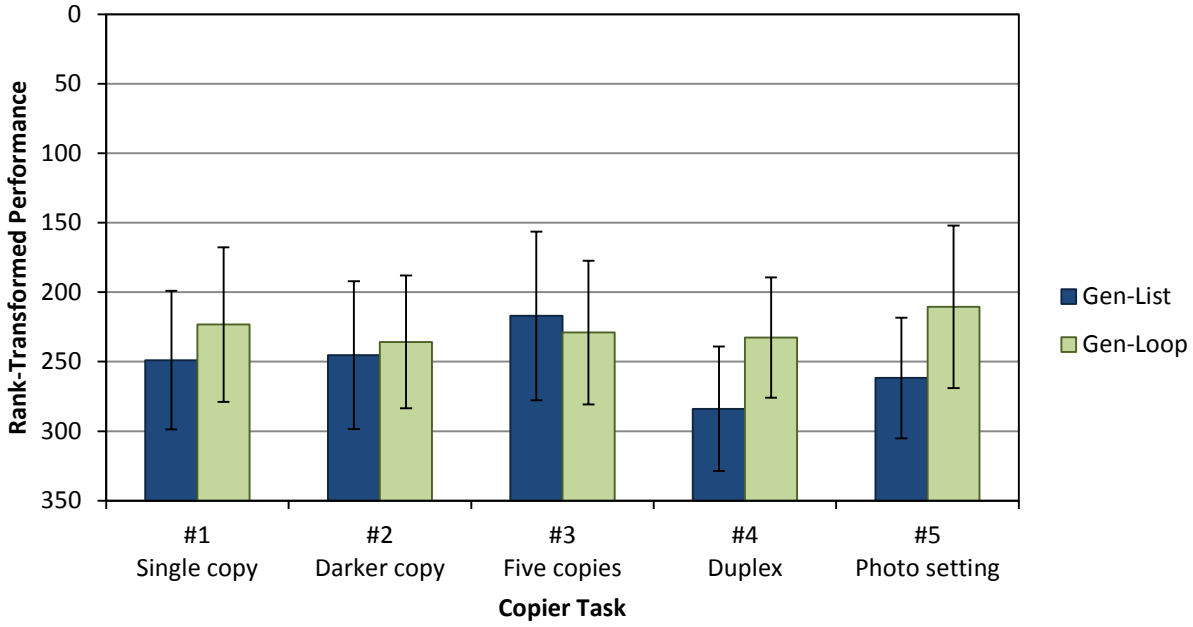


Figure 5-13. The Interface × Task interaction showing the marginal means of the rank-transformed performance data. Note that the scale is reversed on this chart and that lower ranking values are better (i.e., quicker and more successful). The error bars show 95% normal-distribution confidence intervals for rank-transformed performance.

For the thermostat, the Interface main effect (F=1.149, df=1) was not significant (p=0.295). No interactions involving the Interface factor were significant.

5.4.4.3 Single Ease Question (SEQ) data

The single ease question (SEQ) on a 7-point scale anchored with 1= "very difficult" and 7 = "very easy" was asked after each task. In order to figure out the significant factors on the SEQ data, a full factorial Type III Sum of Squares GEE model on the Group (2), Device (2), Interface (3), and Replication (2) factors was run on the SEQ data with a multinomial cumulative logit link function. Only the three iPod interfaces (Mfr-Item, Gen-List, and Gen-Loop) were included in the analysis because the Mfr-Physical interface was low scoring and did not have complete factorial data for the replication. The GEE model was run iteratively removing 4-, 3-, and 2-way interactions that were obviously non-significant one at a time. The significance data of the resulting final model is shown in Table 5-9.

Table 5-9. Factors in the final GEE model of the SEQ data and their significance. Only the Mfr-Item, Gen-List, and Gen-Loop interfaces were included in this analysis because they had complete factorial data.

Source	Wald Chi-Square	df	Significance (p)
Group	6.874	1	0.009
Device	0.364	1	0.546
Interface	98.308	2	< 0.001
Replication	19.532	1	< 0.001
Group × Interface	7.395	2	0.025
Device × Interface	9.626	2	0.025

As a post hoc comparison of the Gen-List and Gen-Loop interfaces, the data was partitioned to remove the Mfr-Item interface and the full factorial GEE multinomial cumulative logit model was run again with iterative removal of non-significant effects. The factors and significance data of the resulting final model is shown in Table 5-10.

Table 5-10. Factors in the final GEE model of the SEQ data for the two auto-generated interfaces and their significance.

Source	Wald Chi-Square	df	Significance (p)
Group	1.463	1	0.227
Device	0.759	1	0.384
Interface	6.290	1	0.012
Replication	21.435	1	< 0.001
Group × Replication	3.722	1	0.054

On the SEQ, participants rated the tasks on the Gen-Loop interface as easier (Mdn=7, IQR=1) than the Gen-List interface (Mdn=6, IQR=2). Figure 5-14 shows the distribution of the SEQ rankings for the Gen-List and Gen-Loop interfaces.

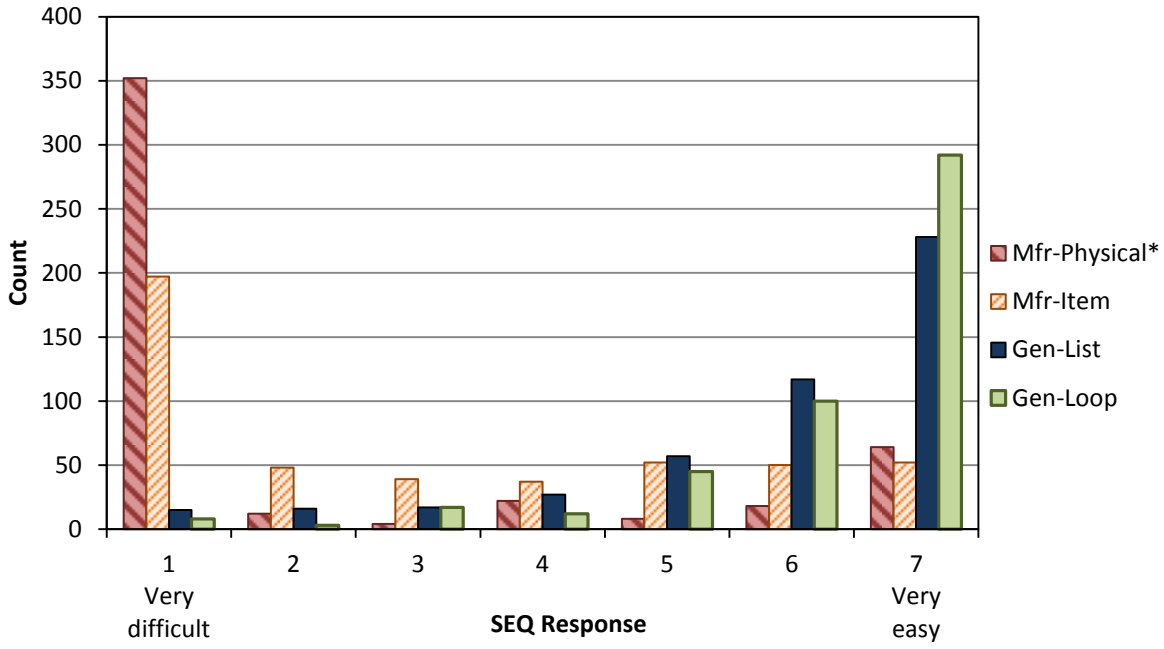


Figure 5-14. A histogram of the SEQ responses for all of the tasks.

* The frequency for the Mfr-Physical interface is scaled to that of the other interfaces, since participants did not give SEQ scores for the replication.

5.4.4.4 System Usability Scale (SUS) data

The 10-question Likert SUS questionnaire was answered by participants after completing all five tasks of each replication with the three iPod interfaces (Mfr-Item, Gen-List, and Gen-Loop). A Cronbach's alpha reliability score was calculated for each of the 12 times the SUS was administered. Agreeing with the literature (e.g., Bangor, Kortum, & Miller, 2008; Lewis & Sauro, 2009), the individual SUS scale items were highly consistent for each administration in this experiment; the average Cronbach's alpha for the items in the SUS in this experiment was 0.931 with values ranging from 0.865 – 0.973. Because of this high reliability, the individual items were transformed and summed, as is typical with SUS measurements, for the subsequent comparative analysis.

The SUS data was analyzed as a Group (2) × Device (2) × Interface (3) repeated measures ANOVA. The main effect of Interface was significant ($F(2)=77.800$, $p < 0.001$), as was the Interface × Replication interaction ($F(1.525)=5.148$, $p=0.018$ with Greenhouse-Geisser correction for significant asphericity of the data), which is shown in Figure 5-15.

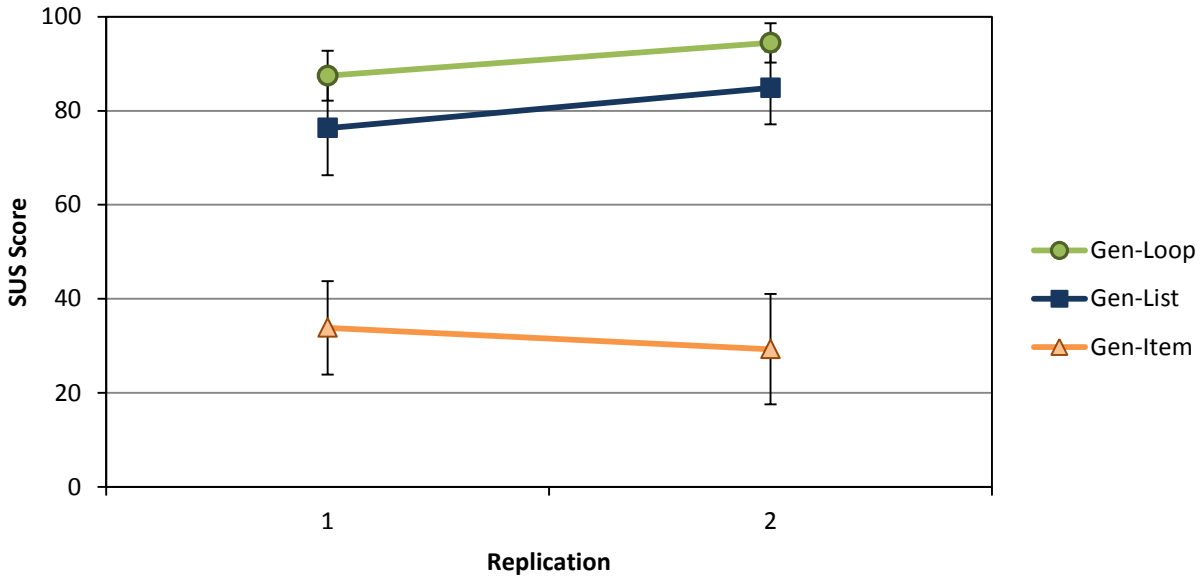


Figure 5-15. The Interface × Replication interaction showing the marginal means of the SUS scores. The error bars show 95% normal-distribution confidence intervals for the marginal interaction means.

Sequential simple effect post hoc tests with the sequential Sidak correction were performed on the Interface × Replication interaction. These tests indicated that participants scored the Gen-List interface 8.56 points higher on the replication (p=0.015), the Gen-Loop interface 7.00 points higher on the replication (p=0.001), and no significant SUS score change on the Mfr-Item interface with the replication (p=0.273). Both the Gen-List and Gen-Loop interfaces significantly outscored the Mfr-Item interface at both time points ($p < 0.001$). The SUS scores for the Gen-List and Gen-Loop interfaces for the first block were not significantly different (p=0.059), but the Gen-Loop did score 9.58 points higher (0.39 – 18.78 95% difference CI) than the Gen-List interface with the replication (p=0.039).

5.5. Discussion

When choosing their favorite interactors, participants had strong preferences for the Gen-List interactors that were designed for this experiment over "pre-existing" interactors. Particularly notable was the dislike of the Picker interactor (which is what is presented to people using the VoiceOver screen reader whenever a person encounters a dropdown list, e.g., a `<select>` element in HTML). One blind participant, who had used pickers before on her own device, said bluntly that, "Pickers could go to hell." Nobody in the experiment chose the Keyboard interactor, which was closely modeled after the iOS keyboard mode that was presented for numeric entry (e.g., entry into a `<input type="number">` element in HTML). Both the Picker and Keyboard interactors were examples of layered interfaces—where users essentially have to use a screen reader interface layer that is reading (and interacting with) a graphical user interface that was specifically designed for mainstream users with vision. Having automatically generated interfaces can eliminate this layering of interfaces, because the interactors can be designed with users' primary modalities in mind. For this experiment, the "new" interactors were designed specifically with speech output and touchscreen gesture manipulation in mind; any visual representations were secondary and mostly included to make it easier for the sighted researchers to observe.

While participants liked the "new" interactors, several of the blind participants wished that additional gestures were supported. For example, two blind participants use the VoiceOver drag-and-tap-with-a-second-finger gesture on their own iPhone devices to make selections rather than the double-tap that was required by the research system. Others had difficulty with the gestures required by both VoiceOver and the research systems and would rather have substituted their own gestures—the blind participant with tremor suggested having a dedicated area on the touchscreen or button on the device for activation rather than double-tapping anywhere on the screen. Such user preferences could be supported by personalized, automatically-generated interfaces.

The interactors that participants chose and the reasons that they reported for making the choices did not always seem optimal to the researchers. For example, one blindfolded participant chose a Hybrid Select interactor for the Select-1-of-2 Gen-List task (which offers the person the flexibility of swiping vertically to switch between values or double-tapping to open a menu), but then chose a Cycle Button interactor for the Select-1-of-7 task (which requires a person to double-tap multiple times to cycle through all of the values). This participant reported some frustration when he actually used Gen-List interfaces, because he would sometimes overshoot the desired selection and have to double-tap through the entire list all over again. Other participants occasionally wished that the Gen-List or Gen-Loop interfaces were a little different when they were trying the actual study tasks rather than the preference elicitation tasks. For example, a blind participant who chose radio button interactors in both the Gen-List and Gen-Loop interface reported that it was awkward to have to switch pages so much when doing the copier or thermostat tasks (which is the tradeoff with choosing radio buttons over menu-based or other interactors). This suggests that a different approach to preference elicitation may be helpful. Users who know ahead of time that they are choosing interactors for personalized interfaces might choose differently. Users may also pick better interactors if they were to do a first pass to screen out the interactors that they strongly dislike, and then try more comparisons with more realistic interfaces with the finalist interactors.

Even with some participant's interactor choices being suboptimal, the two automatically-generated interfaces tested better than the manufacturer-created interfaces for all measures on the copier and thermostat devices. The magnitude of the difference was greater than expected. Before conducting the experiment, it was expected that at least the copier's accessible web-based interface (i.e., the copier's Mfr-Item interface designed specifically for blind screen reader users) would have been a much closer match to the two automatically-generated interfaces, because it had been hand-crafted specifically for people who are blind and using screen readers. One blind participant said that copier's Mfr-Item interface was "100-percent do-able," and that she probably would have scored it more highly on the usability questionnaire (SUS) if she had not just tried a superior interface beforehand (in her case, the Gen-List interface). For participants in this experiment, the automatically-generated interfaces were consistently better than the manufacturer-created ones.

The results of the sub-study of the performance and preference differences between the two automatically-generated Gen-List and Gen-Loop) interfaces are not as clear, however. The subjective measures (ranking, SEQ, and SUS) showed some limited evidence that the Gen-Loop interface may be better than the Gen-List interface, but the success/failure and performance data did not. The fact that the Gen-Loop interface was perceived as generally preferred to or better than the Gen-List interface might also be a bias related to the good-subject effect (A. L. Nichols & Maner, 2008) The Gen-Loop interface was obviously the most different interface to people who were blind, so "good" participants might have felt that the study's aim was to show that the Gen-Loop interface was better and "good" participants might respond in such a way to support that perceived hypothesis. It would be fair to say that participants had different and sometimes changing preferences between the Gen-List and Gen-Loop interfaces (in particular, see the rankings in Figure 5-6). One blind participant disliked the Gen-Loop interface concept in general because he felt it was confusing and unintuitive. Other participants liked how the Gen-Loop interface felt logical and efficient. Some participants did not have much of an opinion either way because

they used both the Gen-List and Gen-Loop interfaces with swiping navigation gestures. If participants chose particular interactors, then they could potentially have exactly the same user experience when swiping to navigate with both Gen-List and Gen-Loop interfaces. Many participants did not like the Gen-Loop interface at first with only limited exposure and use, but later after becoming accustomed to the interface style, many participants' preferences changed. One participant had his own hypothesis and said that the Gen-Loop interface violated the two ways that blind people have interacted with user interfaces so far: (1) navigation using arrow keys and tabbing or (2) scanning through the interface as with a visual magnifier or dragging on a touchscreen. He said that the Gen-Loop interface was good, but that he did not like it at first because it required him to break the force of habit: "Forget all that. [The Loop interface] might be a better, more efficient way to do this."

This lack of a clear difference between Gen-List and Gen-Loop interfaces actually lends more support to model-based generation of user interfaces. Model-based, automatic interface generation can support people's preferences, even for very different layouts and interaction styles. People who prefer and perhaps better comprehend a more typical, linear interface could have automatically-generated interfaces that look and behave like a Gen-List interface. Other people might prefer to use a Gen-Loop interface because it is intuitive and more efficient for them.

While group differences were not the focus of the study, the data supported the expectation that people who were blind perform better with blind-specific interfaces than people who were blindfolded and had no prior experience with those interfaces or techniques and strategies that are used by people who are blind. It was actually remarkable that the blindfolded participants did as well as they did. VoiceOver and screen readers in general have a steep learning curve that can be particularly difficult for elders and others who are not technically savvy (K. M. Fountaine, personal communication, August 31, 2015). The two-way Group × Interface interaction of the success/fail data (plotted in Figure 5-7) could be interpreted along with the flexibility and experience of participants. Blind participants were more flexible and

experienced and thus did better on the Mfr-Item interface than the blindfolded participants. However, the automatically-generated interfaces studied here were very consistent with relatively few interactors and gestures, which seemed to make it easier for even novice blindfolded users to experience success.

5.6. Conclusion

The interfaces that were automatically generated using each participant's preferred interactors were statistically better on all performance and usability measures than the manufacturer-created interfaces. Participants also preferred the auto-generated interfaces over the other interfaces once they had finished training and started performing the actual tasks. These results are notable because the model and interface generators were both relatively simple. Of the two auto-generated interfaces, there was some evidence that participants preferred the novel Loop interface, but no performance difference was found.

This study validates the ability of the FIN-USI modeling approach to automatically create user interfaces that are more preferred and perform better than manufacturer-created interfaces, including one design specifically for people who are blind. It also validates the sufficiency of even simple interface generators created using the model. In the future, model-based auto-generation of interfaces could support a wide range of user needs and preferences, where people could choose the type of interface they want and from what components interfaces are built.

Chapter 6.
Case Studies

The impetus for adding these case studies (or supplemental probes) was a participant who was initially enrolled in the main study comparing different automatically-generated and manufacturer-created interfaces. This participant had no reported disabilities that would affect his use of mobile devices, but he described himself as very bad with technology and was worried that he might "ruin" the experiment. He was excluded from the main study because of how much time it took him to get through the first phase of it. It seemed that neither the main study's interface types nor any of the interactors which might compose those automatically generated interfaces were a good fit for him. It was felt that a simplified screen reader interface that used buttons instead of gestures might be a better fit for this participant and potentially better for two other participants who had completed the main study but had some difficulties that could potentially be solved with non-gesture interfaces.

This chapter describes case studies done to compare self-voicing auto-generated button-like interfaces, auto-generated gesture-based interfaces, and manufacturer-created interfaces use with a keyboard and screen reader, which is the current state of the art.

6.1. Participants

Before the case studies, we consulted for a short time with two participants who were completing the main study. They tried a few tasks on the prototypes being considered for this probe as they were being developed, and they gave feedback about their experience. Their comments were used to make improvements to the systems before testing with the three case study participants.

Three participants were included in the case study. Participant #1, a 60 year old male who had no reported disabilities, had been screened out of the main comparison study because it took an inordinate

amount of time for him to complete the first phase, and it was felt that it would be too much of a burden on the participant to try to finish the main study. He was dropped from the study before any thought of doing this extra probe occurred. In this case study (which includes the gesture-based interfaces from the main study as well as non-gesture interfaces), it turned out that he was able to use the gesture interfaces to some degree, so dropping him from the main study may have been premature. His performance in this study however appeared to follow a similar pattern to the other participants (with auto-generated interfaces being preferred over manufacturer-created ones), so his removal from the main study would appear to not have had any effect on the main study outcome.

Participant #2 was a 49 year old male who was blind and had completed the main study. He was included in this study because he had a tremor which had made it difficult to reliably make many of the gestures that were required for operation of the main study's gestured-based interfaces.

Participant #3 was a sighted, 56 year old female who had also completed the main study. She was included because she had fingers that were crooked (which made it difficult to do some gestures) and because she also performed poorly compared to the other participants in the main experiment.

Participants were paid for travel expenses and at a rate of $15 per hour for their participation.

6.2. Instrumentation

During the experiment, participants frequently interacted with an iPod Touch (Model ME643LL/A, Apple Inc., Cupertino, Calif) running iOS 8.2. The iPod Touch was connected to a computer and a mixer for recording and to a speaker to allow for louder and clearer audio. The iPod Touch was placed in two different modified OtterBox (Fort Collins, Colo.) Defender Series cases, both of which had the screen protector removed for better responsiveness. The case used for the Gen-List and Gen-Loop interfaces additionally had a 1.9-mm-thick plastic piece added to block the top 4.3 mm of the screen with the iOS clock, icons, and notifications bar. For the Mfr-Keyboard interface parts of the

experiment, participants used VoiceOver (a screen reader built into iOS) with a paired Bluetooth keyboard (Apple Wireless Keyboard Model A1314, Apple Inc.). Participants also interacted with a native iOS application for the smart thermostat device (Model CT80, Radio Thermostat Company of America, San Francisco, Calif.) and an accessible web-application for a multifunction copier device (Model X654de, Lexmark International, Inc., Lexington, Ky.). Participants used all of the interfaces while they were placed flat on a table to make it easier to video record their interactions.

6.3. Methods

The main purpose of the experiment was to compare users' performance, preference, and satisfaction with manufacturer-created interfaces used with a screen reader, which is the current state of accessible technology, and a number of different automatically generated interfaces. The two devices in the study (thermostat and copier) had their interfaces modeled using the FIN-USI model, which aims to be a simple model that can cover a broad range of applications and functionality. All four interface generators were basic: they used the FIN-USI model of the interfaces and then built an interface using the interactors that each participant preferred for each modeled input or output. There were no nuances or extra information included in the interface models or generators that could tune interfaces to particular tasks. In this experiment, five general interface types were tested, which for short are called the Mfr-Keyboard, Gen-List, Gen-Loop, Gen-Panel, and Gen-Button interfaces (see Table 6-1 and Figure 6-1).

Table 6-1. The interfaces tested in the case studies.

Interface (Device)	Source	Comments	Tested with participants
Mfr-Keyboard (Thermostat only)	Manufacturer created (new for the case studies)	Native iOS application that worked with VoiceOver (screen reader) and the keyboard, but was not specifically designed for accessibility. The interface failed two WCAG 2.0 provisions. • 2.4.2 "Focus Order" because some menus opened up in confusing locations earlier in the application's navigation order. • 2.4.6 "Headings and Labels" because labels for different elements were read as a single block rather than with each element.	#1, #2, #3
Mfr-Keyboard (Copier only)	Manufacturer created (new for the case studies)	A web-based interface used with VoiceOver and a keyboard. It was specifically designed to be accessible and usable to people using screen readers on computers. It used very simple links and form fields, and met all level-AA WCAG 2.0 provisions. It was specifically chosen for testing because it is an exemplar of accessibility among mainstream device manufacturers.	#1, #2, #3
Gen-List (Both devices)	Automatically generated (from main study)	Interface elements were generally laid out in a conventional list layout with one interactor or element per row. It used similar touchscreen gestures as VoiceOver.	#1
Gen-Loop (Both devices)	Automatically generated (from main study)	Novel interface layout and gestures with interactors and elements arranged around the edges of the screen in a clockwise direction. It was designed to be used by dragging around the edges of the screen (although it could also be used with many of the same gestures as the Gen-List interface) to quickly find elements.	#1, #2, #3
Gen-Panel (Both devices)	Automatically generated (new for the case studies)	A prototype interface with a raised 1.6-mm thick plastic panel with 5 holes in it, which delimited the onscreen buttons necessary for activation. The person would tap the touchscreen buttons through the holes to activate them.	#1, #2, #3
Gen-Button (Both devices)	Automatically generated (new for the case studies)	A prototype interface that had 5 physical push buttons on the touchscreen in the same arrangement as the Panel interface. The physical buttons were constructed of polydomes from a disassembled keyboard (1100 Series keyboard, Storm Interface, West Drayton, England) to provide a tactile press and thumb tacks to create a button top and the required conductive contact to activate the touchscreen.	#1, #2, #3

Figure 6-1. Photographs of Gen-List, Gen-Loop, Gen-Panel, and Gen-Button interfaces. All of the interfaces were on an iPod Touch PAUI device. Rulers in the figure are marked in cm.

As with the main study, the experiment was in two phases. In the first phase, the participant's preference for interactors was elicited. The second phase was a comparative study of the different interfaces with the automatically-generated ones using the participant's Phase 1 preferences. For the sighted participants (#1 and #3), the interfaces were either covered or they wore a blindfold so that they could not see the interfaces.

6.3.1 Phase 1: Preference elicitation

Phase 1 took two sessions for each participant. Each Phase 1 session started with setting the volume, text-to-speech rate, and double-tap acceptance time settings to comfortable levels. Participants were then trained on the basic operation of three interfaces of interest for the session where they navigated through and selected letters from the alphabet. Finally, for the last part of the Phase 1 sessions, participants compared different interactors to see which ones they liked best for particular tasks. On the first day (as part of the main study), they chose their favorite interactors for the Gen-List and Gen-Loop interfaces. On the second day (as part of the case study), they chose their favorite interactors for the Panel

interface. Those favorite Gen-Panel interface interactors were also used to build the Gen-Button interface because the interfaces were so similar.

In both Phase 1 sessions, participants kept comparing interfaces in a pairwise manner while completing simple tasks. Preference elicitation interfaces were constructed using the same two interactors of interest (for example, two dropdown menus or two sets of radio buttons). For each preference elicitation task, participants had to navigate past the first interactor on the screen to get to the second interactor, which they would need to manipulate. Participants would start by trying a random pair of interfaces with the same task and choose their favorite. Their favorite interface from the prior pair was then paired with a random challenger interface, and they were then to try and rank those two interfaces. This pairwise comparison process was repeated until the favorite interface of all of the available ones was chosen. Then the entire process was repeated for the next task and interface style. Participants started with the Gen-List interface and performed three sets of tasks in order: select-one-from-two tasks, followed by select-one-from-seven tasks, and then numeric entry tasks. They then repeated the three sets of tasks and interactor comparisons with the Gen-Loop interface. On the second day, participants performed the three sets of tasks on the Gen-Panel interface. Participants were also asked to score each interactor after their first use it on a 7-point scale with anchors of 1 = “terrible” and 7 = “excellent.”

6.3.2 Phase 2: Comparative interface testing

The main purpose of Phase 2 of the experiment was to gather performance, preference, and satisfaction data for comparing the various interfaces for the two devices. Participants #1 and #2 took two sessions to complete Phase 2 while Participant #3 was quick enough to complete all of the tasks in one session. Each Phase 2 session started with setting the volume, text-to-speech rate, and double-tap acceptance time settings to comfortable levels. Participants were also trained on the operation of the interfaces that they were to use. For training they did the same tasks from Phase 1 with their favorite interactors for each auto-generated interface. The interfaces were trained in random order.

In order to compare interfaces, participants started with a random device (either the copier or thermostat) and then tried five device-specific tasks on each of the interfaces in turn. The interfaces were presented in a random order with the Gen-Button and Gen-Panel interfaces being back to back. This was done so that the Gen-Panel or Gen-Button interface could be replicated after the other interface (i.e., Panel-Button-Panel or Button-Panel-Button orders). After each task, participants were verbally asked the Single Ease Question (SEQ; Sauro & Dumas, 2009). After finishing the five tasks on an interface, participants were verbally administered the System Usability Scale (SUS) questionnaire (Brooke, 1996), which was modified to have 7-point Likert-scale items (instead of 5-point scales). Participants were finally asked to rank and make comments on all of the interfaces after trying all of them for a given device.

6.3.3 Differences in methods between participants

The case study experimental methods differed slightly from participant to participant based on the particular participant and his or her needs.

Participant #2 had tremor, so the double-tap wiggle threshold (i.e., the maximum amount of finger movement allowed on the touchscreen during each tap of the double-tap) was doubled to 5 mm. For Participants #1 and #3, the double-tap wiggle threshold was 2.5 mm, which was the value used for the main experiment.

Participant #1 had difficulty in Phase 1 keeping track of which interface he was using and how he was supposed to interact with it. When he did the Phase 2 part of the study, he was trained on an interface just before he used it to complete the tasks. This contrasts with the order of training and performing tasks in Phase 2 for Participants #2 and #3. These two participants used an order similar to the main study, where they were first trained on all of the interfaces in random order and afterwards tried the tasks on the interfaces in a random order.

Participant #1 tried all five interfaces (Mfr-Keyboard, Gen-List, Gen-Loop, Gen-Panel, and Gen-Button), whereas Participants #2 and #3 did tasks on four interfaces. They did not try the Gen-List interface, because they had done it during the main study and the Gen-Loop interface had already been identified as their favorite interface.

6.4. Results

Two pilot participants tried just a few tasks on the Gen-Button and/or Gen-Panel interfaces while they were being developed. The first person to try a few tasks was blind and only tried the Gen-Panel interface, which was built first. She had difficulty with the tasks on the Gen-Panel interface compared to the gesture-based interfaces in the main study. She would sometimes try to double-tap (the activation gesture of VoiceOver) the "Go" button to activate an element— which actually did a double activation (of the element and then the first item on any subsequent screen). She also occasionally tried to swipe right and left (the navigation gesture on VoiceOver) on the "Go" button—which caused the focused element to activate. Her problems were part of the reason that the Gen-Button interface was developed: physical buttons would prevent somebody from attempting to make gestures on the touchscreen through the holes in the panel. The second participant was a blindfolded participant, who tried a few tasks on both the Gen-Button and Gen-Panel interfaces and did well. She reported that if those interfaces had been included in the main study that she would have preferred either of them to the gesture-based Gen-List and Gen-Loop interfaces.

All three participants in the case study remarked that the Panel interface seemed very sensitive; Participant #1 said it "had a hair trigger." This contrasted with the Button interface, which they all felt was much less sensitive and occasionally required a second button press to register.

Both participants #1 and #2 recognized that the Gen-Button and Gen-Panel interfaces were essentially the same. Participant #3 however said during the experimental session that she was confused

by the layout of the Gen-Button interface. She was very surprised that the Gen-Button and Gen-Panel interfaces actually had the same exact layout when she took off the blindfold at the end of the study and was shown the various interfaces.

For most participants, the four measures (interface ranking, success rate, SEQ, and SUS) were moderately to strongly correlated (see Table 6-2). However, Participant #3 did not show significant correlation between the success rate measure and the other measures.

Table 6-2. Spearman's rank correlation coefficients for each participant's experimental measures. The correlations in parentheses were not significant at p=0.05. All the other correlations were significant with p<0.001.

	Participant #1				*Participant #2*				*Participant #3*			
	Rank	**Success**	**SEQ**	**SUS**	**Rank**	**Success**	**SEQ**	**SUS**	**Rank**	**Success**	**SEQ**	**SUS**
Rank	1				1				1			
Success Rate	-0.485	1			-0.732	1			(-0.069)	1		
SEQ	-0.792	0.555	1		-0.699	0.805	1		-0.844	(0.131)	1	
SUS	-0.822	0.544	0.880	1	-0.586	0.643	0.778	1	-0.859	(0.255)	0.849	1

Participants ranked the interfaces according to their preference after completing all of the tasks on all of the interfaces on copier and on the thermostat devices (see Table 6-3). The Mfr-Keyboard interface was always ranked in last place. Participants did not always rank the other interfaces consistently between devices.

Table 6-3. Participant rankings of interface types.

	Participant #1					*Participant #2*				*Participant #3*			
Device	**Mfr-Kybd**	**Gen-Button**	**Gen-Panel**	**Gen-List**	**Gen-Loop**	**Mfr-Kybd**	**Gen-Button**	**Gen-Panel**	**Gen-Loop**	**Mfr-Kybd**	**Gen-Button**	**Gen-Panel**	**Gen-Loop**
Copier	5th	1st	2nd	4th	3rd	4th	2nd	1st	3rd	4th	3rd	1st	2nd
Thermo.	5th	1st	2nd	3rd	4th	4th	1st	2nd	3rd	4th	3rd	1st	2nd

The success rate measure is the percentage of the tasks that a participant completed successfully for the five device-specific tasks (see Figure 6-2). Note that for Participant #3, the success rate was not correlated with her other measures.

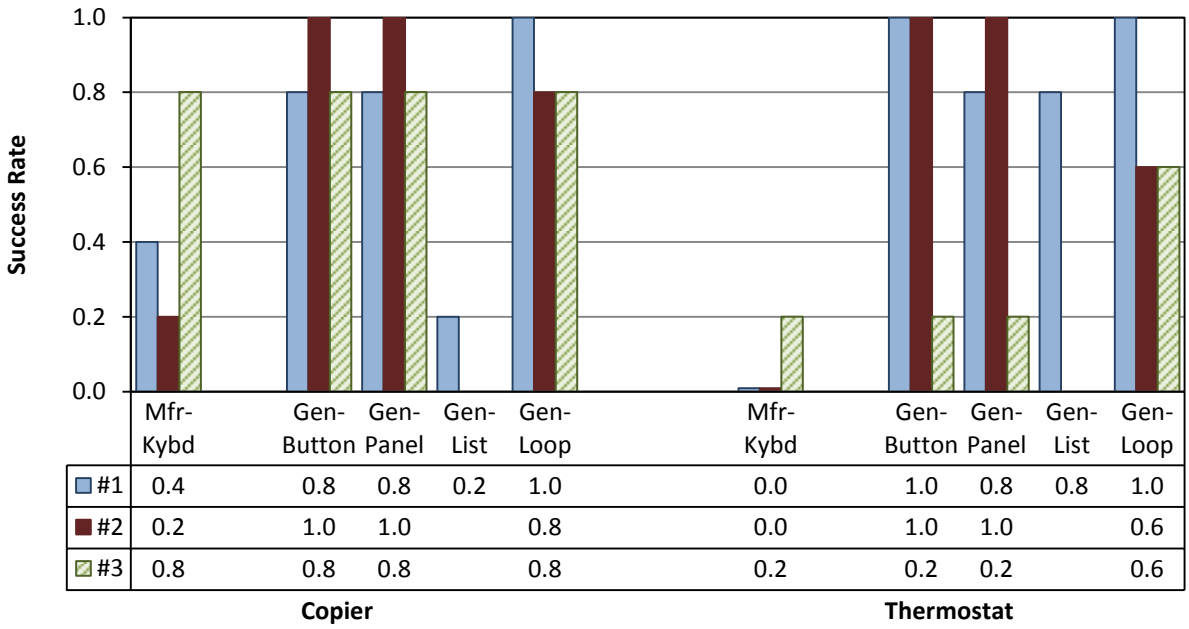


Figure 6-2. Success rate data for each participant. Each participant attempted 5 tasks per interface per device.

The Single Ease Question (SEQ) was asked after participants completed each task. The SEQ data averaged across all five device-specific tasks is shown in Figure 6-3.

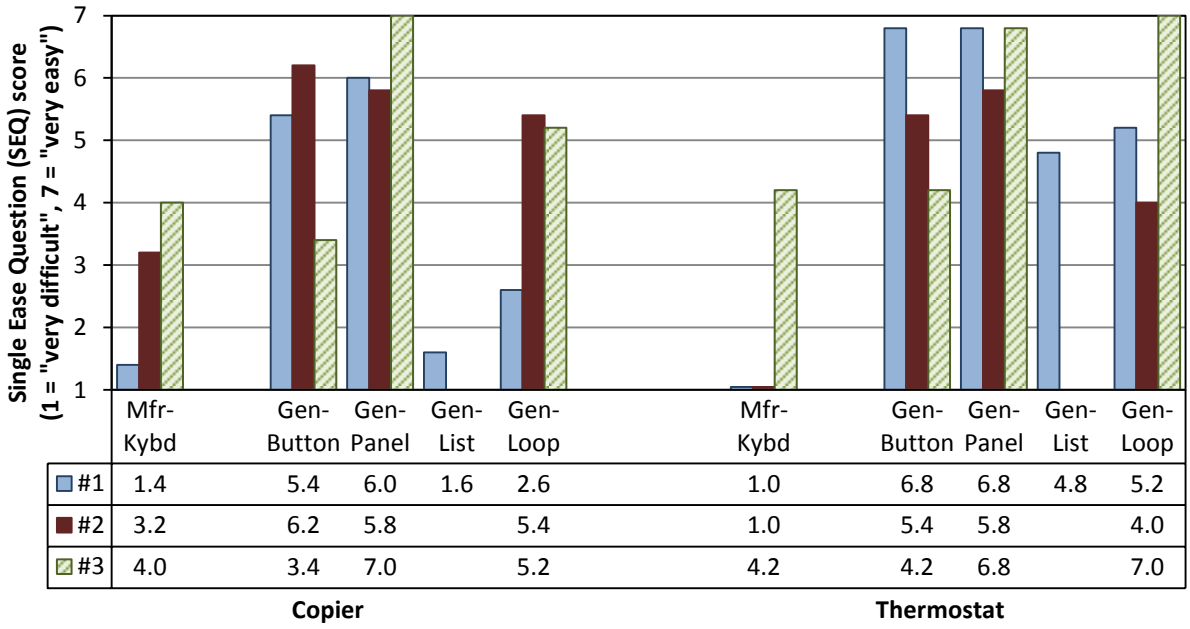


Figure 6-3. SEQ data averaged across the five device-specific tasks for each participant.

The System Usability Scale (SUS) questionnaire was verbally administered to participants after they had finished all five tasks on an interface. Participants' SUS scores are shown in Figure 6-4.

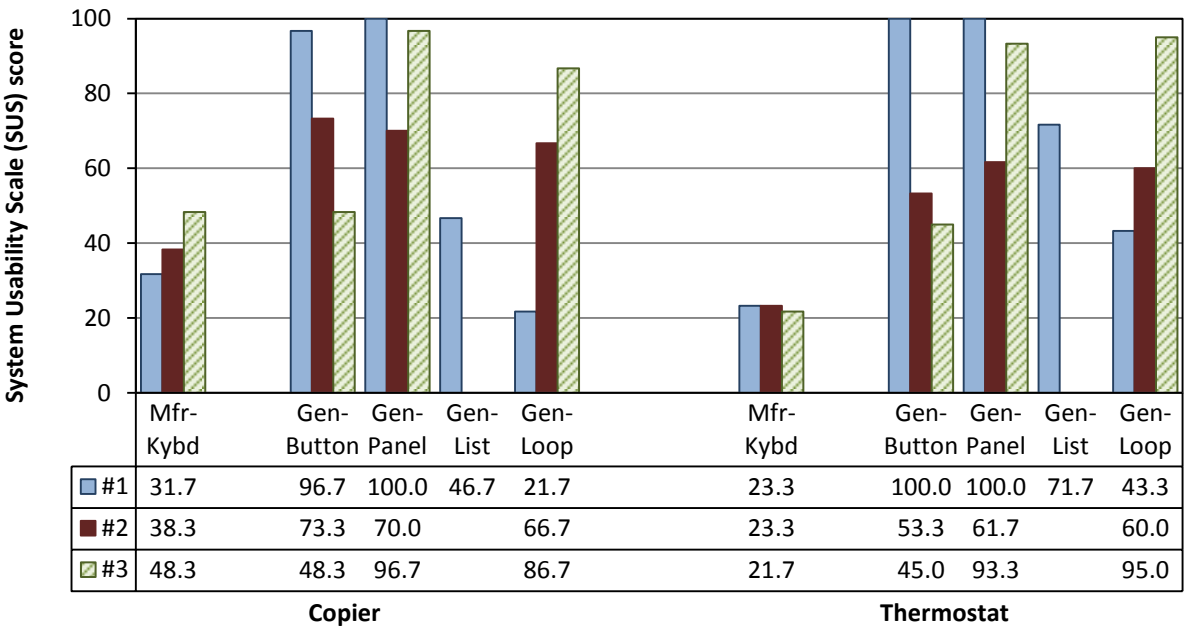


Figure 6-4. Systems Usability Scale (SUS) scores for each participant.

6.5. Discussion

The Gen-Panel and Gen-Button auto-generated interfaces were developed in just a few weeks after a participant was excluded from the main study. It was felt that this participant needed an easier type of self-voicing interface even though it was likely to be less efficient than the gesture-based interfaces in the main study.

There are few obvious trends in the data from such a small number of participants. Each of these participants was included in the case study for different reasons, so it would not be expected that they would all pick the same favorite interface or have similar patterns of performance or usability. Also note that Participants #2 and #3 had more extensive practice with the Gen-Loop interface since they had completed the main study and used it then.

The Mfr-Keyboard interface (i.e., the manufacturer-created electronic interface as used by the iOS VoiceOver screen reader) is the one that would be available to users today without automatically-generated interfaces. In all cases, participants ranked the Mfr-Keyboard interfaces in last place.

The prototype Gen-Button interface that was tested could be improved. The electrically conductive contact pads on the bottom of the polydome buttons did not seem to always make full contact with and activate the touchscreen. A commercial AT button interface could potentially be a Bluetooth device with reliable physical keys that could pair with a mobile phone or other computing device. Alternatively, a button interface could use technology similar to currently available silicon tactile keyboard overlays for touchscreen devices. With a better and more reliable feel, perhaps a physical button based interface would score higher for more people.

The Gen-Panel interface is simpler than Gen-Button interface, with no moving parts, and could be commercially made less expensively. The Gen-Panel interface was very sensitive because most of the touchscreens on current mobile devices do not actually need to be touched in order to be activated. Hovering a finger very close to the screen is often enough to be sensed as a touch. The Gen-Panel interface effectively made it easy to hover very close to the touchscreen when users were feeling the edges of the holes in the panel. This problem could potentially be ameliorated by having a thicker panel, by having raised tactile features around the holes (so users do not have to put their finger over the hole itself when feeling the holes in the panel), or by reducing the sensitivity of the touchscreen.

The participants had different preferences for auto-generated interfaces. The fact that the pilot participant who was blind had noticeable difficulty with the Gen-Panel interface suggests that the Gen-Button and Gen-Panel interfaces are not for everybody. This pilot participant had deep-seated habits for the gestures used with the iOS VoiceOver screen reader. For people who can reliably make gestures, the gesture-based auto-generated interfaces (i.e., Gen-List and Gen-Loop) are likely to be more efficient: instead of navigating step by step, users can drag a finger on the screen to quickly get to the desired

element. However, the three case study participants most preferred one or the other of the non-gesture interfaces specifically created for the case studies.

6.6. Conclusion

This case study demonstrated again that people perform better and prefer automatically-generated self-voicing interfaces to manufacturer-created interfaces that are used with screen readers. People who have difficulty reliably performing or remembering the gestures may similarly prefer Panel- or Button-like auto-generated interfaces over gesture-based. This preference is not universal however. Interestingly, this provides further evidence of the value of auto-generated interfaces, because these disparate preferences for different interface types, and the accompanying different performances (or lack of performance) can potentially be accommodated through automatic model-based user interface generation.

Chapter 7.
Discussion

This study adds to the limited evidence from user testing that auto-generated interfaces can indeed be more usable than manufacturer-created interfaces to people, particularly those with disabilities. The magnitude of the improvement of the auto-generated interface over the manufacturer-created interfaces from this study was higher than expected. Comparing the List auto-generated interfaces to the manufacturer-created Item interfaces, blind participants completed 64% more tasks and the inexperienced blindfolded completed more than 5 times as many tasks on the List interface. On Sauro and Lewis's letter-grade-on-a-curve scheme based on percentiles (2012, p. 204), the SUS usability score for the List interfaces was an *A*− compared to an *F*-grade for the Item interfaces. Both the generator and model are relatively simple by design and did not take into account all of the factors that a designer of such generators might consider. That such a simple model and generator made such an improvement on the study participants' performances is notable, because the best that people have today are like the manufacturer-created "Item" interfaces included in the user study.

Today, screen reader users interact with Item-like interfaces all the time, which may have varying levels of accessibility. For most interfaces, accessibility guidelines are only partially followed, and the accessibility that does exist was not explicitly designed into the system, but was a part of the underlying software development platforms and toolkits. Unfortunately, few interfaces are explicitly designed with accessibility guidelines in mind and even fewer are designed specifically for people with disabilities (like the Lexmark copier's Item interface, which is an exemplar accessible interface for a product). Increasing the accessibility knowledge of application developers is useful, but model-based automatic UI generation is another powerful tool that may be easier to implement and can potentially accommodate more people

and their needs and preferences. More research and development needs to be done in the area of model-based automatic UI generation to make it a reality.

7.1. Other Potential Changes for Model-Based UI Generation

Interface generation rules that combine heuristics and user preferences may help generate optimal individual user interface. The main method of personalization in the study was swapping or changing interactors to ones that are usable and most preferred. This ability to swap interactors is vital for people who can only handle limited modalities and interactors. However, many people are flexible and can easily adapt to using different interactors—many of the participants in the study had two or more interactors that were tied. Even for participants who had a single favorite interactor, many of them seemed to do fine with multiple interactors during the preference elicitation sessions. A more sophisticated interface generator could potentially choose between multiple interactors for the same USI type based on the USI characteristics. Not many people chose radio buttons in the study's basic preference elicitation tasks, but perhaps a set of radio buttons would be the best interactor if a single selection was required (i.e., minimum cardinality = 1) from a small to moderate number of possible selections.

7.1.1 Enhancing the model

Including task-based information as enhancements to the USI model could help a more elaborate interface generator choose between multiple interactors. For example, a switch interactor could be used whenever two choices are opposite in value (e.g., using a switch for On/Off, but not for making a choice between AM and FM radio bands). An enhanced USI model might say what physical property is being adjusted, say an angle in degrees, and then a more appropriate interactor might be chosen, such as a rotary dial instead of linear slider. A "wish list" of some potential enhancements is included in Appendix C.

The problem with enhancing the model with more required elements or characteristics quickly becomes one of feature creep and complexity however. Manufacturers are less likely to use complex

models and use them correctly, especially very abstract models. Also, the people who most need auto-generated interfaces—those with severe and uncommon needs—are the ones who are most unlikely to have sophisticated generators that can take advantage of a complex model. They are more likely to have straightforward generators that simply swap interactors, like the ones tested in this study. The base FIN-USI model was specifically developed to define a minimum set of (abstract) interactors that would be needed to fulfill the needs of a broad range of devices and functionality.

7.1.2 Changing or repairing a target-interface model

After trying a number of different interfaces over several sessions, one participant remarked that using the different interfaces was not really the main issue for him. He could use many of the automatically-generated interfaces to successfully complete the task, but he felt that his biggest problem was understanding the purpose of the different controls and settings. He said that he would like the interface to be clearer, with labels that were more intuitive and understandable to him. This statement underscores the importance of the abstract user interface model to the generation process—a poor model as input is likely to lead to relatively poor quality generated interfaces (junk in, junk out).

It may be possible for interface generators in the future to repair or improve the underlying model and subsequently generated interface. These interface changes could be helpful for many users, but would be particularly so for people with cognitive, language, and learning disabilities.

Poor labels may make it hard to find the right options or make the correct setting. Generators might be able to find alternative labels based on metadata, a thesaurus, or the interfaces a person has already used successfully. These different strategies need to be developed and tested with users.

The structure and nesting of interface also has an effect on usability. Generators have the potential to restructure an interface to improve an interface or better match a person's prior experience as demonstrated by the PUC project (J. Nichols et al., 2007); however that project required an explicit mapping be created between every related interactor on every device. This type of pairing is exponential

and thus not scalable. Device interactors could potentially be linked to a listing of canonical forms, which could then be used to restructure an interface, but this strategy places additional burden on manufacturers and does not support new categories of devices and functionality. Future generators that use artificial intelligence or other strategies that may be able to address the re-structuring issue, but additional research needs to be done in this area.

The structure of an interface could potentially be simplified for users who just want to use the basic features of a device or those who are easily confused with complexity. More advanced features could be automatically removed from the interface or hidden in advanced menus or options. The target-interface model could potentially include information about what functionalities are the primary or advanced ones; although this would increase the manufacturer burden. A simplifying interface generator could try to automatically figure out which features are secondary or advanced ones through artificial intelligence or usage statistics collected from a population of users. These approaches would need to be tested to see the quality of simplified interfaces and the benefit for users.

Another interface transformation that would be useful to many people would be specialized interactors for composite inputs. The potential number of composite inputs are unlimited however, so it would be useful to have a survey and study of interfaces to determine what types of composite inputs are most used and most useful. These primary composite inputs could be collected into catalog that could be used by both manufacturers and interface generator designers. Not all interface generators would need to have specialized composite interactors, but they would be useful for generators that support people who have many needs and preferences shared in common with many others.

7.2. Potential Interface Improvements for Gestures

Misinterpreted gestures were problematic because the system's focus and context would suddenly change on the user. This was disorienting to users and caused frustration because the person had to (1)

recognize that the focus was somewhere unexpected, (2) move back to the intended interactor, and (3) try the gesture again.

It may be possible to reduce these unintended changes of focus and the resultant disorientation by having buffer thresholds for each gesture. Gestures within the acceptance thresholds would be accepted, gestures outside the acceptance thresholds but within the buffer thresholds would be ignored (or an error tone might be supplied), and gestures outside of those two thresholds would then be treated as direct pointing exploration. With buffered gestures users may only be inconvenienced by having to try the ignored gesture again (rather than being disoriented by a sudden change of context and focus with a slightly-missed gesture as with the current system).

It would have been useful to have allowed all of the double-tap parameters to be set to according to personal performance instead of just the double-tap rate. One blind participant had tremor that caused him to have draggy taps on the touchscreen. Often this resulted in the system changing focus rather than activating the desired element. During the session, he mentioned that he thought it would be good to have a "get to know you session" where the system could adjust different gesture detection parameters to the user's actual performance. During the case study conducted with this participant, the double-tap wiggle parameter was doubled to allow 5 mm of movement with a tap. This increased threshold seemed to make it easier for this participant to use the Loop interface.

It seems that it would be useful to have personalized parameters for the other gestures on the List and Loop interfaces. For example, if a person had difficulty swiping as quickly as other people, he or she could instead swipe at a speed that is comfortable. If a person had difficulty making straight swipes, then the system could adapt to that. This parameterized gesture input could be set during a personalization session. As an extra feature, a more sophisticated gesture handling system could adjust the parameters dynamically as the person continues to use the interface (like was done with the double-tap rate in the

experiment). This could be helpful for people whose performance changes with fatigue, time of day, or other factors.

7.3. The Utility of the Novel Loop Interface

The Loop interface is a new interface type that would be most practical as an automatically-generated interface: manufacturers are not likely to manually create Loop interfaces when more typical, accessible interfaces would suffice. The Loop interface had been developed with efficiency in mind. While using a device, a person could quickly drag around the screen and learn an interface. Once familiar with a specific interface, users could just quickly and directly tap the elements of interest based on their spatial knowledge of the interface.

From the study, there was limited evidence in favor of the Loop interface over the List interface, but there was no statistically significant improvement in performance. Comparing the List and Loop interfaces was not the intent of the study however. A future study could be done to compare performance differences of the Loop interface versus other interfaces. A better comparison could be made if the interactors were similar between interfaces rather than letting participants pick their own favorites based on their own personal criteria before they had used the interfaces for very long. As it was, participant did not always choose the quickest or most efficient interactor for a given interface—this reduced the statistical power for finding performance differences, which already tend to have wide variances.

For the blind users in general, the initial reception towards the Loop interface was not particularly favorable. It took a while for participants to become accustomed to the interface. For some it became their favorite, but not for everybody. People may be able to weed out obviously bad interactors or interfaces, but they may need some time and experience with novel interfaces in order to decide on the one that is best for them. This suggests a way in which the preference elicitation session could be improved.

Participants could first eliminate the interactors and interface types that are obviously bad (for them), but then spend some time trying the finalists in several interfaces before making final decisions.

The number of participants who preferred the Loop interface's Slider interactor was unexpected. (The loop slider had all of the values on the right edge of the screen where a person could drag and then double-tap to select a number.) When it was developed it was felt that it might be good interactor for limited circumstances where users did not need to set a particularly accurate value (which is the type of tasks where graphical slider interactors are often used). Many participants seemed to like directly selecting values with the slider rather than having to type a value. However, many of these same participants also found it more difficult to use the slider when there were many values from which to select. This was likely because a small movement of the finger had the potential to change the value. Only a few participants realized that they could use the slider to get close and then swipe right or left to get to an exact value. Because the slider was generally liked by participants, it would be good to develop and test enhancements that could potentially be made to improve a non-visual slider. For example, the granularity or precision of the slider might change as a person moves their finger horizontally (i.e., orthogonal to the direction of the slider).

7.4. Other Uses of the FIN-USI Model

The FIN-USI model is a general model of inputs that systems need and abstract user interfaces that can be used to fulfill those needs. Because it is a general model, it can be applied to more situations than model-based UI generation of entire interfaces.

During its development and user testing, I have found myself using the model when thinking about accessibility problems with particular interfaces. It has been helpful to look at an inaccessible, novel interface or interactor and see what the FIN actually is. By knowing the FIN, it is much clearer

what other types of interfaces might be more appropriate for other groups of users. Otherwise it is easy to get distracted by the features of a particular inaccessible interface.

Instead of modeling a system's entire interface, the USI side of the model could potentially be used with relatively little enhancement to improve the accessibility of mainstream interfaces screen by screen. This is somewhat like how WAI-ARIA attributes can be added to HTML interfaces to improve their accessibility. The WAI-ARIA attributes are limited to specific supported abstract interactors however. Novel interactors are more difficult to make accessible with WAI-ARIA. WAI-ARIA also requires people to interact with layered interactors if multiple interactors together are used to enter values for a single setting.

Since the USI model functions on the level of input rather than interactor type like WAI-ARIA, the USI model may be a better way of tagging an interface for accessibility in the future. Under this approach, single or composite interactors could be tagged with their respective USIs. When AT or an interface generator comes across a USI, then it can substitute the user's preferred interactor rather than users having to interact with a potentially more complex, layered interface (e.g., a screen reader interacting with a WAI-ARIA layer, in order to interact with a mainstream graphical user interface, in order to control a system's functionality).

7.5. Potential Applications to Current Screen Readers

It would take some time for model-based automatic user interface generation to become a commercial reality. Manufacturers would need to model their devices and create interface sockets at the same time that other developers would need to create interface generators for various PAUI devices. However, there are some results of the study that could be applied immediately to the design of screen readers on mobile devices.

Current screen readers on mobile devices allow for control through gestures or through a connected keyboard. Using the gestures is a much more portable solution and would likely be preferred by most people, but the keyboard should remain an option for those screen reader users who have physical difficulty performing gestures reliably.

Touchscreen screen readers must allow for more flexibility with gestures than they do now. The many thresholds that can be used to determine a particular gesture need to be more flexible or adapt to the person. Activation should allow adjustments to accept slower, more draggy, or more widely spaced double-taps or other gestures entirely. Swipe gestures could be adjusted to accept slower and less straight swipes. These parameters could be set for an individual with an initial personalization session where the system cues the user to make particular gestures, measures the user's performance, and then creates a profile for the future use. A dynamic system that adjusts with time would be a nice addition. Some people who do not consider themselves to have a disability aside from blindness are unable to use the current screen readers because they are cannot adapt to the inputs that are required.

Touchscreen screen readers should ideally allow more customization in what gestures the system requires for operation. For example, some people cannot make reliable three finger swipes, but could perhaps trace back and forth quickly with a single finger, drag the side of their hand across the screen, or any other number of gestures. The personalization session could see what gestures are particularly difficult for a user and suggest or allow options for alternatives.

Physical buttons still have uses and benefits even in the touchscreen world. Some people may have disabilities that make reliable gestures impossible. Other people prefer physical buttons, as with a couple of the blind study participants who were using flip phones and landlines. The buttons already on a phone could potentially be used, but these often have very important dedicated purposes. Additional physical Buttons could potentially be added to a case for a mobile device and connect via Bluetooth, a wired connection, through Near-Field Communication (NFC), or other technology. The operating systems

on mobile devices would need to accept these different inputs as trusted input in order for these strategies to work.

The study also pointed out some opportunities for improving the interface design for current screen readers. Participants preferred the newly developed interactors to those that already existed. Some of these new concepts and interactors could be implemented in screen readers today and not just in the auto-generated interfaces in the future. Some participants had difficulty using the native iOS on-screen keyboard with VoiceOver. The keyboard would close with a quiet sound effect if they accidentally touched any control above the keyboard on the screen. The most criticized interactors were particularly awkward because of the layered interface problem, where participants had to use a primarily graphical interactor with a screen reader. More direct interactors could be substituted for users who cannot see well enough to get context from the screen (which is a subset of the blind population). Blind users with some level of vision may be able to handle these layered-interface interactors more easily, but this would need to be tested. It may be that even with some level of vision that users might still prefer more direct interactors.

7.6. In Closing

Automatic model-based UI generation is unlikely to completely replace the manual creation of interfaces, particularly for mainstream users. Designers can consider more factors about the tasks and (mainstream) users than can easily be encoded in an abstract UI model and supported by a generator. Manually designed interactors can be tuned for very specific tasks to provide the best user experience. Another reason for the primacy of manually-designed interfaces is branding—the look and feel of an interface is very often part of the brand of a device. Some companies are very proud of their industrial and interface designs and go to great efforts to have a consistent experience for users.

Where model-based auto-generation is most useful is for creating *alternative* interfaces for those who have difficulty with manually-designed interfaces. This could be for a variety of reasons (inaccessibility, UI not available natively on the user's device platform, remote access, etc.), but increasing accessibility to a wider range of users is a much needed feature. Model-based auto-generation is likely to be one of the least expensive solutions towards broad accessibility across disabilities, particularly if both target device models and user interface generators are relatively easy to implement. With mobile devices becoming more prevalent, more people have access to PAUI devices that can connect to devices and generate personalized interfaces for control.

References

Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, *24*(6), 574.

Bass, L., Faneuf, R., Little, R., Mayer, N., Pellegrino, B., Reed, S., … Szczur, M. R. (1992). A Metamodel for the Runtime Architecture of an Interactive System: The UIMS Tool Developers Workshop. *SIGCHI Bulletin*, *24*(1), 32–37.

Berti, S., Correani, F., Paternò, F., & Santoro, C. (2004). The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction. In *Leveles, Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages* (pp. 103–110).

Bichard, J.-A., Coleman, R., & Langdon, P. (2007). Does My Stigma Look Big in This? Considering Acceptability and Desirability in the Inclusive Design of Technology Products. In C. Stephanidis (Ed.), *Universal Access in Human Computer Interaction. Coping with Diversity* (pp. 622–631). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-73279-2_69

Biron, P. V., & Halhotra, A. (Eds.). (2004, October 28). XML Schema Part 2: Datatypes Second Edition. W3C Recommendation. Retrieved from http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Sacré, B., & Vanderdonckt, J. (1995). Towards a Systematic Building of Software Architecture: the TRIDENT Methodological Guide. In P. Palanque & R. Bastide (Eds.), *Design, Specification and Verification of Interactive Systems '95* (pp. 262–278). Springer Vienna.

Boyer, J. M. (Ed.). (2009, October 20). XForms 1.1. W3C Recommendation. Retrieved from http://www.w3.org/TR/2009/REC-xforms-20091020/

Brooke, J. (1996). SUS: A "quick and dirty" usability scale. In *Usability Evaluation in Industry*. London: Taylor and Francis.

Buxton, W. (1983). Lexical and Pragmatic Considerations of Input Structures. *SIGGRAPH Comput. Graph.*, *17*(1), 31–37.

Buxton, W. (1986). Chunking and Phrasing and the Design of Human-Computer Dialogues. In *Proceedings of the IFIP World Computer Congress* (pp. 475–480). North Holland Publishers.

Caldwell, B., Cooper, M., Reid, L. G., Vanderheiden, G., Chisholm, W., Slatin, J., & White, J. (Eds.). (2008, December 11). Web Content Accessibility Guidelines (WCAG) 2.0. W3C. Retrieved from http://www.w3.org/TR/WCAG20/

Calvary, G., Coutaz, J., Thevenin, D., Bouillon, L., Florins, M., Limbourg, Q., … Santoro, C. (2002, September 3). The CAMELEON Reference Framework. Retrieved from http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEON%20D1.1RefFramework.pdf

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, *15*(3), 289–308.

Card, S. K., Mackinlay, J. D., & Robertson, G. G. (1990). The Design Space of Input Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 117–124). New York, NY, USA: ACM.

Clarkson, J., Coleman, R., Keates, S., & Lebbon, C. (2003). From margins to mainstream. In J. Clarkson, R. Coleman, S. Keates, & C. Lebbon (Eds.), *Inclusive Design: Design for the Whole Population* (pp. 1–25). London: Springer-Verlag.

Conover, W. J., & Iman, R. L. (1981). Rank Transformations as a Bridge Between Parametric and Nonparametric Statistics. *The American Statistician*, *35*(3), 124–129.

Coutaz, J. (1987). PAC, An Object-Oriented Model for Dialog Design. In H.-J. Bullinger & B. Shackel (Eds.), *INTERACT 87 - 2nd IFIP International Conference on Human-Computer Interaction September 1-4, 1987, Stuttgart, Germany*. (pp. 431–436). North-Holland.

Craig, J., Cooper, M., Pappas, L., Schwerdtfeger, R., & Seeman, L. (Eds.). (2014, March 20). Accessible Rich Internet Applications (WAI-ARIA) 1.0. W3C. Retrieved from http://www.w3.org/TR/2014/REC-wai-aria-20140320/

Danturthi, R. S., Shroff, P., & Winters, J. M. (2006). Progress in Using the Universal Remote Console Standard to Create User-Customized Interfaces for Future Medical Devices. In J. M. Winters & M. F. Story (Eds.), *Medical Instrumentation* (pp. 373–391). Boca Raton: CRC Press.

Dawes, J. (2008). Do data characteristics change according to the number of scale points used? An experiment using 5-point, 7-point and 10-point scales. *International Journal of Market Research*, *50*(1), 61–77.

Deng, J., Kemp, E., & Todd, E. G. (2006). Focussing on a Standard Pattern Form: The Development and Evaluation of MUIP. In *Proceedings of the 7th ACM SIGCHI New Zealand Chapter's International Conference on Computer-human Interaction: Design Centered HCI* (pp. 83–90). New York, NY, USA: ACM.

Duce, D. A., Gomes, M. R., Hopgood, F. R. A., & Lee, J. R. (Eds.). (1991). Concepts, Methods, Methodologies Working Group. In *User interface management and design: Proceedings of the Workshop on User Interface Management Systems and Environments, Lisbon, Portugal, June 4-6, 1990* (pp. 35–49). Berlin: Springer-Verlag.

Engel, J., & Märtin, C. (2009). PaMGIS: A Framework for Pattern-Based Modeling and Generation of Interactive Systems. In J. A. Jacko (Ed.), *Human-Computer Interaction. New Trends* (pp. 826–835). Springer Berlin Heidelberg.

Eustice, K. F., Lehman, T. J., Morales, A., Munson, M. C., & Guillen, M. (1999). A universal information appliance. *IBM Systems Journal*, *38*(4), 575–601.

Finstad, K. (2010). Response Interpolation and Scale Sensitivity: Evidence Against 5-Point Scales. *Journal of Usability Studies*, *5*(3), 104–110.

Foley, J. D., Wallace, V. L., & Chan, P. (1984). The Human Factors of Computer Graphics Interaction Techniques. *IEEE Comput. Graph. Appl.*, *4*(11), 13–48.

Gajos, K. (2005). SuppleType: Supple API [API Documentation]. Retrieved from https://www.cs.washington.edu/ai/supple/docs/edu/washington/cs/supple/rep/SuppleType.html

Gajos, K., & Weld, D. S. (2004). SUPPLE: Automatically Generating User Interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces* (pp. 93–100). New York, NY, USA: ACM.

Gajos, K. Z., Weld, D. S., & Wobbrock, J. O. (2010). Automatically generating personalized user interfaces with Supple. *Artificial Intelligence*, *174*(12–13), 910–950.

Gajos, K. Z., Wobbrock, J. O., & Weld, D. S. (2007). Automatically Generating User Interfaces Adapted to Users' Motor and Vision Capabilities. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology* (pp. 231–240). New York, NY, USA: ACM.

Gajos, K. Z., Wobbrock, J. O., & Weld, D. S. (2008). Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1257–1266). New York, NY, USA: ACM.

Gibson, B. (2007). Enabling an Accessible Web 2.0. In *Proceedings of the 2007 International Cross-disciplinary Conference on Web Accessibility (W4A)* (pp. 1–6). New York, NY, USA: ACM.

Gonzalez, A., & Reid, L. G. (2005). Platform-independent Accessibility API: Accessible Document Object Model. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)* (pp. 63–71). New York, NY, USA: ACM.

Guerrero Garcìa, J., Vanderdonckt, J., & Gonzàlez Calleros, J. M. (2008). FlowiXML: A Step Towards Designing Workflow Management Systems. *International Journal of Web Engineering and Technology*, *4*(2), 163–182.

Harper, S. (2007). Is there design-for-all? *Universal Access in the Information Society*, *6*(1), 111–113.

Harris, T. K., & Rosenfeld, R. (2004). A Universal Speech Interface for Appliances (pp. 249–252). Presented at the INTERSPEECH-2004, Jeju Island, Korea. Retrieved from http://www.cs.cmu.edu/~roni/papers/icslp2004-appl.pdf

Hayes, P. J., Lerner, R. A., & Szekely, P. A. (1983). The COUSIN User Interface Project. *ACM SIGOA Newsletter*, *4*(2), 3–7.

Hayes, P. J., Szekely, P. A., & Lerner, R. A. (1985). Design alternatives for user interface management systems based on experience with COUSIN. *ACM SIGCHI Bulletin*, *16*(4), 169–175.

Helms, J., Schaefer, R., Luyten, K., Vanderdonckt, J., Vermeulen, J., & Abrams, M. (Eds.). (2009, May 1). User Interface Markup Language (UIML) Version 4.0: Committee Specification 01. Retrieved from http://docs.oasis-open.org/uiml/v4.0/cs01/uiml-4.0-cs01.pdf

Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E. D., O'Connor, E., & Pfeiffer, S. (Eds.). (2014, October 28). HTML5. W3C Recommendation. Retrieved from http://www.w3.org/TR/2014/REC-html5-20141028/

Hix, D. (1990). Generations of user-interface management systems. *IEEE Software*, *7*(5), 77–87.

Hodes, T. D., Katz, R. H., Servan-Schreiber, E., & Rowe, L. (1997). Composable Ad-hoc Mobile Services for Universal Interaction. In *Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking* (pp. 1–12). New York, NY, USA: ACM.

Icontrol Networks. (2015). *2015 State of the Smart Home Report*. Icontrol Networks. Retrieved from http://www.icontrol.com/blog/2015-state-of-the-smart-home-report/

ISO. (2008). *Ergonomics of human-system interaction - Part 171: Guidance on software accessibility* (Vol. Reference Number: ISO/IEC 9241–171:2008(en)). Geneva, Switzerland: ISO.

ISO/IEC. (2009). *Information technology-Framework for specifying a common access profile (CAP) of needs and capabilities of users, systems, and their environments* (Vol. Reference Number: ISO/IEC 24756:2009). Geneva, Switzerland: ISO/IEC.

ISO/IEC. (2014a). *Information technology-User Interfaces-Universal Remote Console-Part 1: General Framework* (Vol. Reference Number: ISO/IEC 24752–1:2014). Geneva, Switzerland: ISO/IEC.

ISO/IEC. (2014b). *Information technology-User Interfaces-Universal Remote Console-Part 2: User interface socket description* (Vol. Reference Number: ISO/IEC 24752–2:2014). Geneva, Switzerland: ISO/IEC.

ISO/IEC. (2014c). *Information technology-User Interfaces-Universal Remote Console-Part 3: Presentation template* (Vol. Reference Number: ISO/IEC 24752–3:2014). Geneva, Switzerland: ISO/IEC.

ISO/IEC. (2014d). *Information technology-User Interfaces-Universal Remote Console-Part 4: Target description* (Vol. Reference Number: ISO/IEC 24752–4:2014). Geneva, Switzerland: ISO/IEC.

ISO/IEC. (2014e). *Information technology-User Interfaces-Universal Remote Console-Part 5: Resource description* (Vol. Reference Number: ISO/IEC 24752–5:2014). Geneva, Switzerland: ISO/IEC.

ISO/IEC. (2014f). *Information technology-User Interfaces-Universal Remote Console-Part 6: Web service integration* (Vol. Reference Number: ISO/IEC 24752–6:2014). Geneva, Switzerland: ISO/IEC.

Jordan, J. B., & Vanderheiden, G. C. (2013). Modality-Independent Interaction Framework for Cross-Disability Accessibility. In P. L. P. Rau (Ed.), *Cross-Cultural Design. Methods, Practice, and Case Studies* (pp. 218–227). Springer Berlin Heidelberg.

Krasner, G. E., & Pope, S. T. (1988). A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, *1*(3), 26–49.

Lantz, K. A., Tanner, P. P., Binding, C., Huang, K.-T., & Dwelly, A. (1987). Reference Models, Window Systems, and Concurrency. *ACM SIGGRAPH Computer Graphics*, *21*(2), 87–97.

Law, C., & Vanderheiden, G. (2000). The development of a simple, low cost set of universal access features for electronic devices. In *Proceedings on the 2000 conference on Universal Usability* (pp. 118–123). Arlington, Virginia, United States: ACM.

Lee, C., Novak, M., Schauer, J. M., & Vanderheiden, G. (1990, October). *Providing access in Windows 3.0*. Talk presented at the Eighth Annual Closing the Gap Conference, Minneapolis, Minn.

Lee, C., & Vanderheiden, G. (1987). Keyboard equivalent for mouse input. In *Proceedings of the Tenth Annual Conference on Rehabilitation Engineering* (pp. 711–713). Washington, D.C.: RESNA.

Lee, J. H., Herzog, T. A., Meade, C. D., Webb, M. S., & Brandon, T. H. (2007). The use of GEE for analyzing longitudinal binomial data: A primer using data from a tobacco intervention. *Addictive Behaviors*, *32*(1), 187–193.

Lewis, J., & Sauro, J. (2009). The Factor Structure of the System Usability Scale. In *Human Centered Design* (pp. 94–103).

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., & Trevisan, D. (2004). UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces. In *Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages"* (pp. 55–62).

Mace, R. L. (1998). *A Perspective on Universal Design*. Retrieved from https://www.ncsu.edu/ncsu/design/cud/about_us/usronmacespeech.htm

Märtin, C., Herdin, C., & Engel, J. (2013). Patterns and Models for Automated User Interface Construction – In Search of the Missing Links. In M. Kurosu (Ed.), *Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments* (pp. 401–410). Springer Berlin Heidelberg.

Meixner, G., Paternò, F., & Vanderdonckt, J. (2011). Past, Present, and Future of Model-Based User Interface Development. *I-Com*, *10*(3), 2–11.

Molich, R., Ede, M. R., & Kaasgaard, K. (2004). Comparative Usability Evaluation. *Behaviour & Information Technology*, *23*, 65–74.

Montero, F., & López-Jaquero, V. (2007). IdealXML: An Interaction Design Tool. In G. Calvary, C. Pribeanu, G. Santucci, & J. Vanderdonckt (Eds.), *Computer-Aided Design of User Interfaces V* (pp. 245–252). Springer Netherlands.

Mori, G., Paternò, F., & Santoro, C. (2004). Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering*, *30*(8), 507–520.

Mozilla. (n.d.). XUL. Retrieved December 1, 2014, from https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL

Myers, B., Hudson, S. E., & Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, *7*(1), 3–28.

Nichols, A. L., & Maner, J. K. (2008). The Good-Subject Effect: Investigating Participant Demand Characteristics. *Journal of General Psychology*, *135*(2), 151–166.

Nichols, J., Chau, D. H., & Myers, B. A. (2007). Demonstrating the Viability of Automatically Generated User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1283–1292). New York, NY, USA: ACM.

Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., & Litwack, K. (2003). Personal Universal Controllers: Controlling Complex Appliances with GUIs and Speech. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems* (pp. 624–625). New York, NY, USA: ACM.

Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., & Pignol, M. (2002). Generating Remote Control Interfaces for Complex Appliances. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology* (pp. 161–170). New York, NY, USA: ACM.

Nichols, J., Myers, B. A., & Litwack, K. (2004). Improving Automatic Interface Generation with Smart Templates. In *Proceedings of the 9th International Conference on Intelligent User Interfaces* (pp. 286–288). New York, NY, USA: ACM.

Nichols, J., Myers, B. A., Litwack, K., Higgins, M., Hughes, J., & Harris, T. K. (2004). Describing Appliance User Interfaces Abstractly with XML. In *Proceedings of the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages* (pp. 9–16). Gallipoli, Italy.

Nichols, J., Myers, B. A., & Rothrock, B. (2006). UNIFORM: Automatically Generating Consistent Remote Control User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 611–620). New York, NY, USA: ACM.

Nichols, J., Rothrock, B., Chau, D. H., & Myers, B. A. (2006). Huddle: automatically generating interfaces for systems of multiple connected appliances. In *Proceedings of the 19th annual ACM*

symposium on User interface software and technology (pp. 279–288). New York, NY, USA: ACM.

Olsen, D. R., Jr. (1989). A Programming Language Basis for User Interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 171–176). New York, NY, USA: ACM.

Olsen, D. R., Jr., & Dempsey, E. P. (1983). SYNGRAPH: A Graphical User Interface Generator. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 43–50). New York, NY, USA: ACM.

Olsen, D. R., Jr., Jefferies, S., Nielsen, T., Moyes, W., & Fredrickson, P. (2000). Cross-modal Interaction Using XWeb. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology* (pp. 191–200). New York, NY, USA: ACM.

Paternò, F. (2000). *Model-Based Design and Evaluation of Interactive Applications*. London: Springer-Verlag.

Paternò, F., Santoro, C., & Spano, L. D. (2009). MARIA: A Universal, Declarative, Multiple Abstraction-level Language for Service-oriented Applications in Ubiquitous Environments. *ACM Trans. Comput.-Hum. Interact.*, *16*(4), 19:1–19:30.

Peissner, M., Häbe, D., Janssen, D., & Sellner, T. (2012). MyUI: Generating Accessible User Interfaces from Multimodal Design Patterns. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (pp. 81–90). New York, NY, USA: ACM.

Peißner, M., Sellner, T., & Janssen, D. (2012). MyUI Individualization Patterns for Accessible and Adaptive User Interfaces (pp. 25–30). Presented at the SMART 2012, The First International Conference on Smart Systems, Devices and Technologies.

Peterson, D., Gao, S., Malhotra, A., Sperberg-McQueen, C. M., & Thompson (Eds.). (2012, April 5). W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. W3C Recommendation. Retrieved from http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/

Pfaff, G. E. (Ed.). (1985). *User Interface Management Systems, Proceedings of IFIP/EG Workshop on UIMS*. Seeheim, Federal Republic of Germany: Springer-Verlag.

Ponnekanti, S. R., Lee, B., Fox, A., Hanrahan, P., & Winograd, T. (2001). ICrafter: A Service Framework for Ubiquitous Computing Environments. In G. D. Abowd, B. Brumitt, & S. Shafer (Eds.), *Ubicomp 2001: Ubiquitous Computing* (pp. 56–75). Springer Berlin Heidelberg.

Potel, M. (1996). *MVP: Model-View-Presenter, The Taligent Programming Model for C++ and Java*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.189.782&rep=rep1&type=pdf

Rodgers, B., Kelso, D., & Vanderheiden, G. C. (1982). Design of universal keyboard emulators. In *Proceedings of the Fifth Annual Conference on Rehabilitation Engineering* (p. 45). Bethesda, MD: RESNA.

Sanguinetti, A., Haga, H., Funakoshi, A., Yoshida, A., & Matsumoto, C. (2003). FReCon: A Fluid Remote Controller for a FReely Connected World in a Ubiquitous Environment. *Personal Ubiquitous Comput.*, *7*(3-4), 163–168.

Sauro, J., & Dumas, J. S. (2009). Comparison of three one-question, post-task usability questionnaires. In *Proceedings of the 27th international conference on Human factors in computing systems* (pp. 1599–1608). New York, NY, USA: ACM.

Sauro, J., & Lewis, J. R. (2012). *Quantifying the User Experience: Practical Statistics for User Research*. Waltham, Mass.: Morgan Kaufmann.

Schwerdtfeger, R. S. (1991, December). Making the GUI Talk. *BYTE*, 118–128.

Scott, N. G., & Gingras, I. (2001). The Total Access System. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems* (pp. 13–14). New York, NY, USA: ACM.

Shroff, P. P. (2005). *Algorithm to automatically generate multi-modal interfaces for hand held devices based on user preferences and abilities* (Master's thesis). Marquette University.

Simon, R., Jank, M., & Wegscheider, F. (2004). A Generic UIML Vocabulary for Device- and Modality Independent User Interfaces. In *In: Proc. of the 13th Int. World Wide Web Conference WWW'13* (pp. 17–22). ACM Press.

Smith, J. (2009, February). Patterns-WPF Apps With The Model-View-ViewModel Design Pattern. *MSDN Magazine*, *24*(2), 72–81.

Szekely, P. A., Sukaviriya, P. N., Castells, P., Muthukumarasamy, J., & Salcher, E. (1996). Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach. In *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction* (pp. 120–150). London, UK, UK: Chapman & Hall, Ltd.

Toothaker, L. E., & Newman, D. (1994). Nonparametric Competitors to the Two-Way ANOVA. *Journal of Educational and Behavioral Statistics*, *19*(3), 237–273.

Trewin, S., Zimmermann, G., & Vanderheiden, G. (2003). Abstract user interface representations: how well do they support universal access? In *Proceedings of the 2003 conference on Universal usability* (pp. 77–84). New York, NY, USA: ACM.

UCL (Ed.). (2013, February 6). D1.3: UsiXML Definition (UsiXML Deliverable) v 1.4.3. Retrieved from https://itea3.org/project/workpackage/document/download/1583/08026-UsiXML-WP-1-D13v3UsiXMLDefinition.pdf

Vanderdonckt, J. M., & Bodart, F. (1993). Encapsulating knowledge for intelligent automatic interaction objects selection. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems* (pp. 424–429). New York, NY, USA: ACM.

Vanderheiden, G. (2000). Fundamental principles and priority setting for universal usability. In *Proceedings on the 2000 conference on Universal Usability* (pp. 32–37). New York, NY, USA: ACM.

Vanderheiden, G. C. (1981). Practical Application of Microcomputers to Aid the Handicapped. *Computer*, *14*(1), 54–61.

Vanderheiden, G. C. (1996). Use of audio-haptic interface techniques to allow nonvisual access to touchscreen appliances. In *Proceedings of the Human Factors and Ergonomic Society Annual Meeting* (Vol. 40, p. 1266). SAGE Publications.

Vanderheiden, G. C. (1998). Universal Design and Assistive Technology in Communication and Information Technologies: Alternatives or Complements? *Assistive Technology*, *10*(1), 29–36.

Vanderheiden, G. C. (2008). Ubiquitous Accessibility, Common Technology Core, and Micro Assistive Technology: Commentary on "Computers and People with Disabilities." *ACM Trans. Access. Comput.*, *1*(2), 10:1–10:7.

Vanderheiden, G. C. (2009). Accessible and Usable Design of Information and Communication Technologies. In C. Stephanidis (Ed.), *The Universal Access Handbook* (pp. 3–1 – 3–26). Boca Raton: CRC Press.

Vanderheiden, G. C., & Jordan, J. B. (2012). Design for People with Functional Limitations. In G. Salvendy (Ed.), *Handbook of Human Factors and Ergonomics* (4th ed., pp. 1407–1441). Hoboken, New Jersey: John Wiley & Sons, Inc.

Vanderheiden, G. C., & Kelso, D. (1982). Dual and nested computer approach to vocational and educational computer systems. In *Proceedings of the Fifth Annual Conference on Rehabilitation Engineering* (p. 46). Bethesda, MD: RESNA.

Vander Zanden, B., & Myers, B. A. (1990). Automatic, Look-and-feel Independent Dialog Creation for Graphical User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 27–34). New York, NY, USA: ACM.

Venkatesh, V. (2000). Determinants of Perceived Ease of Use: Integrating Control, Intrinsic Motivation, and Emotion into the Technology Acceptance Model. *Information Systems Research*, *11*(4), 342–365.

Watson, L., & McCathie Nevile, C. (2015, March 15). Accessibility APIs: A Key To Web Accessibility. Retrieved November 11, 2015, from http://www.smashingmagazine.com/2015/03/web-accessibility-with-accessibility-api/

Webster, J., & Martocchio, J. J. (1992). Microcomputer Playfulness: Development of a Measure with Workplace Implications. *MIS Quarterly*, *16*(2), 201–226.

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, *265*(3), 94–104.

Weiser, M. (1993). Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, *36*(7), 75–84.

Weiser, M., Gold, R., & Brown, J. S. (1999). The origins of ubiquitous computing research at PARC in the late 1980s. *IBM Systems Journal*, *38*(4), 693–696.

Wendler, S., Ammon, D., Kikova, T., & Philippow, I. (2012). Development of Graphical User Interfaces based on User Interface Patterns (pp. 57–66). Presented at the PATTERNS 2012, The Fourth International Conferences on Pervasive Patterns and Applications.

Zimmermann, G., Jordan, J. B., Thakur, P., & Gohil, Y. (2013). GenURC: generation platform for personal and context-driven user interfaces. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility* (pp. 6:1–6:4). New York, NY, USA: ACM.

Zimmermann, G., & Vanderheiden, G. (2005). Use of User Interface Sockets to Create Naturally Evolving Intelligent Environments. In *Proceedings of the 11th International Conference on*

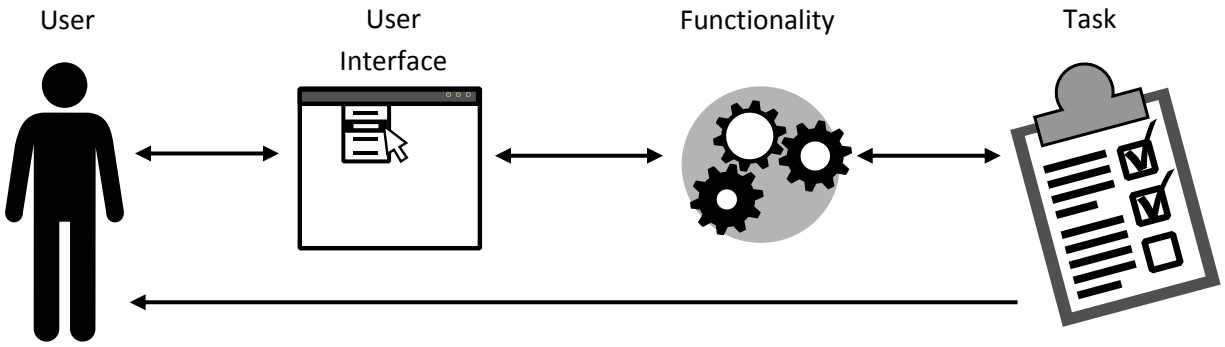
Human-Computer Interaction (HCII 2005). Las Vegas, Nevada, USA: Lawrence Erlbaum Associates.

Zimmermann, G., Vanderheiden, G., & Gilman, A. (2002). Prototype Implementations for a Universal Remote Console Specification. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems* (pp. 510–511). New York, NY, USA: ACM.

Appendix A. The Development of the FIN-USI Model

In order to be maximally useful to people with disabilities, a PAUI must be able to automatically generate an interface for any target device, software, or system a person might wish to control. This is a very broad range of potential tasks and systems. An interface generator must have a model with a finite number of elements. To be practical, the model should have a relatively small number of elements.

To this end, I examined the interaction model (Figure 2-1, which is also copied here) to look for a "bottle neck"—where a model could be developed with a relatively small number of elements and have the widest possible coverage.



A.1. Reasoning for Focusing on Functionality Input Needs

Users are diverse. Their needs, skills, characteristics, and preferences have a potentially infinite number of combinations. Even organizing general needs by disability type is problematic. Disability groups are hardly homogeneous; for example, people meeting the definition of legal blindness in the United States can range from people who can read large print to those who need significant magnification for reading to people who cannot see well enough to read visual text at all. A person can also be legally

blind if they have narrow central vision, even if their visual acuity is normal. People can also have varying needs along multiple dimensions, like people who are deafblind and many people who are aging and who may have a combination of physical and sensory needs. It is important to study this wide array of needs in order to come up with specific, practical techniques for users. The complexity and specific needs of users could be handled by a variety of UI generators. However, this wide variability of specific user needs does not lend itself to a fundamental strategy for interface modeling and generation.

Some model-based interface generation approaches have a task-level model that is used in conjunction with other models and knowledge about the specific domain in order to generate task-tuned interfaces. While a task model could potentially improve the generation process, there are too many possible tasks that a user might want to do with an arbitrary system. Building a universal and fundamental model on the task level is problematic because of the practically infinite variety of tasks that people may wish to complete.

A universal and fundamental model of input built on the interface component of the interaction model is more of a possibility than basing it on tasks or needs of users. Many mainstream interfaces are built using a relatively small number of controls, like those in a widget toolkit. This approach, providing accessibility by adjusting the interface layer is how assistive technology like screen readers work today. A screen reader transforms the widgets and other elements that comprise an interface into speech and accessible behaviors. This approach lags UI innovation however. When a new widget is added to the toolkit or where elements are used in atypical ways, assistive technology makes a poor transformation if at all. The UI widget transformation approach also has another potential issue—layering of interfaces. A person using a screen reader is using an audio interface in order to control a graphical user interface in order to complete a task. If the accessible interface were instead to directly be used to complete the task, this would avoid a layer of indirection, which might lead to confusion, etc. Even with these issues, this

approach remains the most common one today. It has allowed some people with disabilities to use otherwise inaccessible interfaces.

When examining the interaction model it was felt that basing a model on the needs of the functionality was the best approach to having a small, fundamental model. A system has particular input needs that a user must be able to fulfill in order for proper operation. The method that the person uses should make no difference to proper functioning as long as the functionality input needs were fulfilled.

A.2. The Model's Evolution

The functionality input need model evolved over time as part of an iterative process. During the process, a model was proposed and then tested and validated against real-world interfaces. In the first iteration, the functionality input need model consisted of a hierarchical model with the elements and structure shown in Figure A-.

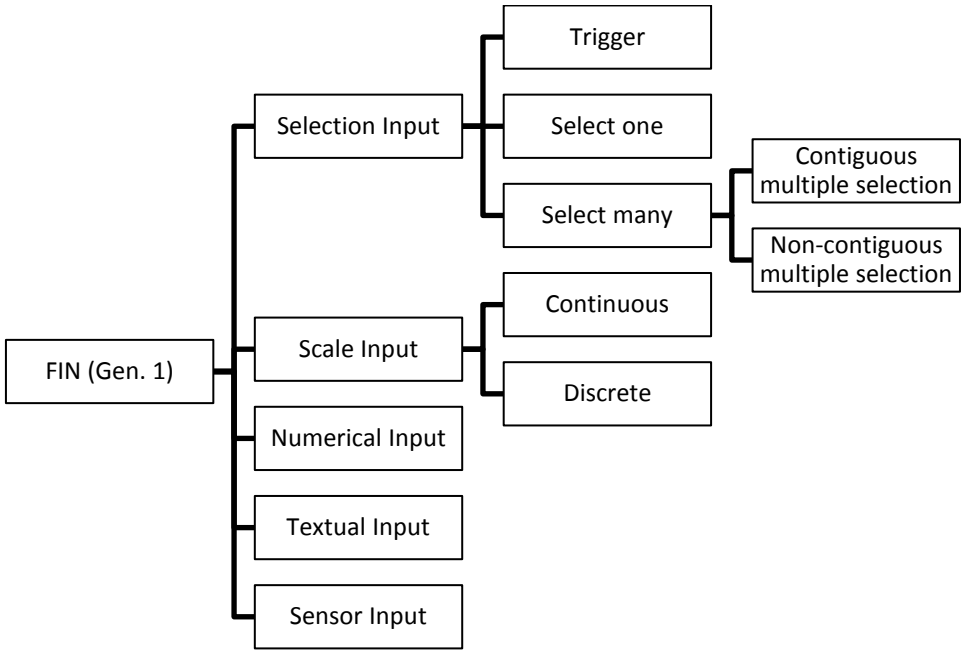


Figure A-1. A diagram of the first-generation FIN model.

This first-generation FIN model covered a wide range of potential inputs, but in applying the model to real-life interfaces (particularly those beyond standard web-based and office-related interfaces) it was realized that the *how* of the input was as much needed as *what* was needed. The *how* of input was addressed by adding FIN characteristics to the FIN model.

The first FIN characteristics that were identified were termed Sequential (where input had to be provided in a particular order), Temporal (where the time at which the input occurred was important), and Multifocal (where multiple inputs needed to be simultaneously provided). The addition of FIN characteristics to the model was important because the characteristics could have a drastic effect on what interactors were appropriate. However, the definitions of the original three FIN characteristics were not clear which lead to significant overlap and difficulty in applying them to real-world interfaces. In particular, the first three FIN characteristics were all related in different ways to the time dimension of using an interface.

The FIN model continued to evolve because the needs of the functionality were still felt to be the "bottleneck" of the interaction system. Both the FIN types and characteristics changed during the development process. One change from the first generation FIN model was the separation of the Command Input (which was initially known as a Trigger Input) from Value Inputs. Initially, I had made a command a type of selection input because users had to select from a list of possible commands. Selecting a value and selecting a command are two different things, however.

While continuing to develop the FIN model, the needs of the user kept creeping into the model. It was recognized that users could not just provide any arbitrary input, but that any input they were to provide had to be understandable to them. The FIN model at one point blended some of these vital user needs with the functionality input needs. The FINs under this blended model were defined and restricted to have user sensible values. The blended model got away from the original intent of the FIN model and the reality of input. A system needs information in the form that it expects—it does not matter to the

system if the value is sensible to the user. It is the job of the user interface to provide take the user's input and provide it in a form that the system needs.

A breakthrough came with the addition of the user-sensible input (USI) component of the model. Adding the USI layer released the unrealistic user-related restrictions on the FINs so that functionality could indeed require whatever was actually required. If a system needed a date and time in an ISO 8601 format (for example, the extended format date-time of 2007-01-06T15:20:55-06:00) or in an epoch time format (the number of seconds since some arbitrary epoch reference, such as the UNIX time representation of 1168118455) or in some other format, that is the FIN. The USI and transformation layer were responsible for taking something sensible to the user and converting it to something sensible to the user where the USI and FIN forms were not mutually sensible.

Appendix B. Example USIs & Interactors

The following tables list USIs individually and generic GUI widgets that can be used for each USI. Some USIs are not common types of input in mainstream interfaces and thus no generic GUI widgets are known (although there may exist some specialized inputs for very particular tasks). Those USIs with no generic GUI widgets are marked "N/A."

Some widgets require an explicit submission step, which might be a "Submit" button on the screen or the user pressing Enter on a keyboard or via other methods—these widgets are marked with "(+submit)." Similarly, some widgets require an "Add" button or other method of adding a new single input field, and these widgets are marked "(+add)."

Table B-1. Single-input USIs and generic GUI interactors that can be used for each USI.

Time Dependence	Command	Virtual Object	Numeric & Integer	Text	Partial Selection	Select (Complete)	Composite
None	Button	URL text entry field (+ submit) - or - Single File Picker (+ submit)	Spinner (+ submit) - or - Slider	Text input field (+ submit)	Combobox (+ submit) - or - Radio button set with "Other" text field (+ submit)	Radio button set - or - Dropdown - or - Listbox	Group of individual simple value input interactors (+ submit)
Momentary	Same as above	Same as above	Same as above (+ submit)	Same as above	Same as above	Same as above (+ submit)	Same as above
Continuous (Potential accessibility barrier)	Hold button	N/A Hold button for each file a person must select (User cannot easily enter arbitrary files quickly)	Slider Physical knob, steering wheel	N/A Speech recognition may be one of the fastest ways to input text	N/A	Set of Hold buttons	N/A Specialized driving controls (steering wheel, throttle pedal, brake pedal, etc.)

Table B-2. Multiple, unordered USIs and generic GUI interactors that can be used for each USI.

Time Dependence	Virtual Object	Numeric & Integer	Text	Partial Selection	Select (Complete)	Composite
None	Set of URL text entry fields (+ submit) - or - Multiple File Picker (+ submit)	Set of spinners (+ submit) - or - Set of numeric text input fields (+ submit)	Set of text input fields (+ submit) - or - Text input field (+ add) (+ submit)	Check box set with "Other" text field (+ add)	Checkbox set - or - Multiselect Listbox	Group of individual simple value input interactors (+ add new group) (+ submit)
Momentary	Same as above	Same as above	Same as above	Same as above	Same as above	Same as above
Continuous	N/A Multitouch screen with a set of hold buttons (one for each file a person can select from) (User cannot easily enter arbitrary files quickly)	N/A	N/A	N/A	N/A Multitouch screen with set of hold buttons	N/A

Table B-3. Multiple, ordered (Array) USIs and generic GUI interactors that can be used for each USI.

Time Dependence	Virtual Object	Numeric & Integer	Text	Partial Selection	Selection (Complete)	Composite
None	Ordered set of URL text entry fields (+ submit) - or - Ordered set of single File Pickers (+add) (+ submit)	Ordered set of spinners (+ submit) - or - Ordered set of numeric text input fields (+ submit)	Ordered set of text input fields (+ submit) - or - Text input field (+ add) (+ submit)	ListBuilder with a text field for adding choices to the list	ListBuilder - or - ListSorter (when maxCardinality = listLength)	Group of individual simple value input interactors (+ add new group) (+ submit)
Momentary	Same as above	Same as above	Same as above	Same as above	Same as above	Same as above
Continuous	N/A Multitouch screen with a set of hold buttons (one for each file a person can select from) (User cannot easily enter arbitrary files quickly)	N/A	N/A	N/A	N/A Multitouch screen with set of hold buttons	N/A

Some interesting things to note from these tables:

- In general, a momentary time-dependent USI can use the same interactor as a non-time-dependent input as long as an explicit method of submission is provided (so that the time of the input can be defined)
- There are few mainstream, generic interactors for continuous time-dependent USIs.
- Multiple input USIs typically require explicit submission.

Appendix C. USI+

The USI+ is the first set of task-related metadata that I would add to USIs to improve the output of an interface generator. With this additional information, usability would be improved over the "dumb" selection of widgets based only on the USI and FIN.

Because the USI+ is related to the task that a user is attempting, it has no bearing on the FINs of a system. Adding the USI+ information adds extra work for manufacturers, so the USI+ information was not included in the base USI model.

Table C-1. Additional characteristics that could be added to USIs to improve usability.

USI+ Characteristic	Data type	Description	What USIs does it apply to (inherited)
readonly	boolean	Specifies if a user is or is not allowed to edit the value.	Value USI
hidden	boolean	Specifies if an input may be hidden from a user because the field is not relevant to the current context.	All USIs
disabled	boolean	Specifies if an input is disabled or not. Disabled fields do not allow for entry or manipulation.	All USIs
suggestedValues	List of values	A list of suggested values that a user is most likely to pick from. A UI could optionally present this list as a set of quick options to some users. E.g., Number of copies {1,2,3,5,10} or other.	Primitive Value USIs
uniqueInput	boolean	Whether the input value is likely to be unique (never entered again). E.g., a one-time authorization code should not be stored as a potential autocomplete value for future use.	Value USI, Any USI group

USI+ Characteristic	Data type	Description	What USIs does it apply to (inherited)
valueClose	boolean	Whether the user is expected to set a value close to the current value or not.	Numeric USI
valueRange	boolean	Whether the user is expected to set the value over the entire range of values or not. This is not mutually exclusive with valueClose. For example, a volume control might have valueClose and valueRange both TRUE, where a user is expected to make small adjustments sometimes and large, quick adjustments at other times.	Numeric USI with upper & lower bound
password	boolean	Whether the use's entry should be treated as a password and hidden from those who might overlook or eavesdrop.	Primitive Value USI
logicalOrder	boolean	Whether the list of enumerated values is already provided in a logical order that is sensible to users (e.g., the days of a week) or not (e.g. list of ice cream flavors). The order is logical if given any arbitrary value, a person will know whether to go up or down in the list to get to a second arbitrary value. Note that the order of alphabetical values may not be immediately clear to some people.	Select USI
accuracyValue	decimal	Expected level of accuracy of the user's input. Smaller values give a smaller window of accuracy, which means that a user must enter a more accurate value.	Numeric USI
accuracySpeed	decimal	Expected speed in seconds of input of the desired accuracy. E.g., a SelectUSI with a short, quick accuracySpeed may be presented as a set of large radio buttons rather than as a drop-down list.	Primitive Value USI

USI+ Characteristic	Data type	Description	What USIs does it apply to (inherited)
cohesion	"strong", "normal", "weak"	From URC (ISO 24752:5). Cohesion, is a hint to the interface generator about how strongly sibling elements within a group relate to each other. There are three levels of cohesion: • "strong" – sibling elements have very strong cohesion, to the exclusion of other groups. Other groups should not be presented with the "strong" group. • "normal" – the default cohesion denotes that a group logically fits together and should generally be labeled as a group. • "weak" – sibling elements can be grouped together but do not need to be delimited or labeled as a group	Any Group
opposites	boolean	Whether two values are opposites of each other. E.g., it might make sense to make opposites a switch and non-opposites a drop-down.	Select USI with only 2 values
sensitive	boolean	Whether or not the submission of an input element or group will result in financial, legal, or other consequences that may result in the loss of data or property. E.g., a sensitive command such as "delete" that cannot be undone may have a confirmation dialog.	All USIs, Any USI group
inputmode	enumerated	From HTML 5.1, a hint on which keyboard to display	Text USI

USI+ Characteristic	Data type	Description	What USIs does it apply to (inherited)
preview	boolean	Whether or not it makes sense for at least some users to be able to preview the results of the value they input, especially as they browse or step up/down through the choices (e.g., hearing the volume or viewing the channel while stepping up or down). Since the user is browsing, there are no ill effects of selecting a "wrong" choice at least temporarily.	Numeric USI, Select USI
liveFilter	boolean	Whether a system can provide an enumerated list of valid values that are filtered as a person types.	Text USI
help	resource	Helpful tips or other information that can be displayed at the user's discretion or when the PAUI senses that a person is having difficult with particular input.	

Appendix D. Loop Interface

[*] The Loop interface is a user interface method that allows for quick, efficient exploration and usage of user interfaces by people who cannot see. The interface method is a circular direct-selection interface where many user interface elements are arranged around the edges of a touch panel or touch screen. This makes a circular list of elements that users can touch to get speech output and then activate when they find the desired element. This arrangement can be explored easily but also allows users to efficiently and directly select elements on familiar interfaces.

D.1. Introduction to the Problem

Using a touchscreen device is challenging for people with low and no vision because they cannot easily see any of the controls or other elements on the screen. Currently, two general categories of accessible interfaces for touchscreen devices have been developed to allow people to get speech output and activate elements: navigation and direct exploration/selection. With a *navigation interface*, a person must move or navigate through elements one at a time. The navigation paradigm is frequently used but is inherently inefficient because users must navigate past options they do not want until they finally get to an element with which they want to interact. The user often spends more time navigating than doing what he or she really wants to do with the interface. With *direct exploration and selection*, a person touches elements on the screen to hear them. When the desired element is found, it can be activated. One major

[*] This appendix is based on a public disclosure sponsored by the Linux Defenders program.

Jordan, J. B. (2015). A circular direct-selection interface for non-visual use. *IP.com Prior Art Database*, Disclosure Number: IPCOM000241004D. Available: https://priorart.ip.com/IPCOM/000241004

problem with this method of activation is that a user has to search for elements by touch on a featureless screen. With an unfamiliar interface, users may have difficulty finding what they want. This difficulty is exacerbated with large screen sizes, with small elements, or with widely spaced elements.

The Loop interface, a circular direct-selection mode, is a method that allows for direct exploration and selection of an interface in a common, systematic pattern. It allows for more efficient use for both novel interfaces and interfaces with which people are familiar.

D.2. Description

The user interacts with a system, which is comprised of: a touch panel sensor, optional buttons or other user inputs, an optional display associated with the touch panel sensor (together forming a touchscreen), a microprocessor, memory, and a method for providing audio output (e.g., speakers, a headphone jack, or wireless audio connection). The system might take the form of a phone, tablet computer, or other device. A schematic diagram of the system is shown in Figure D-1.

The Loop interface mode, which enables the user to easily find and quickly activate components of a user interface, is implemented in software that runs on the system.

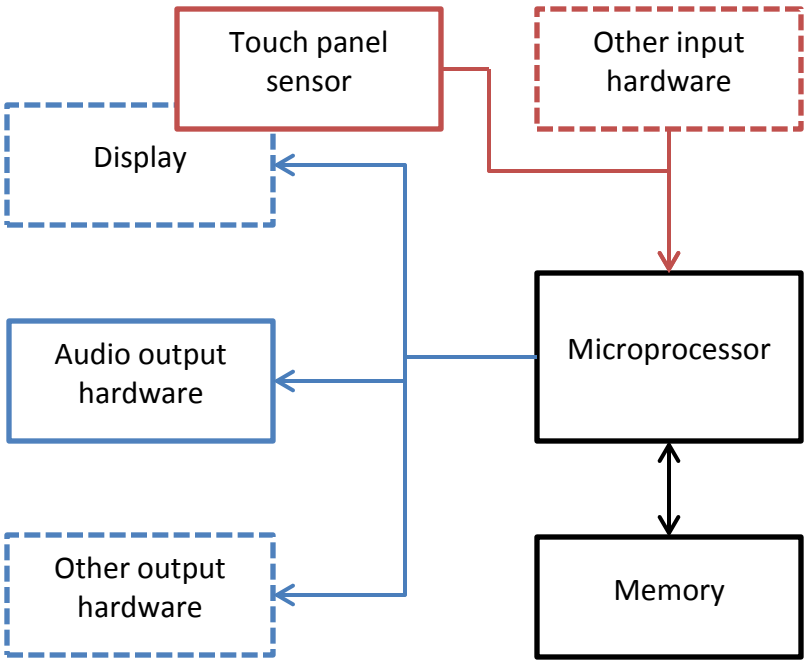


Figure D-1. A diagram of the system. For input, the system has a touch panel sensor and other optional input hardware. The touch panel sensor may directly overlay an optional display forming a touchscreen. For output, the system has at least audio output hardware and may have an optional display (visual or tactile, such as dynamic braille). The input and output components are connected to a microprocessor, which runs software that may be loaded into memory.

D.2.1. Description of the Loop interface

In the Loop interface, most user interface elements (such as buttons, checkboxes, text input fields, and text) are touch panel areas that are arranged along the edges of the touch panel sensor (which might be a touchscreen). In effect, this makes all of the elements into a circular list around the touch panel. Users can follow the edges of the touch panel with a finger and get to all of the elements in the circular list. This consistent pattern makes it easy for users to search for elements without having to try a variety of search patterns and techniques like they might have to with a typical system that allows for direct exploration and selection. See Figure D-2 for a generic representation of the circular direct-selection mode on a touchscreen of a mobile device.

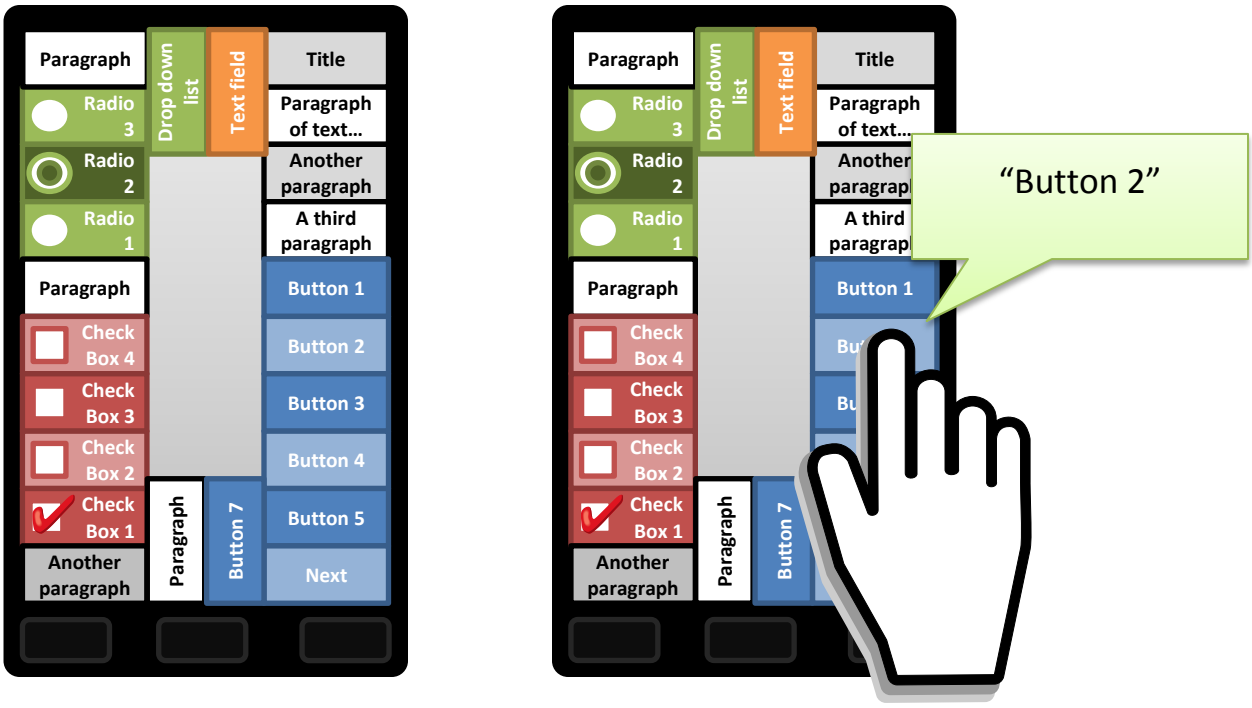


Figure D-2. A representation of a Loop interface on a mobile touchscreen device. On the left, all of the (generic) elements of the interface are arranged in a circular list around the edges of the screen. When the user touches an element (right), the label or other information about the element is provided to the user in speech output.

Exploring an interface: With encountering a novel interface, it is envisioned that a user will slide or drag his or her finger around the edge of the touch panel and listen to the speech output. When the user's finger is on the area that represents a particular element, information about that element will be provided in speech. For users with low vision, the element information might be displayed on a screen in large text. For users who read braille, the element information might be provided on a dynamic braille display.

Activating a user interface element: When the user finds the element he or she wants, the user makes an action to activate or otherwise manipulate that element. For example, the user can activate a button, toggle a checkmark in a checkbox, or increment a numeric value. The action that a user takes to activate or manipulate an element may vary depending upon the system capabilities and the element that is focused or under the user's finger. For example, the user might:

- Press a separate physical button such as an Enter key
- Speak a command
- Tap the on-screen element again
- Tap a dedicated area of the screen
- Double-tap anywhere on the screen
- Press harder on the on-screen element with pressure sensing touchscreens
- Make a swipe up or swipe down gesture to increment or decrement a numeric value
- Make some other gesture on the touchscreen

The activating/manipulating user action may take place while the user's finger is in contact with the desired element. Alternatively and preferably, the user may lift his or her finger leaving the last-touched element in focus. The element in focus could then be activated with a subsequent user action.

Using a familiar interface: With continued use of the circular direct-selection mode, a user may learn the general location of controls of interest in the circular list for a particular interface. When the user is familiar with an interface, the user can directly point to the control in the circular list, confirm through the speech output (or other output) the desired element, and then activate the desired the element. This direct selection technique is faster than exploring the interface by touch or navigating it with keystrokes or gestures.

When there are too many elements to fit on one screen: For pages with a relatively small number of elements, all elements can fit in the circular list on a single "screen." When there are more elements in the interface than can fit in one circular list, the user must be able get to the overflow elements. Two solutions are proposed for this problem:

- The last element in a circular list that overflows might be a "Next Screen" or "Page Down" button. The user could then activate that button to move to a next screen which contains additional elements in the circular list that did not fit on the earlier circular list

screen. A “Previous Screen” button could be provided to go back to previous circular lists.

- A user might make or continue a sliding or dragging gesture across the boundary between the last and first item of the circular list. With this gesture, the system would automatically change to the next group of circular list items. A gesture in the reverse direction would go to a previous group of circular list items. This gesture has the benefit of being like that of a jog wheel. By convention, turning a jog wheel clockwise goes to later elements, and turning it counterclockwise goes to earlier elements. With each circumnavigation of the touch panel, a user would be presented with a new screen-worth of interface elements until they have explored the entire interface.

D.2.2. Potential Additional Loop Interface Features

It is envisioned (but not required) that the circular list would start in the upper right corner of the screen and continue clockwise around the screen. This arrangement has the following potential benefits:

- If there are few elements in an interface, they would all fit along the long edge of a smartphone screen held in portrait orientation.
- To progress through more elements, the user would slide their finger clockwise. This clockwise motion is a familiar gesture/direction from using jog wheels for navigation.

Ensuring the user knows when they have reached the end of an interface: It is helpful for many users to know when they have reached the first or last elements in a list. The beginning and end of the list, which might be on different “screens” because of overflow, might have extra elements: one denoting the top of the list and the other denoting the bottom of the list.

Dedicated controls: The operating system and many apps may have a set of common or global controls to which it might be useful for the user to have easy, quick access. Dedicated controls might

include a Cancel or Back function, navigation controls, an activation button, a “Where am I?” button to allow people to get an overview of the current page, speech output settings menu, etc. Dedicated controls could be placed in a consistent location on the screen and might be user configurable. The circular list of items might be “interrupted” by these controls in dedicated locations. Three beneficial arrangements have been identified (which are not mutually exclusive):

- *Corners:* The dedicated controls could be place at the corners of the screen. These locations are easy to find tactilely. The dedicated control in each corner would interrupt the circular list.
- *Along a short edge:* The dedicated controls might also be placed along the short edge of the screen (for example the top of a smartphone). With a limited number of dedicated controls, their position would be easy enough to find without sight. By having the dedicated controls at the top (where the starting point is the top right corner), they would only interrupt the circular list at the end of each screen-worth of elements.
- *In the middle of the screen:* A limited number of dedicated controls may be placed in the middle of the screen—not in the loop of general interface elements.

See Figure D-3 for a representation of the first two dedicated control locations.

Figure D-3. Two possible alternatives for dedicated controls. Dedicated controls in this figure are yellow and are labelled: “Where am I?”, “Back”, “Next”, and “Prev.” The figure on the left shows the dedicated controls in the corners. Notice that they interrupt the circular flow of the circular list of elements. The figure on the right shows the dedicated controls all at the top of the screen in a toolbar-like arrangement.

Hysteresis: It can be difficult to use a touchscreen without sight because a person does not know where edges of controls are. Button hysteresis is a technique that effectively makes the currently touched control bigger. When on an element, the user must drag slightly further outside of the current element to move to a new element. Once in the new element, it is then effectively larger, so the user would have to drag slightly further outside the new element to get back to the previous element. This prevents the focus/speech from moving quickly back and forth between elements if the user was touching exactly on the border and had a slight tremor.

Element resizing: Depending upon the number of elements in the circular list, the elements might resize. When there are fewer elements, each element might be bigger to allow for more accurate

targeting. This is envisioned to be a user-specific setting. In implementations, there is likely an upper limit to the size of elements so that users do not have to drag "too far."

Flexibility: The circular list arrangement of controls might be one arrangement of controls from which users can select. It does not preclude a different interface arrangement with which a user may already be familiar. For example, a person might use the circular list for most interfaces, but switch to a standard on-screen QWERTY keyboard with which they have a lot of experience when they need to enter text. Note also that the circular list arrangement does not preclude navigating an interface if a user would rather do that. For example, to navigate, a user might press dedicated controls or make swiping gestures across the screen (or in the middle where the circular list is not located).

D.3. Potential Advantages of the Loop Interface

Navigation interfaces require a person to step through elements to find the one they want. This might be done with navigation buttons, gestures, or a jog wheel. Some of these methods require many motions and may be fatiguing. Other methods may require less motion, but still require a person to potentially navigate through a long list of elements that the person does not wish to choose at this time. Where users may use an interface often enough to learn the position of elements on screens, this invention is superior to using navigation controls to move through the list of elements. The user can directly point to the location of the desired element, which is faster than navigation.

With a direct exploration and selection interface, a person touches elements on the screen to hear them. This potentially allows for quicker access than navigation, but introduces a new problem: the user with little or no vision does not know where elements are located on the screen. The user must search a featureless surface with a finger and audio and might pass by or miss the element they desire. The spatial re-arrangement of elements into a circular list with the current invention allows for a consistent search

and exploration strategy that guarantees that a user will not miss an element if they cover the entire circular list.

Note that the Trace Research & Development Center at the University of Wisconsin-Madison developed a technique called Speed List in the 1990s, which has some similarities to the current invention. Some of the claims and descriptions of U.S. patent number 6,049,328 concerns this Speed List mode. When Speed List mode was activated, all of the elements on the screen were arranged from top to bottom along the left edge. Users could slide their finger down the left edge of the screen to get audio output as they touched each element. They would then press a physical button to activate the desired element. The number of elements in the list was limited by the number of elements that could be placed along the left edge of the screen. This was a reasonable limitation for kiosk systems but is unacceptable for personal devices and interfaces. The circular list of elements described in this invention allows for overflow onto additional “screens,” which circumvents the Speed List limitations.

Appendix E. Questionnaires

E.1. Initial Questionnaire

This questionnaire was asked after the participant consented to the research.

Demographics

1. Age: ____
2. Gender
    a. Female
    b. Male
3. Living arrangement:
    a. Alone, no help
    b. With significant other
    c. Alone, with occasional assistance
    d. Alone, with frequent assistance
    e. Facility with occasional help (e.g. assisted living centers)
    f. Facility with frequent help (e.g. nursing home)
4. Which hand is your dominant one / do you use most often?
    a. Right
    b. Left
    c. Both
    d. Do not use hands regularly
5. Ethnic background:
    a. American Indian or Alaskan Native
    b. Asian or Pacific Islander
    c. Black, not of Hispanic origin
    d. Hispanic
    e. White, not of Hispanic origin
    f. Other
6. Education:
    a. Less than high school
    b. High school
    c. Beyond high school

Disability History

7. Disability: _____
8. Number of years since onset of disability: ____ Comments: _____
9. Medication that may affect your performance during the session _____
10. Other conditions:
    a. Arthritis
    b. Diabetes
    c. Cardiovascular disease
    d. Hearing Impairment
    e. Obesity
    f. Osteoporosis
    g. Other

Technology Use

11. How often do you use a computer?
    a. No experience
    b. Rarely
    c. Occasionally (somewhat regular use once/month)
    d. Frequent (use at least once/week)
    e. Daily use
12. Computer activities – What activities do you typically use the computer for (e.g. internet, playing games, check email, etc.)?
    _____
13. How often do you use a mobile device like a smartphone or tablet?
    a. No experience
    b. Rarely
    c. Occasionally (somewhat regular use once/month)
    d. Frequent (use at least once/week)
    e. Daily use
14. What activities do you typically use the smartphone or tablet for?
    _____
15. How often do you use a cell phone?
    a. No experience
    b. Rarely
    c. Occasionally (somewhat regular use once/month)
    d. Frequent (use at least once/week)
    e. Daily use
16. Is it a smartphone or a feature phone (non-smartphone)?
17. If a smartphone: What type of smartphone do you use?

   a. iPhone
   b. Android
   c. Windows Phone
   d. Other: _____

18. Do you regularly use a screen reader?
19. If yes: Which screen readers do you use regularly?
   a. VoiceOver (iPhone/iPad iOS)
   b. Talkback (Android)
   c. Other mobile screen reader: _____
   d. VoiceOver (Mac OS X)
   e. JAWS
   f. Windows-Eyes
   g. NVDA
   h. Zoom Text
   i. Other computer screen reader: _____
20. Do you use any [other] assistive or adaptive devices? (circle all that apply)
   a. Magnifier
   b. Reading devices
   c. Talking appliances
   d. Switch devices
   e. Voice activation software
   f. Mouthstick
   g. Scanning Programs
   h. Splint/Brace
   i. Ambulation device (e.g. cane, walker, wheelchair)
   j. Other _____

E.2. Questions from Phase 1

These questions were asked after every time a person tried a pair of interfaces comparing two different interactors.

Which interface of the pair was your favorite? _____
On a scale from 1 to 7, with 1 being 'terrible' and 7 being 'excellent', how would you rate the first interface you tried? [*This question only asked the first time in each task block*]

1 2 3 4 5 6 7

How would you rate the second interface? (1 = 'terrible' & 7 = 'excellent')

1 2 3 4 5 6 7

Why did you like the first/second interface better than the other one? _____

E.3. Ranking Questions for the Three iPod Interfaces

Asked when ranking the three iPod interfaces (Item, List, and Loop) at various points.

I wanted to get your impressions of the three interfaces on the iPod Touch: the item, list, and loop interfaces.

1. Overall, which of the three iPod interfaces is your favorite? Why is it your favorite?

Circle one: **Item** **List** **Loop**

2. Which is your second favorite interface on the iPod? Why do you like it better than the *item/list/loop [least favorite]* interface?

Circle one: **Item** **List** **Loop**

E.4. Questionnaire for Second Session

These questions were asked after a person trained on all of the interfaces on the second session. Adapted from Webster & Martocchio (1992).

The following questions ask you how you would characterize yourself when you use computers, smartphones, tablets, and other technology. On a seven-point scale, with 1 being “strongly disagree” and 7 being “strongly agree”, pick the number that best matches a description of yourself when you interact with technology.

		Strongly Disagree						Strongly Agree
Play 1	Spontaneous	1	2	3	4	5	6	7
Play 2	Unimaginative	1	2	3	4	5	6	7
Play 3	Flexible	1	2	3	4	5	6	7
Play 4	Creative	1	2	3	4	5	6	7
Play 5	Playful	1	2	3	4	5	6	7
Play 6	Unoriginal	1	2	3	4	5	6	7
Play 7	Uninventive	1	2	3	4	5	6	7

Adapted from (Venkatesh, 2000). Lrn questions are original.

In the following questions, when I say "computers" I am speaking broadly about computing devices such as smartphones and tablets as well as laptop and desktop computers.

For the following seven-point scale, with 1 being "strongly disagree" and 7 being "strongly agree", how much do you agree with the following statements?

		Strongly Disagree						Strongly Agree
CANX 1	Computers do not scare me at all.	1	2	3	4	5	6	7
CANX 2	Working with computers makes me nervous	1	2	3	4	5	6	7
CANX 3	I do not feel threatened when others talk about computers	1	2	3	4	5	6	7
CANX 4	It wouldn't bother me to take computer courses	1	2	3	4	5	6	7
CANX 5	Computers makes me feel uncomfortable	1	2	3	4	5	6	7
CANX 6	I feel at ease in a computer class	1	2	3	4	5	6	7
CANX 7	I get a sinking feeling when I think of trying to use a computer	1	2	3	4	5	6	7
CANX 8	I feel comfortable working with a computer	1	2	3	4	5	6	7
CANX 9	Computers make me feel uneasy	1	2	3	4	5	6	7
LRN 1	I enjoy learning how to use computers and other [accessible] technology	1	2	3	4	5	6	7
LRN 2	I find new [accessible] technologies easy to learn	1	2	3	4	5	6	7

E.5. Post-task Question

This question was asked after every task in Phase 2. Single Ease Question (SEQ) from Sauro & Dumas (2009)

Overall this task was (*very difficult* 1 – 7 *very easy*): _____

E.6. Phase 2 Post-Interface Questionnaire

These questions were asked after participants had completed all of the tasks on a given interface.

OK, now that you are finished with those tasks, I would like to ask you some questions. I would like you to consider all of the thermostat/copier tasks together when answering.

For all of these questions, I want to know how strongly you agree or disagree with the following statements. Please answer on a 7-point scale with 1 being "Strongly disagree" and 7 being "Strongly agree."

Adapted from the System Usability Scale (SUS) from Brooke (1996).

		Strongly Disagree						Strongly Agree
SUS 1	[If I couldn't see,] I think I would like to use this system frequently	1	2	3	4	5	6	7
SUS 2	I found the system unnecessarily complex	1	2	3	4	5	6	7
SUS 3	I thought the system was easy to use	1	2	3	4	5	6	7
SUS 4	[If I couldn't see,] I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5	6	7
SUS 5	I found the various functions in this system were well integrated	1	2	3	4	5	6	7
SUS 6	I thought there was too much inconsistency in the system	1	2	3	4	5	6	7
SUS 7	[For people who cannot see,] I would imagine that most people would learn to use this system very quickly	1	2	3	4	5	6	7
SUS 8	I found the system very cumbersome to use (awkward)	1	2	3	4	5	6	7
SUS 9	I felt very confident using the system	1	2	3	4	5	6	7
SUS 10	I needed to learn a lot of things before I could get going with this system	1	2	3	4	5	6	7

Adapted from Venkatesh (2000).

		Strongly Disagree						Strongly Agree
ENJ 1	I find using the system to be enjoyable	1	2	3	4	5	6	7
ENJ 2	The actual process of using the system is pleasant.	1	2	3	4	5	6	7
ENJ 3	I have fun using the system.	1	2	3	4	5	6	7
EOU 1	My interaction with the system is clear and understandable	1	2	3	4	5	6	7
EOU 2	Interacting with the system does not require a lot of my mental effort	1	2	3	4	5	6	7
EOU 3	I find the system to be easy to use	1	2	3	4	5	6	7
EOU 4	I find it easy to get the system to do what I want it to do.	1	2	3	4	5	6	7

OK, now I have some other questions for you.

1. What do you think about the interface you just tried?
    a. Difficulties participant encountered
    b. Things the participant would change
2. [If you couldn't see,] Would you like to use a system like this again in the future? If not, why not?

E.7. Ranking Questions for All Four Interfaces

Asked when ranking all four interfaces when participant complete all interface for a particular device.

OK, you just tried four interfaces: Physical, Item, Loop, List. Please rank the interfaces from best to worst:

Probes: What was your favorite interface? Second favorite?

_____ Physical: "The __nth__ interface that you tried. The interface on the actual device"

_____ Item (R, Red icon): "The Item interface is the __nth__ interface that you tried."

_____ Loop (S): "The Loop interface was the __nth __ interface that you tried."

_____ List (T): "The List interface was the __nth__ interface that you tried."

How strongly do you prefer the __Loop/List__ interface to the __List/Loop__ interface?

0. I actually don't prefer one to the other
1. I prefer the Loop/List interface a little bit, but would not mind if I had to use the other one
2. I moderately prefer the Loop/List interface over the other one, but would use the other one if I had to
3. I strongly prefer the Loop/List interface over the other one, and do not want to use the other one unless it were the only interface available to me

Why do you like the __Loop/List__ interface better than the __List/Loop__ interface?

How strongly do you prefer the __List/Item__ interface to the __Item/List__ interface?

0. I actually don't prefer one to the other
1. I prefer the List/Item interface a little bit, but would not mind if I had to use the other one
2. I moderately prefer the List/Item interface over the other one, but would use the other one if I had to
3. I strongly prefer the List/Item interface over the other, and do not want to use the other one unless it were the only interface available to me

Why do you like the __List/Item__ interface better than the __Item/List__ interface?

Why was the <*favorite interface*> your favorite?

Why was the <*last-ranked interface*> your least favorite?

E.8. Final Questions

Asked at the end of the entire experiment.

Now, I would like you to think about the three interfaces you tried on the iPod Touch today—the item, loop, and list interfaces.

F1. Of the interfaces on the iPod Touch that you tried today, which one did you like best? Why was it your favorite?

Circle one: **Item** **Loop** **List**

F2. Which interfaces was your least favorite? Why did you like it least?

Circle one: **Item** **Loop** **List**

F3. Which interface do you think most blind people would like? Why?

Circle one: **Item** **Loop** **List**

F4. Which interface do you think would work best for blind people who may not be as familiar with smartphones and computers as you are? Why?

Circle one: **Item** **Loop** **List**

F5. Do you have any other thoughts about the experiment or the interfaces that you tried?

Appendix F. Scripts

F.1. Introduction to the Experiment (Phase 1)

Blindfolded participants only

We are interested in studying interfaces that may be helpful for people who are blind. We have had a number of people who are actually blind come in and try things, but we also want to test people who are new and unfamiliar with being blind and some of the techniques and strategies they might use. When you do the different tasks with the interfaces, we will need you to wear this blindfold.

- Show the participant the blindfold.

It can get kind of disorienting wearing a blindfold, so we will take breaks where you can take off the blindfold. Please don't take off the blindfold without checking with me first. I need to make sure that things are covered so that you don't see them when you take off your blindfold. Also, please don't try to look around or under the blindfold when doing the tasks.

There are several parts to today's study. First, we will need to get your consent to take part in the study and then I will ask you some questions. After the questions, you will be trying different things and I'll be asking you questions along the way. First, you will have some basic training. Then you will try very simple tasks on different pairs of interfaces on the iPod Touch so we can see which ones you like the best.

We may ask you to come back for a second session, which will be much like the first with different tasks.

It is important to remember throughout the study that we are not testing you. We are testing the products and the interfaces to see which things work better for different people. If something is difficult or frustrating, let me know! We want that information so that we can improve devices in the future. Also, if you feel like you need a break, let me know and we can take a break at the next opportunity.

- Start recording
- Go over consent forms
- Administer questionnaire (E.1)

Many of the interfaces that you will be testing today are self voicing. First, I would like to find a good volume for you.

- Set volume with participant

OK, now that we have found a good volume, I would like to find a good speech rate—how fast the voice speaks. To do that, I will have the system read a phrase, which you will listen to and then repeat back to me. I will do this a number of times. OK, are you ready?

- Use application to set speech rate

OK, now I'm going to have you double-tap the screen several times. Go ahead and double-tap now.

- Use application to set double-tap rate

OK, now we will do some basic training on the different systems we will use today.

Note: The Item, List, and Loop interfaces were presented in random order.

F.1.1. Item interface initial training

Now, I want to introduce you to an interface on the iPod Touch called the ITEM interface. We will use this later to control both a copier and a thermostat.

With an item interface, everything on the screen is an item. These items may be arranged in different locations on the touchscreen.

There are two ways to move through the items. You can either EXPLORE the items of the interface by dragging your finger around, or you can STEP THROUGH the items of an interface by making swiping gestures.

Let's try the item interface now on the iPod Touch. I'd like you to keep it on the table while you are using it so that it is easier for the overhead camera to record it.

Right now, there is a set of buttons on the touchscreen—one button for each letter of the alphabet.

Go ahead and drag your finger around on the screen to explore the items.

- [Participant drags finger around the screen and listens to different buttons]
- If participant does not drag: You drag on the screen by putting your finger down on the screen and sliding it around without picking it up. Repeat instructions.

You can also make quick swiping gestures to the right or left to step through the items on the screen. To go to the next item on the screen, swipe your finger quickly from left to right anywhere on the screen.

Go ahead and try that now—swipe quickly from left to right to go to the next item on the screen.

- [Participant swipes to the right]

- If participant does not swipe fast enough: That wasn't a fast enough swipe. Try swiping from left to right (right to left) more quickly.
- If participant swipes in the wrong direction: That was the wrong way. Try starting at the left (right) side of the screen and swiping to the right (left) quickly.

You can step to all of the items on a screen one-by-one by swiping. If you pass something you actually wanted, you can swipe in the opposite direction—swipe to the left—and it will go back to the previous item.

Go ahead and swipe left now.

- [Participant swipes to the left]

So, with the ITEM interface, you can drag your finger around the screen. Or, you can use the stepping technique.

If there are too many items to fit on the screen, like the alphabet, you can scroll the screen up to get to the other items that are down below the bottom of the screen.

To go to items below the bottom of the screen, you should swipe UP with three fingers. You can think of this as using three fingers to push the page up and away from you.

When you do this, it will push up one full or complete screen. Go ahead and try a three-finger swipe up now.

- [Participant does three-finger swipe upwards]
- If participant does not use 3 fingers on the screen OR does not swipe fast enough: That didn't work. Try again by touching the screen with three fingers and quickly swiping upwards (downwards).

To go to earlier items, use the opposite gesture—a three-finger swipe down. Try that now.

- [Participant does three-finger swipe downwards]

Now is a good time to practice exploring & moving around the screen on your own. Let me know when you'd like to go on.

- [Participant tries navigating & scrolling on own and then indicates he or she is done.]

OK, so now you know how to move around the screen.

But, you also need to be able to DO things on the screen. The type of things that you can do depends on the item that is highlighted on the screen.

For many items, you can activate them by rapidly double-tapping anywhere on the screen.

Go ahead and find the H (as in Hotel) button on the screen.

- [Participant finds H button]

Participant does not find the right button: That's not the right button. Repeat the task.

Now, activate the H button by double-tapping.

- [Participant double-taps]
- If participant does not double-tap fast enough: That didn't work. You will need to be faster when you double-tap the screen. Repeat the "find the button task." Repeat the "activate" task.

Now, find and activate the T (as in Tango) button.

- [Participant should find the T button and then double-tap]
- Have the person find letter buttons (use the random letter sheet) until they find and activate 5 in a row correctly.

Different items on the screen may allow for different gestures. For example you might quickly swipe up or down with a finger to change a number a little bit.

You will learn these other gestures later when we try out different interfaces.

F.1.2. List interface initial training

Now, I want to introduce you to an interface on the iPod Touch called the LIST interface. We will use this later to control both a copier and a thermostat. With a list interface, everything is arranged in a list that goes down the touchscreen.

There are two ways to move through the list on the screen. You can either EXPLORE the list by dragging your finger up and down the screen, or you can STEP THROUGH the list by making swiping gestures.

Let's try the list interface now on the iPod Touch. I'd like you to keep it on the table while you are using it so that it is easier for the overhead camera to record it.

Right now, there is a set of buttons on the touchscreen—one button for each letter of the alphabet.

Go ahead and drag your finger down and up the screen to explore the items.

- [Participant drags finger down and up the screen and listens to different buttons]
- If participant does not drag: You drag on the screen by putting your finger down on the screen and sliding it around without picking it up. Repeat instructions.

You can also make quick swiping gestures to the right or left to step through the list. To go to the next thing in the list, swipe your finger quickly from left to right anywhere on the screen.

Go ahead and try that now—swipe quickly from left to right to go to the next thing in the list.

- [Participant swipes to the right]
- If participant does not swipe fast enough: That wasn't a fast enough swipe. Try swiping from left to right (right to left) more quickly.
- If participant swipes in the wrong direction: That was the wrong way. Try starting at the left (right) side of the screen and swiping to the right (left) quickly.

You can step to all of the things in the list one-by-one by swiping. If you pass something you actually wanted, you can swipe in the opposite direction—swipe to the left—and it will go back to the previous thing in the list.

Go ahead and swipe left now.

- [Participant swipes to the left]

So, with the LIST interface, you can drag your finger up and down the screen. Or, you can use the stepping technique.

If there are too many things in the list to fit on the screen, like the alphabet, you can scroll the screen up to get to the other things in the list that are down below the bottom of the screen.

To go to things below the bottom of the screen, you should swipe UP with three fingers. You can think of this as using three fingers to push the page up and away from you.

When you do this, it will push up one full or complete screen. Go ahead and try a three-finger swipe up now.

- [Participant does three-finger swipe upwards]

To go to earlier things in the list, use the opposite gesture—a three-finger swipe down. Try that now.

- [Participant does three-finger swipe downwards]

Participant does not use 3 fingers on the screen OR does not swipe fast enough: That didn't work this time. Try again by touching the screen with three fingers and quickly swiping upwards (downwards).

Now is a good time to practice exploring and moving around the screen on your own. Let me know when you'd like to go on.

- [Participant tries navigating & scrolling on own and then indicates he or she is done.]

OK, so now you know how to move around the list.

But, you also need to be able to DO things on the list. The type of things that you can do depends on the thing that is highlighted in the list.

For many things, you can activate them by rapidly double-tapping anywhere on the screen.

Go ahead and find the H (as in Hotel) button on the screen.

- [Participant finds H button]
- If participant does not find the right button: That's not the right button. Repeat the task.

Now, activate the H button by double-tapping.

- [Participant double-taps]
- If participant does not double-tap fast enough: That didn't work. You will need to be faster when you double-tap the screen. Repeat the "find the button task." Repeat the "activate" task.

Now, find and activate the T (as in Tango) button.

- [Participant should find the T button and then double-tap]
- Have the person find letter buttons (use the random letter sheet) until they find and activate 5 in a row correctly.

Different things on the list may allow for different gestures. For example you might quickly swipe up or down with a finger to change a number a little bit.

You will learn these other gestures later when we try out different interfaces.

F.1.3. Loop interface initial training

Now, I want to introduce you to an interface on the iPod Touch called the LOOP interface. We will use this later to control both a copier and a thermostat.

With a loop interface, everything is arranged in a loop around the edges of the screen. This allows you to drag your finger in a loop to get to things.

There are two ways to move through the loop on the screen. You can either EXPLORE the loop by dragging your finger around the edges of the screen, or you can STEP THROUGH the loop by making swiping gestures.

Let's try the loop interface now on the iPod Touch. I'd like you to keep it on the table while you are using it so that it is easier for the overhead camera to record it.

Right now, there is a loop interface on the screen with buttons—one button for each letter of the alphabet. The upper right corner is the starting corner.

First, start with your finger in the upper right corner of the screen and then drag your finger down and around the edges of the screen clockwise.

- [Participant drags finger around the screen and listens to different buttons]
- If participant does not drag: You drag on the screen by putting your finger down on the screen and sliding it around the edges of the screen without picking your finger up. Repeat instructions.
- If participant does not stay on the edges of the screen, but deviates into the middle: With the loop interface, you follow the edges of the screen. You don't need to drag your finger into the middle.
- If participant drags the wrong way: To get to later (earlier) things in the loop, you need to drag your finger the other way—clockwise, not counterclockwise (counterclockwise, not clockwise).

You can also go counterclockwise.

- [Participant drags finger around the screen counterclockwise and listens to different buttons]

You can also make swiping gestures to the right or left to step through the loop. To go to the next thing in the loop, swipe your finger quickly from left to right anywhere on the screen.

Go ahead and try that now—swipe quickly from left to right to go to the next item on the screen.

- [Participant swipes to the right]
- If participant does not swipe fast enough: That wasn't a fast enough swipe. Try swiping from left to right (right to left) more quickly.
- If participant swipes in the wrong direction: That was the wrong way. Try starting at the left (right) side of the screen and swiping to the right (left) quickly.

You can step to all of the things in the loop one-by-one by swiping. If you pass something you actually wanted, you can swipe in the opposite direction—swipe to the left—and it will go back to the previous thing in the loop.

Go ahead and swipe left now.

- [Participant swipes to the left]

So, with the LOOP interface, you can use the looping movement around the edges of the screen. Or, you can use the stepping technique.

If there are too many things to fit into one loop around the screen, like the alphabet, you can continue to go to the next page of the loop, by just continuing to loop around the screen as many times as you need to. By looping around the screen multiple times, you can get to all of the things in the loop.

OK, now start at the top right starting corner and drag all the way around the loop to find the next page of the loop.

- [Participant makes a complete loop around the screen]

Participant does not make a complete loop: That didn't work. To go to the next page of the loop, start at the starting corner and drag your finger clockwise all the way around and pass the starting corner again.

To go to earlier items, use the opposite gesture—a counterclockwise loop starting at the starting corner. Try that now.

- [Participant makes a counterclockwise loop]

Now is a good time to practice exploring and moving around the screen on your own. Let me know when you'd like to go on.

- [Participant tries navigating & scrolling on own and then indicates he or she is done.]

OK, so now you know how to move around the loop.

But, you also need to be able to DO things on the loop. The type of things that you can do depends on the thing that is highlighted in the loop. For many things, you can activate them by rapidly double-tapping anywhere on screen.

Go ahead and find the H (as in Hotel) button on the screen.

- [Participant finds H button]
- If participant does not find the right button: That's not the right button. Repeat the task.

Now, activate the H button by double-tapping.

- [Participant double-taps]
- If participant does not double-tap fast enough: That didn't work. You will need to be faster when you double-tap the screen. Repeat the "find the button task." Repeat the "activate" task.

Now, find and activate the T (as in Tango) button.

- [Participant should find the T button and then double-tap]
- Have the person find letter buttons (use the random letter sheet) until they find and activate 5 in a row correctly.

Different things on the loop may allow for different gestures. For example you might swipe up or down with a finger in the middle of the screen to change a number a little bit.

You will learn these other gestures later when we try out different loop interfaces.

F.1.4. After all initial interfaces are trained

- Ask ranking questions in E.3
- Go through the gesture practice application with participant

F.2. Preference elicitation (Phase 1)

We have trained on the basics of the item, loop, and list interfaces on the iPod. Now we are going to do some simple tasks and compare some different interfaces on the iPod.

During this part of the experiment, you will do three simple tasks many times.

1. On a very simple radio, you will change the radio band to F.M.
2. On an interface where you can build a simple ice cream sundae, you will change the topping to hot fudge.
3. And on another basic interface, you will set the volume to 15.

First, we will compare different interfaces on the list and item interfaces since they are very similar. Later, we will then compare different loop interfaces while doing the same simple tasks.

We will compare two interfaces at a time. You will try the task on one interface. Then you will try the same task on the second interface. After trying two interfaces, you will need to pick your favorite interface of the two on that task. If you don't have a favorite, you may also say that they are tied.

After you pick a favorite, you will try your favorite again and then compare it to a different interface with the same task until we've tried several different interfaces.

Before each interface I will describe it and instructions on how to use it. If you have any questions, you can ask them. During this part of the experiment, you can also ask me questions if you are having difficulty trying to complete a task. You may find that some of the interfaces do not work so well for you. This is okay. We want to know which interfaces work well for you and which ones are your favorite.

Do you have any questions so far? OK, let's get started. First we are going to try list and item interfaces. The first task that we will try is "Set the radio to the FM radio band."

- Go through the preference elicitation session with participant
- Questions from E.2 asked after each comparison/pair

F.3. Introduction to Phase 2 session/s

Thank you for coming back for the next session of the experiment. This session will be much like the first session. First, you will get some basic training on several different interfaces. Then we will go over some questions. Finally, we will have you try a number of tasks on several different devices and get your feedback and comments.

Blindfolded participants only Like before, you will be wearing a blindfold for portions of the experiment. We can take breaks from the blindfold, but check with me first so we can make sure it is a good time and that things are covered up. Please don't try to look around or under the blindfold when doing the tasks.

It is important to remember throughout the study that we are not testing you. We are testing the products and the interfaces to see which things work better for different people. If something is difficult or frustrating, let me know! We want that information so that we can improve devices in the future. Also, if you feel like you need a break, let me know and we can take a break at the next opportunity.

Let's find a comfortable volume.

- Set volume with participant

OK, now that we have found a good volume, I would like to find a good speech rate. To do that, I will have the system read a phrase, which you will listen to and then repeat back to me. I will do this a number of times. OK, are you ready?

- Use application to set speech rate

OK, now I'm going to have you double-tap the screen several times. Go ahead and double-tap now.

- Use application to set double-tap rate
- Do gesture practice session with participant

Note: The five interfaces (Copier Physical, Thermostat Physical, Item, List, and Loop) were presented in random order.

F.3.1. Training for copier physical interface

Now, I want to introduce you to the physical interface on the multifunction copier that you will try later. The copier is on your right. The copier has a touchscreen and some buttons to the right of the touchscreen.

Go ahead and feel the copier. You can ask questions about the copier while I double check the video camera. If you want, just point to any button and I will tell you about it.

If you want to make a single copy of a document, you would first put the document in the top tray of the copier. Don't worry about putting the copy in for this experiment however. Then you would press the Copy button. You can find the Copy button to the right of the touchscreen. The Copy button is the middle button in the bottom row of buttons.

- [Participant finds the green Copy button]
- If there is difficulty: Let's find the Copy button now. First, find the touchscreen in the middle of the copier. … The buttons are to the right. … Find the bottom row of buttons. … The Copy button is the middle button.

If you want to make multiple copies of a document, you would put the document in the top tray and then enter the number of copies you want with the number pad and then press the Copy button. The number pad is on the right side of the touchscreen.

- [See that the participant feels the numeric keypad.]

The number pad is arranged like the keypad on a telephone with the 1 key in the upper left corner. The 5 key has a nib or bump on it.

- [Participant feels the numeric keypad and finds the 5 key with nib.]

The more advanced features of the copier are only available through menus on the touchscreen of the copier.

F.3.2. Training for thermostat physical interface

Now, I want to introduce you to the physical interface on the smart thermostat that you will try later. The thermostat is plugged in but is not attached to the wall. I'm going to put it in front of you now. Keep it on the table while you are using it so that it is easier for the overhead camera to record it.

This smart thermostat has a touchscreen and some buttons on both the right and left sides of the screen. You can ask questions about the thermostat while I double-check the video camera. If you want, just point to any button and I will tell you about it.

If you wanted to change the temperature with this thermostat, you would press the Plus or Minus buttons. The first time you press the Plus or Minus button it will wake up the thermostat without a beep if the screen was dim. The second time you press the Plus or Minus button, it opens the temperature change screen with a beep but without changing the temperature. Then, each additional time you press it, it will beep and the target temperature will go up or down one degree. So you will hear one beep

before the temperature starts to change. The second and additional beeps will let you know that you are changing the target temperature by degree.

Touch the top right button.

- [See that the participant puts their finger on the Plus (+) button]
- If there is difficulty: Let's find the Plus button now. First, find the touchscreen in the middle of the thermostat. … There are buttons just to the right of the screen. … Find the top button, which is just off the top right corner of the touchscreen.

That is the Plus button, which turns up the temperature.

Now, touch the bottom right button.

- [See that the participant puts their finger on the Minus (-) button]
- If there is difficulty: Let's find the Minus button now. First, find the touchscreen in the middle of the thermostat. … There are buttons just to the right of the screen. … Find the bottom button, which is just off the bottom right corner of the touchscreen.

That is the Minus button, which turns down the temperature.

The more advanced features of the thermostat are only available through menus on the touchscreen of the thermostat.

F.3.3. Item interface training

You tried the Item interface during the first session, but we will try it again quickly to make sure that you remember how to use it.

With an item interface, everything on the screen is an item. These items may be arranged in different locations on the touchscreen—there is no standard pattern or layout. You can either EXPLORE the items of the interface by dragging your finger around, or you can STEP THROUGH the items of an interface by making swiping gestures. Swiping right or left to step through the interface guarantees that you will get to everything in the interface. However, dragging to explore may be quicker, especially if the items on the screen are large enough to find easily.

You use different gestures to interact with different items on the screen. If you are not sure what to do, you can just lift your finger and wait for a few seconds. Many times, the system will give you a hint and tell you the things you can do.

For training, we will have you do the three tasks from last time: setting the radio band to FM, choosing the hot fudge sundae topping, and changing the volume to 15.

If you are having difficulty with the tasks, let me know and I can give you a little bit of help. Are you ready to get started?

- Task 1. Set the radio to the FM radio band.
- Task 2. Change the sundae topping to hot fudge.
- Task 3. Change the volume to 15.

F.3.4. List interface training

You tried the List interface during the first session, but we will try it again quickly to make sure that you remember how to use it.

With a list interface, everything on the screen is in a list from the top of the screen to the bottom. You can either EXPLORE the things that make up the list of the interface by dragging your finger up and down the screen, or you can STEP THROUGH the list by making swiping gestures. Swiping right or left to step through the interface guarantees that you will get to everything in the interface. However, dragging up or down to explore may be quicker, especially if the things on the list are large enough to find easily.

You use different gestures to interact with different things on the screen. If you are not sure what to do, you can just lift your finger and wait for a few seconds. Many times, the system will give you a hint and tell you the things you can do.

For training, we will have you do the three tasks from last time: setting the radio band to FM, choosing the hot fudge sundae topping, and changing the volume to 15.

If you are having difficulty with the tasks, let me know and I can give you a little bit of help. Are you ready to get started?

- Task 1. Set the radio to the FM radio band.
- Task 2. Change the sundae topping to hot fudge.
- Task 3. Change the volume to 15.

F.3.5. Loop interface training

You tried the Loop interface during the first session, but we will try it again quickly to make sure that you remember how to use it.

With a loop interface, everything is arranged in a clockwise loop around the edges of the screen. Nothing is in the middle of the screen. This allows you to drag your finger in a loop to get to things. You can either EXPLORE the things of the loop interface by dragging your finger around the edges of the screen, or you can STEP THROUGH the things of an interface by making swiping gestures. Swiping right

or left to step through the interface guarantees that you will get to everything in the interface. However, dragging around the edges of the screen to explore may be quicker.

The starting corner for the loop is the upper right corner and continues clockwise. If you need to get to a different screen, you continue to make a loop all the way around the screen.

You use different gestures to interact with different things on the screen. If you are not sure what to do, you can just lift your finger and wait for a few seconds. Many times, the system will give you a hint and tell you the things you can do.

For training, we will have you do the three tasks from last time: setting the radio band to FM, choosing the hot fudge sundae topping, and changing the volume to 15.

If you are having difficulty with the tasks, let me know and I can give you a little bit of help. Are you ready to get started?

- Task 1. Set the radio to the FM radio band.
- Task 2. Change the sundae topping to hot fudge.
- Task 3. Change the volume to 15.

F.3.6. After all 5 interfaces are trained

- Have participant rank interfaces (E.3)
- Administer questionnaire (E.4)

F.4. Thermostat Interface Testing (Phase 2)

F.4.1. Introduction to thermostat (first time)

We are going to do several tasks with a smart thermostat. Before we do anything, let me first tell you all of the things I'm going to have you try. Then, before you do any of them, I will ask you how hard you think it will be to do the tasks if you had thermostat interface that was accessible to you.

On a scale from 1 to 7, how difficult or easy do you expect the following tasks will be? (With 1 being "very difficult" and 7 being "very easy")

- Task 1. Raise the target temperature 2 degrees. ___
- Task 2. Set the target temperature to 70 degrees. ___
- Task 3. Turn the fan on. ___
- Task 4. Change the mode to cool. ___
- Task 5. Set the target temperature to 75 degrees and turn hold on. ___

When doing the tasks, you do not need to worry about actually changing the temperature or other settings because it is not hooked up to an actual heating or air conditioning unit.

Note: The four interfaces (Physical, Item, List, and Loop) were presented in an order counterbalanced for each participant across devices and replication.

F.4.2. Reintroduction to thermostat (replication)

OK, now we are going to go back and try all four interfaces again with the thermostat tasks. I'm looking to see if there is any improvement after you've tried everything once before.

F.4.3. Physical thermostat

Now we are going use the actual thermostat, which I will put in front of you.

OK, now I'm going to ask you to do some tasks. Remember that some tasks may be difficult, and you may not be able to complete some tasks. For each task, let me know when you think you are done or if you don't think that you can complete the task. I can't offer you any help, so just try to perform the task as quickly and as accurately as possible. If you forget the task you are trying to do at any point, just ask me and I will repeat it.

After you hear the task, you can go ahead and start right away.

- Task 1. Raise the target temperature 2 degrees.
- Task 2. Set the target temperature to 70 degrees.
- Task 3. Turn the fan on.
- Task 4. Change the mode to cool.
- Task 5. Set the target temperature to 75 degrees and turn hold on.
    - Ask SEQ after each task (E.5)
- Administer interface-level questionnaire (E.6)

F.4.4. Thermostat Item interface

Now, we are going to use the item interface on the iPod Touch to control the thermostat. I'm going to ask you to do some tasks. Remember that some tasks may be difficult, and you may not be able to complete some tasks. There is no "Done with Task" button on this interface, so you will need to let me know when you think you are done or if you don't think that you can complete the task. I can't offer you any help, so just try to perform the task as quickly and as accurately as possible. If you forget the task you are trying to do at any point, just ask me and I will repeat it.

Remember, this is the Item interface. After you hear the task, you can go ahead and start right away.

- Task 1. Raise the target temperature 2 degrees.
- Task 2. Set the target temperature to 70 degrees.
- Task 3. Turn the fan on.
- Task 4. Change the mode to cool.
- Task 5. Set the target temperature to 75 degrees and turn hold on.
    - Ask SEQ after each task (E.5)
- Administer interface-level questionnaire (E.6)

F.4.5. Thermostat List interface

Now, we are going to use the list interface on the iPod Touch to control the thermostat. I'm going to ask you to do some tasks. Remember that some tasks may be difficult, and you may not be able to complete some tasks. For each task, when you are done, you can use the "Submit job" button. This will automatically bring you to the next task, which I will then read to you. If you don't think that you can complete a task, then let me know. I can't offer you any help, so just try to perform the task as quickly and as accurately as possible. If you forget the task you are trying to do at any point, just ask me and I will repeat it.

Remember, this is the List interface. When I'm done reading the task, double-tap the screen in order to start the task.

- Task 1. Raise the target temperature 2 degrees.
- Task 2. Set the target temperature to 70 degrees.
- Task 3. Turn the fan on.
- Task 4. Change the mode to cool.
- Task 5. Set the target temperature to 75 degrees and turn hold on.
    - Ask SEQ after each task (E.5)
- Administer interface-level questionnaire (E.6)

F.4.6. Thermostat Loop interface

Now, we are going to use the loop interface on the iPod Touch to control the thermostat. I'm going to ask you to do some tasks. Remember that some tasks may be difficult, and you may not be able to complete some tasks. For each task, when you are done, you should use the "Submit job" button. This will automatically bring you to the next task, which I will then read to you. If you don't think that you can complete a task, then let me know. I can't offer you any help, so just try to perform the task as quickly and as accurately as possible. If you forget the task you are trying to do at any point, just ask me and I will repeat it.

Remember, this is the Loop interface. When I'm done reading the task, double-tap the screen in order to start the task.

- Task 1. Raise the target temperature 2 degrees.
- Task 2. Set the target temperature to 70 degrees.
- Task 3. Turn the fan on.
- Task 4. Change the mode to cool.
- Task 5. Set the target temperature to 75 degrees and turn hold on.
    - Ask SEQ after each task (E.5)
- Administer interface-level questionnaire (E.6)

F.4.7. After all 4 thermostat interfaces are tested

- Administer device-level questionnaire (E.7)

F.5. Copier Interface Testing (Phase 2)

F.5.1. Introduction to copier (first time)

We are going to do several tasks on the multifunction copier and iPod interfaces. Before we do anything, let me first tell you all of the things I'm going to have you try. Then, before you do any of them, I will ask you how hard you think it will be to do the tasks if you had a copier interface that was accessible to you.

On a scale from 1 to 7, how difficult or easy do you expect the following tasks will be? (With 1 being "very difficult" and 7 being "very easy")?

- Task 1. Make a single copy. ___
- Task 2. Make a darker copy (greater number). ___
- Task 3. Make 5 copies. ___
- Task 4. Make a 2-sided copy from a 2-sided original. ___
- Task 5. Set the Content setting to "Photograph." ___

Note: The four interfaces (Physical, Item, List, and Loop) were presented in an order counterbalanced for each participant across devices and replication.

F.5.2. Reintroduction to copier (replication)

OK, now we are going to go back and try all four interfaces again with the copier tasks. I'm looking to see if there is any improvement after you've tried everything once before.

F.5.3. Physical copier

Now, we are going to use the physical interface on the actual multifunction copier. The copier is to your right. Go ahead and face the copier and I will double-check the video camera.

OK, now I'm going to ask you to do some tasks. Remember that some tasks may be difficult, and you may not be able to complete some tasks. For each task, let me know when you think you are done or if you don't think that you can complete the task. I can't offer you any help, so just try to perform the task as quickly and as accurately as possible. If you forget the task you are trying to do at any point, just ask me and I will repeat it.

After you hear the task, you can go ahead and start it right away.

- Task 1. Make a single copy.
- Task 2. Make a darker copy (greater number).
- Task 3. Make 5 copies.
- Task 4. Make a 2-sided copy from a 2-sided original.
- Task 5. Set the Content setting to "Photograph."
    - Ask SEQ after each task (E.5)
- Administer interface-level questionnaire (E.6)

F.5.4. Copier Item interface

Now, we are going to use the Item interface on the iPod Touch to control the copier. I'm going to ask you to do some tasks. Remember that some tasks may be difficult, and you may not be able to complete some tasks. For each task, when you are done, you can use the "Submit Job" button. Let me know if you don't think that you can complete the task. I can't offer you any help, so just try to perform the task as quickly and as accurately as possible. If you forget the task you are trying to do at any point, just ask me and I will repeat it.

Remember, this is the Item interface. After you hear the task, you can go ahead and start it right away.

- Task 1. Make a single copy.
- Task 2. Make a darker copy (greater number).
- Task 3. Make 5 copies.
- Task 4. Make a 2-sided copy from a 2-sided original.
- Task 5. Set the Content setting to "Photograph."
    - Ask SEQ after each task (E.5)
- Administer interface-level questionnaire (E.6)

F.5.5. Copier List interface

Now, we are going to use List interface on the iPod Touch to control the copier. I'm going to ask you to do some tasks. Remember that some tasks may be difficult, and you may not be able to complete some tasks. For each task, when you are done, you can use the "Submit job" button. This will automatically bring you to the next task, which I will then read to you. If you don't think that you can complete a task, then let me know. I can't offer you any help, so just try to perform the task as quickly and as accurately as possible. If you forget the task you are trying to do at any point, just ask me and I will repeat it.

Remember, this is the List interface. When I'm done reading the task, double-tap the screen in order to start the task.

- Task 1. Make a single copy.
- Task 2. Make a darker copy (greater number).
- Task 3. Make 5 copies.
- Task 4. Make a 2-sided copy from a 2-sided original.
- Task 5. Set the Content setting to "Photograph."
    - Ask SEQ after each task (E.5)
- Administer interface-level questionnaire (E.6)

F.5.6. Copier Loop interface

Now, we are going to the Loop interface on the iPod Touch with a loop interface to control the copier. I'm going to ask you to do some tasks. Remember that some tasks may be difficult, and you may not be able to complete some tasks. For each task, when you are done, you can use the "Submit Job" button. This will automatically bring you to the next task, which I will then read to you. If you don't think that you can complete a task, then let me know. I can't offer you any help, so just try to perform the task as quickly and as accurately as possible. If you forget the task you are trying to do at any point, just ask me and I will repeat it.

Remember, this is the Loop interface. When I'm done reading the task, double-tap the screen in order to start the task.

- Task 1. Make a single copy.
- Task 2. Make a darker copy (greater number).
- Task 3. Make 5 copies.
- Task 4. Make a 2-sided copy from a 2-sided original.
- Task 5. Set the Content setting to "Photograph."
    - Ask SEQ after each task (E.5)
- Administer interface-level questionnaire (E.6)

F.5.7. After all 4 copier interfaces are tested

- Administer device-level questionnaire (E.7)

F.6. At the end of the experiment

- Administer final questionnaire (E.8)

Appendix G. Repeated Measure ANOVA Tables

G.1. Ranked performance data for the copier

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial η^2
Within-Subject Effects						
Interface	72452.853	1	72452.853	3.346	0.082	0.137
Interface * Group	9187.760	1	9187.760	0.424	0.522	0.020
Error(Interface)	454787.321	21	21656.539			
Task	59281.023	4	14820.256	1.317	0.270	0.059
Task × Group	76098.516	4	19024.629	1.691	0.160	0.075
Error(Task)	945058.993	84	11250.702			
Replication	396529.371	1	396529.371	43.039	< 0.001	0.672
Replication × Group	274.621	1	274.621	0.030	0.865	0.001
Error(Replication)	193477.371	21	9213.208			
Interface × Task	68383.753	4	17095.938	3.385	0.013	0.139
Interface × Task × Group	14143.063	4	3535.766	0.700	0.594	0.032
Error(Interface × Task)	424213.421	84	5050.160			
Interface × Replication	6336.110	1	6336.110	0.853	0.366	0.039
Interface × Replication × Group	3084.534	1	3084.534	0.415	0.526	0.019
Error(Interface × Replication)	155994.014	21	7428.286			
Task × Replication	43546.006	4	10886.502	2.370	0.059	0.101
Task × Replication × Group	48763.321	4	12190.830	2.654	0.039	0.112
Error(Task × Replication)	385889.307	84	4593.920			
Interface × Task × Replication	10345.033	4	2586.258	0.611	0.656	0.028
Interface × Task × Replication × Group	11547.174	4	2886.794	0.682	0.607	0.031
Error(Interface × Task × Replication)	355735.452	84	4234.946			
Between-Subjects Effects						
Intercept	13091441.690	1	13091441.690	136.876	<0.001	0.867
Group	411135.856	1	411135.856	4.299	0.051	0.170
Error	2008528.634	21	95644221			

G.2. Ranked performance data for the thermostat

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial η^2
Within-Subject Effects						
Interface	19507.500	1	19507.500	1.149	0.295	0.050
Interface × Blind	8027.852	1	8027.852	0.473	0.499	0.021
Error(Interface)	373593.523	22	16981.524			
Task	1336670.818	4	334167.704	26.080	< 0.001	0.542
Task × Blind	42197.732	4	10549.433	0.823	0.514	0.036
Error(Task)	1127539.550	88	12812.949			
Replication	289395.408	1	289395.408	25.540	< 0.001	0.537
Replication × Blind	2905.752	1	2905.752	0.256	0.618	0.012
Error(Replication)	249283.315	22	11331.060			
Interface × Task × Blind	14930.111	4	3732.528	0.468	0.759	0.021
Error(Interface × Task)	701360.758	88	7970.009			
Interface × Replication	2227.408	1	2227.408	0.273	0.607	0.012
Interface × Replication × Blind	10444.002	1	10444.002	1.279	0.270	0.055
Error(Interface × Replication)	179666.665	22	8166.667			
Task × Replication	161671.889	4	40417.972	4.830	0.001	0.180
Task × Replication × Blind	5445.420	4	1361.355	0.163	0.957	0.007
Error(Task × Replication)	736412.592	88	8368.325			
Interface × Task × Replication	25224.076	4	6306.019	0.808	0.523	0.035
Interface × Task × Replication × Blind	9706.357	4	2426.589	0.311	0.870	0.014
Error(Interface × Task × Replication)	686982.867	88	7806.623			
Between Subjects Effects						
Intercept	13881660.000	1	13881660.000	211.786	< 0.001	0.906
Group	145558.376	1	145558.376	2.221	0.150	0.092
Error	1442005.261	22	65545.694			

G.3. SUS data for three Interfaces (Item, List, and Loop)

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial η^2
Within-Subject Effects						
Device	123.158	1	123.158	0.454	0.508	.020
Device × Group	885.503	1	885.503	3.263	0.085	.129
Error(Device)	5971.200	22	271.418			
Interface	193054.548	2	96527.274	77.800	< 0.001	.780
Interface × Group	2642.511	2	1321.255	1.065	0.353	.046
Error(Interface)	54591.599	44	1240.718			
Replication	969.223	1	969.223	3.819	0.064	.148
Replication × Group	2231.491	1	2231.491	8.792	0.007	.286
Error(Replication)	5583.777	22	253.808			
Device × Interface	1152.233	2	576.116	2.374	0.105	.097
Device × Interface × Group	642.202	2	321.101	1.323	0.277	.057
Error(Device × Interface)	10677.093	44	242.661			
Device × Replication	181.028	1	181.028	0.950	0.340	.041
Device × Replication × Group	323.003	1	323.003	1.695	0.206	.072
Error(Device × Replication)	4191.570	22	190.526			
Interface × Replication [Greenhouse-Geisser corrected]	2460.335	1.525	1613.514	5.148	0.018	.190
Interface × Replication × Group [Greenhouse-Geisser corrected]	320.057	1.525	209.897	0.670	0.480	.030
Error(Interface × Replication) [Greenhouse-Geisser corrected]	10514.284	33.546	313.426			
Device × Interface × Replication	1011.724	2	505.862	2.450	0.098	.100
Device × Interface × Replication × Group	701.230	2	350.615	1.698	0.195	.072
Error(Device × Interface × Replication)	9085.889	44	206.497			
Between-Subjects Effects						
Intercept	440028.938	1	440028.938	772.135	< 0.001	0.972
Group	2679.941	1	2679.941	4.703	0.041	0.176
Error	12537.494	22	569.886			

S. D. G.