Flexible Physical Interfaces

by

Nicolas Villar

B.Sc. Hons. Computer Science with Multimedia (Lancaster University), 2002

Submitted in part fulfilment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

COMPUTING DEPARTMENT

of

LANCASTER UNIVERSITY

November 2007

ABSTRACT

Flexible Physical Interfaces

Nicolas Villar

Human-computer interface devices are rigid, and afford little or no opportunity for end-user adaptation. This thesis proposes that valuable new interaction possibilities can be generated through the development of user interface hardware that is increasingly *flexible*, and allows end-users to physically shape, construct and modify physical interfaces for interactive systems.

The work is centred around the development of a novel platform for flexible user interfaces (called *VoodooIO*) that allows end-users to compose and adapt physical control structures in a manner that is both versatile and simple to use. VoodooIO has two main physical elements: a pliable material (called the *substrate*), and a set of physical user interface *controls*, which can be arranged on the surface of the substrate. The substrate can be shaped, applied to existing surfaces, attached to objects and placed on walls and furniture to designate interface areas on which users can spatially lay out controls.

From a technical perspective, the design of VoodooIO is based on a novel architecture for user interfaces as networks of controls, where each control is implemented as a network node with physical input and output capabilities. The architecture overcomes the inflexibility that is usually imposed by hard-wired circuitry in traditional interface devices, by enabling individual control elements that can be connected and disconnected ad hoc from a shared network bus. The architecture includes support for a wide and extensible range of control types; fast control identification and presence detection, and an application-level interface that abstracts from low level implementation details and network management processes.

The concrete contributions to the field of human-computer interaction include a motivation for the development of flexible physical interfaces, a fully working example of such a technology, and insights gathered from its application and study.

PREFACE

This dissertation has not been submitted in support of an application for another degree at this or any other university. It is the result of my own work and include nothing which is the outcome of work done in collaboration except where specifically indicated.

Excerpts of this thesis have been published in journal, conference and workshop manuscripts, most notably in:

Nicolas Villar and Hans Gellersen. "A Malleable Control Structure for Soft-Wired User Interfaces" In Proc. *Tangible and Embedded Interaction (TEI '07)*. Baton Rouge, USA. January 2007.

Wolfgang Spiessl, Nicolas Villar, Hans Gellersen and Albrecht Schmidt. VoodooFlash: Authoring across Digital and Physical Form" In Proc. *Tangible and Embedded Interaction (TEI '07)*. Baton Rouge, USA. January 2007.

Nicolas Villar, Florian Block. "Distributed and Adaptable Home Control." In Adj. Proc. *ACM Symposium on User Interface Software and Technology (UIST '06)*. Montreux, Switzerland. October 2006.

Nicolas Villar and Hans Gellersen. "The Friday Afternoon Project: A Two-Hour Prototyping Exercise" In Proc. *Mobile and Embedded Interactive Systems (MEIS'06)*. Dresden, Germany. October 2006.

Florian Block, Nicolas Villar, Mike Hazas, Dave Molyneaux and Hans Gellersen. "Locating Physical Objects on Interactive Surfaces" In Proc. Mobile and Embedded Interactive Systems (MEIS'06). Dresden, Germany. October 2006.

Nicolas Villar, Kiel Gilleade, Devina Raymundy-Ellis and Hans Gellersen. "The VoodooIO Gaming Kit: A Real-Time Adaptable Gaming Controller." In Proc. of *ACM Advances in Computer Entertainment (ACE '06)*. Los Angeles, USA. June 2006.

John Bowers and Nicolas Villar. "Creating Ad Hoc Instruments with PPP". In Proc. *New Interfaces for Musical Expression (NIME '06)*, Paris, France. June 2006.

Yasue Kishino, Tsutomo Terada, Nicolas Villar and Shojiro Nishio. A Position Detection Mechanism using Camera Images for Pin-Shaped Input/Output Devices. In Adj. Proc. *International Conference on Ubiquitous Computing (UbiComp)*. Tokyo, Japan. September 2005.

Nicolas Villar, Adam Lindsay and Hans Gellersen. "A rearrangeable tangible interface for musical composition and performance." In Proc. *New Interfaces for Musical Expression (NIME '05)*, Vancouver, Canada. May 2005.

Nicolas Villar, Kristof Van Laerhoven, Hans Gellersen. "A Physical Notice Board with Digital Logic and Display". In Adj. Proc. *European Symposium on Ambient Intelligence (EUSAI '04)*. Eindhoven, Netherlands. November 2004.

Kristof Van Laerhoven, Nicolas Villar and Hans Gellersen. "Pin&Mix: When Pins Become Interaction Components." In Proc. *Workshop on Real World User Interfaces (PI03) at the Fifth International Symposium on Human Computer Interaction with Mobile Devices and Services (MobileHCI 2003)*. Udine, Italy. September 2003.

Kristof Van Laerhoven, Nicolas Villar, Albrecht Schmidt, Hans Gellersen, Maria Hakansson, and Lars Erik Holmquist. "Pin&Play: The Surface as Network Medium" In *IEEE Communications Magazine*, Vol.41 No.4, IEEE Press: pp. 90-96. April 2003.

"...So now you see what I meant about Lego blocks. They have more or less the same properties as those which Democritus ascribed to atoms. And that is what makes them so much fun to build with. They are first and foremost indivisible. Then they have different shapes and sizes. They are solid and impermeable. They also have 'hooks' and 'barbs' so that they can be connected to form every conceivable figure. These connections can later be broken again so that new figures can be constructed from the same blocks.

The fact that they can be used over and over is what makes Lego the most ingenious toy in the world. Each single Lego block can be part of a truck one day and part of a castle the day after. We could also say that Lego blocks are 'eternal.' Children of today can play with the same blocks their parents played with when they were little..."

- Jostein Gaarder, Sophie's World

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1
Introduction

The design of user interfaces has been a subject of active investigation over the last three decades, ever since personal computers popularised the experience of direct human-computer interaction. In this time, user interface design has advanced considerably, and modern interfaces are significantly better than the original text-based terminals of early PCs. In their seminal book "The Psychology of Human-Computer Interaction," published in 1983, the authors Card, Moran and Newell wrote:

> "[The] radical increase in both the computer's power and its performance/cost ratio has meant that an increasing amount of computational resources have become available to be spent on the human-computer interface itself, rather than on purely computational tasks. This increase of deployable resources exacerbates the novelty of the area, since entirely new styles of interaction become available coincidentally. These new styles often lead to completely new interfaces, which are then even more ragged than before. At the same time, opportunities for the invention of good interfaces also increase rapidly, accounting for the leaps and bounds we have seen in terms of major improvements in functionality and ease of use." [17]

The work presented in this thesis explores the interaction possibilities created by the development of a novel user interface paradigm, which is enabled by recent advances in technology and inspired by pioneering human-computer interaction (HCI) research from the past thirty years. The focus of the research are user interfaces that are characteristically *flexible*, being able to accommodate a high degree of user configuration and physical adaptation.

From a technical standpoint, the work is levered by the advent of low-cost embedded computing devices, improvements in networking techniques, and the availability of novel materials. The theoretical motivation for the research is based on previous work on novel interaction methods, alternative interface designs and established HCI principles.

The research efforts documented in this thesis follow a largely pragmatic approach: a novel interface technology is developed into a robust research platform, which is then characterised from a functional and usability perspective. New interaction possibilities are illustrated through a number of prototypical application scenarios, and the value of the interaction concepts is evaluated through formal user studies and other experiential methodology. The concrete contributions to the field of human-computer interaction include a motivation for the design of flexible user interfaces, a fully working example of such a technology, and insights gathered from its application and study.

1.1 Flexible Physical Interfaces

The degree of flexibility of an the interface is determined by how effectively it can support explicit adaptation to particular user interaction requirements. Display screens and graphical user interfaces (GUIs) are inherently flexible interface elements, being, in essence, flat surfaces that can render any number of virtual controls and visualisations. The flexibility of GUIs makes them the focus of attention of most modern user interfaces, where they play an essential role in accommodating a continually evolving variety of computer applications, each with its own interface requirements. The software that controls the behaviour of the interface – the application code and underlying operating system – is also, in principle, infinitely flexible in its functionality. Some of this flexibility is often exposed through user-configurable options and parameters, intended to accommodate individual user preference. In fact, the only elements of the interface that are not inherently flexible are the hardware devices necessary for physical user control.

The area of physical user interface flexibility has received relatively little attention from the research community. Perhaps the fundamental explanation for this lies in the lack of readily available technologies that afford users the ability to freely define, shape and configure the control structures of a human-computer interface. Current interface devices are inherently rigid in their design: the product of technical, mechanical material choices are devices that do not readily lend themselves to be modified after their manufacture. Take as an example the most ubiquitous interface device of today: the computer keyboard. Once a keyboard has been assembled at the factory – its circuit board laid out, electronic components soldered, keys fixed in place and encased by a plastic housing – it is no longer possible to alter its physical configuration without the use of specialist tools and expert knowledge. Even when practicable, any modifications to the keyboard (say, a forcible augmentation with an additional bank of keys) would not necessarily be supported by the rest of the interface. The interface software would likely need to be reprogrammed in order to accommodate any new functionality that the physical alterations are intended to supply.

The motivation behind this work is to make the physical elements of the user interface as flexible as software applications and graphical user interfaces. Following the example of the keyboard, it is envisioned that with a truly flexible interface the user should be able to effortlessly change the arrangement of the keys, add new keys if necessary, and remove extraneous ones if desired. The user should be able to exchange keys for some other type of controls if appropriate – input sliders, for example – and even change the shape and size of the keyboard itself. The result should be a readily usable, fully functioning interface that remains continually modifiable.

In order to extend flexibility across the physical interface to this degree, it is first necessary to rethink current interface design using a holistic approach to the problem: at the low level, the design of electronic interface devices needs to be reconsidered, including their materials and form-factors; at the high level, it is important to take into account how the technology will integrate into the software elements of the interface, and consider what mechanisms need to be in place for the user to be able to make functional use of the added flexibility. Only then is it possible to explore application possibilities, refine design decisions, and evaluate the benefits, limitations and possible pitfalls of giving people the ability to modify the physical control structures that allow them to interact with their computers.

1.2 VoodooIO

The core contribution of this thesis is the development of a new technology for flexible user interfaces, which is called VoodooIO (short for Voodoo-Input/Output, and sometimes abbreviated as VIO). The technology supports the ad hoc composition and modification of physical user interface elements, in a manner that is both versatile and simple to use. In addition, the VoodooIO implementation includes a set of software drivers and tools that facilitate its integration with a wide range of applications.

VoodooIO has two main physical elements: a pliable, sheet-like material – called the substrate – that acts as the underlying physical layer on which the interface is constructed; and a set of physical user interface controls, which can be arranged on the surface of the substrate. Sections of substrate material can be shaped, placed on existing surfaces, attached to objects, walls, furniture or other elements of the architecture to create physical interface areas. Deployed substrate areas serve as the physical equivalent of graphical displays, framing the interaction by defining areas on which users can spatially lay out interface controls (Figure 1.1).

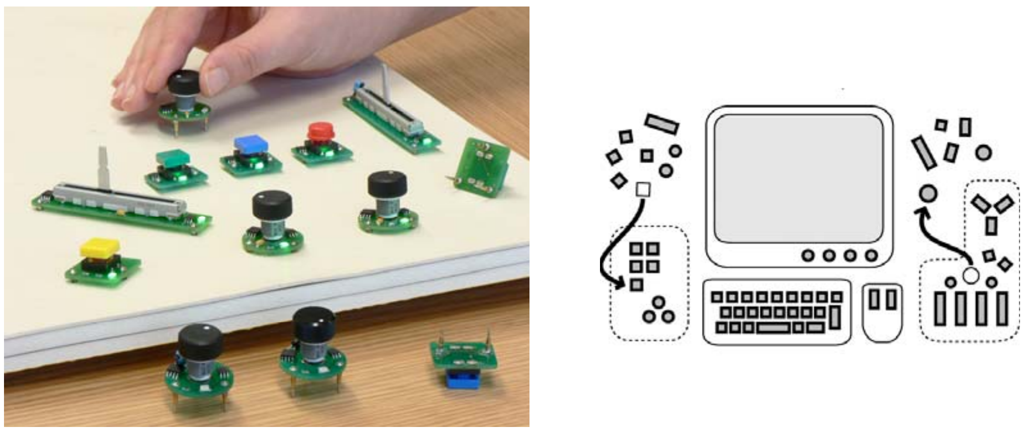


Figure 1.1: VoodooIO is a human-computer interface that users can shape, construct and modify.

Users of VoodooIO are supplied with a control set that includes a wide variety of basic control devices, both input (e.g buttons, switches, sliders and knobs) and output (e.g. lights and indicators). Each control is equipped with small pin-like connectors at its base, which can pierce into the soft substrate material to securely fasten it in place; controls can be plucked back out of the substrate in order to be removed, reoriented or rearranged. The interface can be physically modified at any time, supporting smooth transitions between configuration and use.

The technical design of VoodooIO is based on a novel concept of implementing user interface devices as networks of controls. Each physical control is a network node – a minimal, embedded computer – that can be connected and disconnected from a network bus. Controls are equipped with a "Pin&Play" connection mechanism, whereby the act of attaching a control to the substrate connects it to a network bus, which is built into the material as internal conductive layers that extend across its area. The connector mechanism uses pins act as coaxial connectors that require no predefined sockets on the substrate; rather, the pins create their own sockets on attachment. The substrate acts as a two-dimensional network cable, able to transmit network signals across its surface between controls and a network master, which drives the network and monitors the state of connected controls. When a control is connected, its presence is quickly detected on the network and identified by a unique ID. VoodooIO controls can be used by applications as personal interface devices on a single computer, as a privately shared resource on a local network, or as a public user interface service over the Internet.

1.3 Thesis Contributions and Overview of Work

The need to support some form of interface personalisation, tailoring or adaptation is well understood, and is, to some degree, common practice in user interface design. What is less well understood is how interface flexibility can be extended to include the physical parts of the human-computer interface so that they too can be changeable, reconfigurable and enabled with some of the versatility usually afforded only by software programs and graphical user interfaces.

This thesis presents the following contributions to the area of human-computer interaction research:

1. A novel design for flexible physical interfaces, which includes:
   - A. A system architecture for physical interface devices as reconfigurable networks of controls.
   - B. Innovative materials and mechanisms to support physical user interface modification.
   - C. Programming libraries and end-user configuration tools that act as software "glue" between the physical interface and software applications.
2. A characterisation of the technology, in terms of:

   a. Its technical capabilities and limitations.

   b. Its user-related properties and novel interaction techniques it enables.

3. Insights into the applicability of the technology and associated interaction techniques, collected through:

   a. Self-generated application examples, used to illustrate the envisioned style of interaction.

   b. User studies, focused on evaluating the novel interaction techniques afforded by the technology.

   c. Expert feedback in a number of application areas, commenting on the perceived benefits of greater interface flexibility applied to current practice.

The work is presented in a bottom-up approach: Chapter 2 lays the theoretical foundations for the work, and discusses previous research in related areas. Chapter 3 presents an architecture developed to support runtime physical interface reconfiguration, which implements interface equipment as networks of controls. The implications of the design on the performance of the physical interface (and how this relates to the user experience) are then characterised in Chapter 4.

Having described a generic architecture for networked controls, Chapter 5 describes how the concept of networked controls is applied to a concrete physical design, called VoodooIO. The VoodooIO design, including some software application tools discussed in Chapter 6, is then applied as a research platform to investigate the end-user benefit of physical interface flexibility. Chapter 7 collects a series of expert evaluations, formal user studies and application experiences that illustrate and inform the interaction possibilities of flexible interfaces in a variety of application areas, including product prototyping, game-playing and music production and performance.

Chapter 2

Background

In 1983, the first ACM Computer Human Interaction (CHI) conference marked an important milestone for a nascent research field; in 2007 a second milestone has been reached as the conference series celebrated its twenty-fifth year anniversary, publicised under the banner of "Look how far we have come. Imagine how far we can go." Admittedly, the human-computer interface is more usable today than it was twenty-five years ago, but it remains that much of the valuable knowledge and experience accumulated over this intense period of research is still underrepresented in mainstream user interface designs. In the practice of enabling new interaction possibilities through technological application, it remains important to apply the lessons learnt over the past quarter century, in parallel with observing current practice and looking to the future for inspiration. This chapter looks at work in the field of human-computer interaction and related areas that has shaped, inspired and motivated our vision for the development of a novel user interface paradigm, and its realisation in the VoodooIO technology.

2.1 A Language of Controls

At the first ever CHI conference in 1983, Nakatani and Rochlich presented a paper titled "Soft Machines: A Philospy of User-Interface Design" [63]. At the time, the authors were expressing a concern with the overwhelming complexity that general-purpose computer interfaces can present to a user, compared with the simplicity of more traditional "machines" that are "...special-purpose, have forms suggestive of their functions, are operated with controls in obvious one-to-one correspondence with their actions, and the consequences of the actions on visible objects are immediately and readily apparent." Nakatani and Rohlich envisioned a solution that allowed computers to be operated in a more machine-like way, which they demonstrated by using a touch-sensitive display to render graphical, machine-like user controls (such as knobs, switches, keys and pushbuttons) that could be used to directly manipulate application functions. The "soft-machine" design capitalised on what Chapanis called the "language of controls" (Chapanis in [63]): a semantic significance attached to machine control artefacts, learnt and evolved from several generations of machine interface design and use. It is this language that lets us simply recognise and inherently "know" that buttons are for pressing and dials are for turning; that rotating a knob in the direction of the arrow is going to have a gradually increasing effect on something, and that turning the opposite way will likely reverse the action. The presence (or absence) of controls can give an indication of the underlying capabilities of a system. The use of labelling, clustering and arrangement of controls can further reinforce what the purpose of the interface is, and how we are expected to use it.

The concept of a language of controls is closely related to Gibson's idea of affordances [33], in particular Norman's application of the term to the field of HCI [66] and Gaver's notion of "perceptible affordances" in technological objects [32]. It is also present in Scheiderman's motivation for the use of real-world metaphors in direct-manipulation user interface designs [77]. However, neither the concept of affordance or the use of metaphors encapsulate the way a designer can communicate an idea to the user through their interface design choices: affordances, in the strict sense of the word, cannot be "added" to an interface [67]. Perhaps the most thorough understanding of the language of controls can be found in the field of human-computer semiotic engineering, which studies exactly how semantic meaning can be intentionally encoded into an interface design as a way for designers to communicate their vision and intentions to the user at the time of interaction [81].

2.2 The Right Tools for the Job

Beyond instilling a sense of familiarity and ease of use through their design, there exists further motivation for the use of multiple, specialised controls that can be directly grasped and manipulated by users in a multi-handed fashion (like machine controls), rather than virtual control representations that are operated serially and sequentially via a generic proxy device, such as the computer keyboard or mouse.

2.2.1 Space-Multiplexing

The concepts of *space-* and *time-multiplexed control* were coined by Fitzmaurice, Ishii, and Buxton through their work in "Graspable User Interfaces" [29]. Time-multiplexed control is characteristic of the way a user operates a graphical interface with a mouse: using that single physical control device, the user sequentially selects and manipulates virtual elements on the screen. The actual function of the mouse is constantly redefined over time, its function determined by its graphical context at a particular point in time. In contrast, a computer keyboard operates in a space-multiplexed manner, with each keyboard key being permanently associated with a single function. Each key provides dedicated access to a single interface function, readily accessible on the physical space that the keyboard occupies.

Through a series of empirical studies, Fitzmaurice et al. demonstrated some of the advantages of time-multiplexed over space-multiplexed control [28]. In their experiments, users usually performed better when operating interfaces which used dedicated physical control devices to manipulate graphical objects on the screen, compared to conditions where time-multiplexed control schemes were used. The results provided quantifiable evidence for the role of space-multiplexing in interface design, the benefits of which had been vocalised in even earlier work by Bill Buxton, who proposed that "distinct controls for specific functions provide the potential to improve the directness of the user's access, such as through decreased homing time and exploiting motor memory." [11].

2.2.2 Specialised Input Devices

The experiments reported in [28] also provided an insight into the advantage of using *specialised* devices in the interface: objects that are specifically devised to carry out a particular kind of interaction task. Buxton identified this same concern in [14], when he expressed his belief that "the quality of human input can be greatly improved through the use of *appropriate gestures...* [that] pay more attention to the "body language" of human computer dialogues." As a simple illustration of this, Buxton shows how the use of one of two very similar devices – a joystick and a trackball – can have a drastic effect on the user's ability to effectively operate the interface, even though they both afford continuos two-dimensional system input. The benefits of using the right tools for the job are further highlighted in Card et al's morphological analysis of input devices [16]. Beyond providing a comprehensive taxonomy of interface devices, the work takes into account human performance metrics, and illustrates how the subtle operational properties of a particular device can make it more or less suitable for a specific interaction task.

Fitzmaurice et al. conclude that we might want to design interfaces that "...[allow] for space-multiplexed, rapidly reconfigurable, specialised, input devices" in order to accommodate the individual needs of users engaged in a variety of different tasks [28]. The mention of support for "rapid reconfigurability" as a desirable property in a user interface is relevant to this work, and the thought will be paid particular attention later in the discourse. First, however, it is worth focusing on another factor which has only implicitly be mentioned so far: the use of physical objects and space in the interface.

2.3 The Role of Physicality

There are solid grounds from which to argue the benefits of space-multiplexed and specialised interface tools. However, the application of these principles is more an exception than the norm in current user interface design practice. The more established paradigm is the use of graphical user interfaces (GUIs), which follow the principle of Direct Manipulation [77] where virtual interface elements (such windows, widgets, icons, images and text) are operated *like* physical objects, but indirectly and sequentially, through a generic proxy device such as mouse or keyboard. There are clear advantages to this approach: GUIs are portable, compact and are infinitely reconfigurable. Even so, there exists a body of research which specifically argues for making the interface elements more real and less virtual, as solid objects which can be directly grasped, touched, and moved around real-world space.

2.3.1 Tangible Objects

In 2004, Fishkin contributed a taxonomy to a growing area of investigation – collectively known as "tangible" interfaces – that he generalised as: "interfaces where the user uses their hands to manipulate some physical object(s) via physical gestures; a computer system detects this, alters its state, and gives feedback accordingly" [26]. The term was originally coined by Ishii and Ullmer in [47], where it was used to describe interfaces where the physical "atoms" (objects) are tightly coupled with digital "bits" of information. Some of the early motivation for investigating these types of interfaces was originally articulated in [29], where it was suggested that making the interface out of "graspable" interface elements would be beneficial for the user, given that it:

- "[encourages] two handed interactions;
- shifts to more specialised, context sensitive input devices;
- allows for more parallel input specification by the user, thereby improving the expressiveness of the communication capacity with the computer;
- leverages off our well developed, everyday skills of prehensile behaviours for physical object manipulations;
- externalises traditionally internal computer representations'
- facilitates interactions by making interface elements more "direct" and more "manipulatable" by using physical artefacts;
- takes advantage of our keen spatial reasoning skills;
- offers a space multiplex design with a one to one mapping between control and controller; and finally,
- affords multi-person, collaborative use."

A similar line of reasoning has been followed by a number of different research approaches, which elaborate on the same basic concern. All have identified some desirable quality in interface designs which are characteristically "physical" [37], "token-based" [45], "manipulative" [42] or "embodied" [27].

Many compelling examples of physical interfaces have been investigated in recent HCI research, demonstrating a variety of advantages that physical interface objects can have over graphical ones controlled with mouse and keyboard. For example, physical affordances can be utilised for ease of use [18], physicality of interface objects can make for more engaging user experiences ([36], [69]), and embedding in the physical environment can facilitate interaction and collaboration away from the desktop [40], [64].

2.3.2 Use of Space

When interface elements exist as tangible physical objects – rather than virtual elements on a screen – it is interesting to consider how their location in real-world space can play a role in the interaction. David Kirsh, in an essay titled "The Intelligent Use of Space," [51] elaborates on the premise that:

> "...in having a body, we are spatially located creatures: we must always be facing some direction, have only certain objects in view, be within reach of certain others. How we manage the spatial arrangement of items around us is not an afterthought; it is an integral part of the way we think, plan and behave."

Kirsh brings attention to the way we reconfigure the environment – and in particular, the way we organise objects within it – as a way to provide semantic cues for ourselves: such as laying out tools on a workbench in the sequence they are intended to be used; or clustering paper documents together to indicate a relationship between them.

The ability to spatially arrange tangible interface objects within the environment has been the focus of research by Patten and Ishii, which is reported in [68]. The results of the study illustrated some promising interaction advantages over a graphical user interface, resultant from being able to use of the environment as a way to spatially organise interface concepts (embodied by physical tokens, rather than graphical icons). It is interesting to note the particular observation that "...it can be useful for an interface to provide ways for the user to move and organise objects without these operations being interpreted by the [interface]." This is in contrast to many designs for tangible user interfaces that include the ability to track object movement and position, and can attach (explicit or implicit) causality to the act of modifying their relative arrangement to other objects; their location on a surface, or their position within in the environment [83].

2.4 Room for Change

A present concern in user interaction research is an awareness that it is simply not possible to design generic systems and interfaces that are appropriate for all individual users, taking into account different settings and possible contexts of use [59], [15], [50]. Earlier, the discourse touched on the benefits of designing user interfaces that are specific to a particular task, rather than generic to all of them. Given the many generations of machine-control design practice, the design of such interfaces should be well understood. However, a problem arises when the underlying functionality of the system is subject to change, as is the case with any general-purpose computing platform – how can the interface remain specific, if the underlying purpose of the machine can change?

The same problem can be formulated from a different perspective. As discussed previously, people adapt and modify their environment to suit their needs and optimise their interaction with the world. In addition, individuals may have particular needs and opinions that may dictate a unique interface design that is specific to their individual requirements, raising the question: how can an interface design remain appropriate for different users, given individual preference and requirements?

A number of different approaches to accommodate individual user preference and support task conformance have been reported in the research literature. The scope of the research is broad, but shares the common motivation of exploring how (and to what extent) it is possible to include some room for change in the design of a user interface in a way that affords users the flexibility to tailor, adapt, reconfigure and personalise their interaction environments.

2.4.1 Adaptive and Adaptable Interfaces

It is important to make a distinction between user interfaces that are *adaptive*, and those that are described as being *adaptable*. Both terms are sometimes used interchangeably in the research literature to refer to interfaces that can somehow be personalised in their configuration, and can be tailored to changing circumstances [21]. In a stricter definition, adaptive interfaces can be said to be *self*-adapting, i.e. the system plays an active role in altering the interface configuration. In contrast, adaptable interfaces provide some form of customisation mechanism, but expect the user to explicitly carry out the adaptation.

An initial occurrence of the notion of adaptable user interfaces can be found in [49], which discusses support for alternative interaction dialogues to achieve the same task. Early work in the area has been motivated by the need to support different user interaction practices [59]. Motivated by the problem of applications that are increasingly "bloated" with features, and that any one user only ever uses a small subset of these features, Findlater and McGrenere report on a comparative study of static, adaptive and adaptable user interfaces [25]. Their results indicate that users prefer to explicitly adapt their interface, rather than using a static or self-adaptive design. The authors conclude that users can carry out customisation tasks effectively, and that the majority of users want a personalised interface. However, the point is made that users are less likely to customise their design if the mechanisms for adaptation are difficult to learn and use, and that users need to be guided in the process by way of example. The same feeling is echoed in work by MacLean et al. [59], which purports the need to create a "tailoring culture" where users are comfortable performing interface customisations, and have access to the support and tools necessary to do so.

2.4.2 End-User Development

Related to the notion of user-adaptable software is the idea of end-user development (EUD), which aims to allow non-technical computer users (who that have little or no knowledge of computer programming) to create their own software applications. Research in EUD began with the development of the Pygmalion visual programming environment, which was released in 1975 [79]. Pygmalion introduced the concept of programming-by-example, by editing graphical artefacts rather than writing code. Later research has focused on developing simplified programming notations such as HyperTalk (the programming language behind the Apple Computer's HyperCard hypermedia application program); graphical development languages such as Visual

Agent Talk [73], and other environments that encourage programming by demonstration [19].

More recent developments have been enabled by the widespread use of the internet. The Koala system allows users to record actions performed on a web page, and then replay them on demand [58]. Marmite is an end-user programming tool that allows users to create web "mashups" by combining disparate internet data sources and services [96]. Other projects have focused on allowing users to manage interactive systems and appliances in a home scenario. Rodden et al. report on a system that allows users to build simple programs by dynamically assembling jigsaw pieces that represent ubiquitous computing components [75]. Targeting the same design space are the Media Cubes programming language [6], the iCAP context-aware prototyping system [80] and the a CAPella environment for developing context-aware applications by demonstration [20].

2.4.3 Physical Flexiblity

Finding some level of end-user adaptability is increasingly common in graphical user interface designs; it can range from support for cosmetic tailoring (such as the ability to apply different interface "skins" or "themes") to functional extensibility (e.g. tools for recording or programming custom macros). Enabling physical interface flexibility is a growing area of research, and, whether coincidentally or explicitly, it is a concern touched upon by a number of different research efforts.

There are a number of toolkit and infrastructure contributions intended to simplify the development and deployment of physical interfaces. The Phidgets platform introduced the concept of "physical widgets": physical sensors and actuators that can be easily connected to a computer and programmed like graphical widgets [37]. The Phidgets shield interface designers from low-level device management and interfacing details, and expose their functionality through a high-level application programming interface (API). A particular use of the Phidgets platform is presented in [38], where the authors introduce a notion of customisable physical interfaces, and provide a system of "widget taps" that let user bind physical controls to graphical controls in conventional applications.

The Arduino [1] and Gainer [31] kits include general-purpose I/O boards that abstract from the low-level complexity involved in working with electronics and microcontrollers, and are programmable from graphical and object-oriented development environments. They are widely used by researchers, designers and artists who wish to embed interactivity into their design, but have limited programming or embedded systems development experience. The Calder [56], Papier-Mâché [53] and d.tools [43] projects focus on supporting interface designers in the process of developing rapid prototypes of their ideas., and allow designers to embed working controls in their models at an early stage in the design cycle. The Behavior Construction Kits [74] and Electronic Blocks [97] provide simple interactive physical building elements that can be easily assembled and programmed, and are aimed children for playful learning.

The iStuff framework supports the development of physical interfaces in ubiquitous computing scenarios, where users interact with multiple networked, distributed and heterogeneous computing platforms [4]. The the iStuff PatchPanel component enables runtime re-mapping of interface devices to application functions; similarly, the ICON toolkit provides a mechanism for input adaptability, allowing heterogeneous input devices to be mapped onto personal computer GUIs [23].

The research literature includes several examples of physical interfaces that allow users to modify the interface configuration during use. In some cases, the intention is provide a mechanism for customisation, and afford personalisation of the user interface. This is the case with Ungvary and Vertegaal's design for the SensOrg cyber-musical instrument, which supports a high degree of interface flexibility in order to accommodate individual user's ergonomic and cognitive requirements [84]. A separate example is the the development of a system that allows a touch sensitive tablet to be partitioned into discrete virtual controls [11]. By placing interchangeable cardboard templates on the tablet, users can partition the surface of the tablet into a set of discrete control areas and "virtual" input devices, which can then be assigned to dedicated interface functions.

One final category includes designs where the act of constructing and assembling physical objects is meaningful: the process of adapting the interface configuration, bringing together different interface elements, or arranging them in space forms part of the interaction. The Triangles system supports information access through exploration of interface configurations [36]. Block Jam allows users to arrange blocks to construct musical structures [64]. Also in the musical domain, the reactTable allows musicians to program a complex synthesiser by arranging and manipulating physical tokens on an interface surface [48].

2.5 Discussion

The design for the VoodooIO interface is deeply influenced by the train of thought behind the work discussed in "A Language of Controls". Whenever possible, the design intentionally makes use of interface elements that are intended be familiar to users of technology, and which are suggestive of their function through their form. An example of this is the choice of machine-like user controls described in Section 5.6, or the the way controls connect to the substrate material via a sharp pin-like connectors - reminiscent of everyday interaction with notice boards and pushpins.

With respect to the discussion about the "Right Tool for the Job": interaction with VoodooIO typically follows a space-multiplexed scheme, where users interact with multiple and discrete controls, and each control can be coupled to an individual software function. The ability add more controls to the interface, as they become necessary, obviates the need to reuse controls in a time-multiplexed manner. In addition, VoodooIO supports a wide variety of different control types, providing users with a wide range of specialised tools with which to carry out the interaction.

The focus of the user experience with VoodooIO is on the physical elements of the interface: the substrate material and control objects. The design aims leverages the user interaction experience on the advantages of making the interface increasingly physical, and is motivated by the same line of reasoning that has inspired tangible user interface

(TUI) research. Interaction with VoodooIO makes extensive use of the ability to arrange objects (controls) in space (substrate areas). Spatial arrangement of controls is not normally tracked or interpreted by the system; rather, it is intended simply as a mechanism that allows users to arrange controls in meaningful and practical ways. Nonetheless, related work has explored extensions to the VoodooIO technology to support tracking of control arrangements on rectangular substrate areas ([52], [7]).

VoodooIO is not a classic example of a TUI in that it lends itself to be used as a more traditional peripheral to a graphical user interface (e.g. the application examples described in sections 7.2 and 7.3). In this sense, the interface does not inherently provide a "close coupling between bits and atoms," any more than a computer keyboard provides a close coupling between physical keys and digital characters. However, this does not prevent the use of VoodooIO as a platform to develop novel forms of TUIs, enabling scenarios where the interaction takes places exclusively with physical objects and without graphical displays. Sections 7.5 and 7.6 of this thesis present two examples of such interfaces.

Finally, the work discussed under the heading of "Room for Change" is of particular relevance to the efforts of realising a flexible physical interface through the development of VoodooIO. When end-users are given the ability to construct and modify the interface configuration, they are also exposed to design decisions and choices that are normally reserved for expert interface designers. Informed by previous practice, VoodooIO addresses these concerns by providing end-users with accessible and understandable mechanisms for physical construction (c.f. Chapter 5, Physical Design) and software configuration (c.f. Chapter 6, Application Support Tools).

The remainder of this thesis documents the development of a user interface paradigm (a novel interface design and associated style of interaction) that brings together many of the design philosophies and research findings that have been discussed in this chapter. The VoodooIO design takes into account the importance of developing interfaces that are suggestive of their function through their form; the benefits of supplying users with the right tools to carry out specific interaction tasks, the advantages of interaction via manipulation of physical objects that can be arranged in real-world space, and the ability to change that arrangement as part of the interaction. The goal of the work is to explore a style of interaction that is characteristically flexible, where users play a central role in constructing, adapting and modifying the physical control devices that constitute the human-computer interface.

Chapter 3

An Architecture for Networked Controls

Computer interface equipment has evolved into a wide array of devices with different form factors and capabilities. Some device types, such as PC keyboards and mice, are fairly generic to many types of computing application. Others – such as joysticks, graphics tablets and mixing desks – are aimed at more specific applications, from game-playing through to graphic design or music production. In some cases – such as the keypad on a mobile phone, a machine control panel or a vehicle dashboard – the interface devices are unique and permanent to a particular instance of a system.

We tend to think of these devices as indivisible interface elements: a keyboard, a gamepad, a mixing desk. In some cases, devices are interchangable as a result of a standardized protocol or connection mechanism. A keyboard can be unplugged from a USB port on a PC and replaced by a more ergonomic version, and a TV remote control can be swapped for another model that emits compatible infrared codes. In general, this is as far as modularity and reconfigurability in the physical human computer interface extends to: swapping one device for another of the same type, with some degree of variation between form factors, control composition or input/output capabilites.

Conceptually, it is possible to desconstruct individual examples of interface devices into even smaller interactive units than we normally perceive them to be: a keyboard into a set of keys, a gamepad into a joystick and some buttons, a mixing desk into rows of sliders and feedback lights. At this level of granularity, it is possible to see a large of overlap between what would normally be considered disparate and specialized devices: most interface devices are actually composed of numbers of individual controls of finite types, both input (e.g. buttons, sliders, dials, knobs, switches) or output (e.g. displays, buzzers, vibrators, lights) in a given arrangement.

This chapter deals with the technological paradigm shift which is involved in deconstructing interface equipment beyond what we traditionally consider to be a device as a whole and into its constituent interactive elements: basic input/output controls. The goal of this deconstruction is to enable interface equipment that is reconfigurable and modular at the control level. Each control element becomes an interface device in its own right, and collections of such controls can be brought together in an ad hoc manner to create composite control devices which are continuously reconfigurable in their composition.

3.1 Hard-Wired vs. Networked Controls

In traditional interface device design, individual controls are hard-wired: soldered to a printed circuit board, connected to controlling circuitry via conductive traces and fixed in place by a rigid casing. In terms of cost and complexity this design is suitable if we consider the device as a whole – rather than the individual controls it is composed of – as an atomic and indivisible unit.

Embedded systems – special-purpose computers designed for a specific and pre-defined task – are continually becoming smaller in size and more cost effective to develop due to advances in manufacturing and the increasingly widespread use of microcontrollers, integrated circuits (IC) and other electronic components. It is now possible to embed processing, memory and communication capabilities into increasingly smaller devices, giving rise to such technologies as distributed sensor networks that are composed of large numbers of minimal, self-contained and low-cost network nodes, capable of autonomous and decentralised operation.

Given the ready availability of the same technologies that enable sensor networks, it it becomes possible to consider an alternative design for interface equipment. Rather than being hard-wired to a fixed circuit configuration, controls can be implemented as embedded network nodes. In this approach, the interface circuitry is replaced by a network bus, and individual controls to be added or removed from the network constellation in a modular way (cf. Figure 3.1).

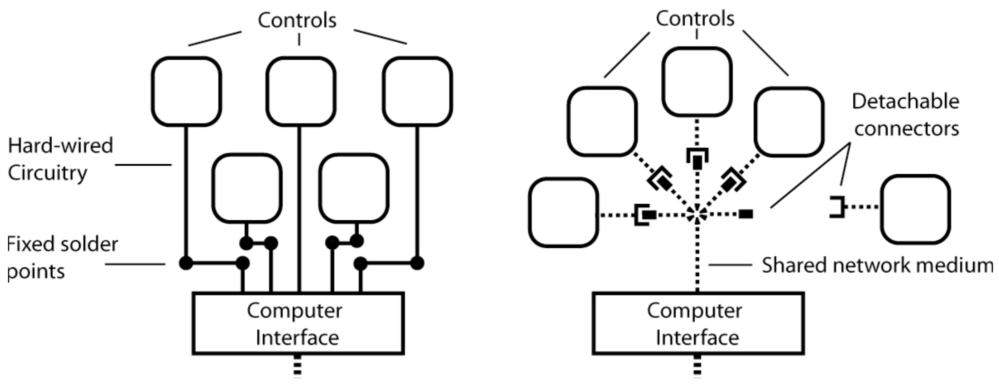


Figure 3.1: Traditional hard-wired interface hardware (left) vs. a network-based design (right).

The interface architecture presented in this chapter distributes the complexity towards the outer edges of the control structure. Each control becomes a user interface device in its own right, embedding all the circuitry necessary to power it, drive it, and give it a common network interface. The design affords a high degree of control-level modularity, with the implication of enabling reconfigurability of the control structure without the limitations imposed by predefined designs.

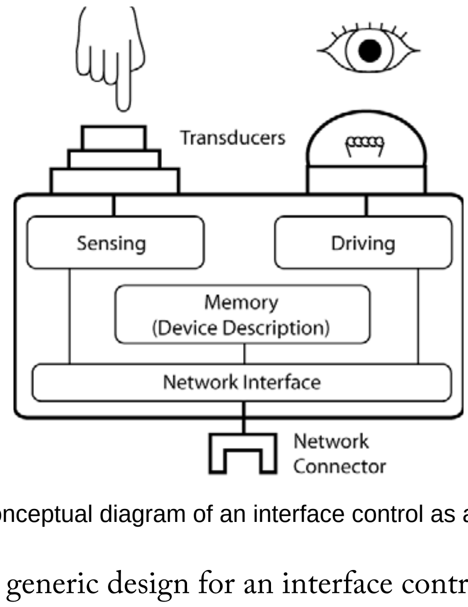


Figure 3.2: Conceptual diagram of an interface control as a network node.

Figure 3.2 illustrates a generic design for an interface control as a network node: one or more transducers (which can be manually manipulated and electronically sensed) provide a physical way for users to perceive output or effect input; IC modules with input/output capabilities are used to sense and drive the state of the transducers; non-volatile memory is used to store a unique identification, type information and device-description of the control (necessary to identify a control in a changing network constellation); a network interface allows the I/O and memory to be remotely accessed and controlled; and some form of connectors or antennae that provide a physical link to the network medium.

From a user's point of view, the defining characteristic of the design is that the interface controls, as well as being subject to traditional forms of physical operation, are also highly modular and hot-swappable. It empowers the user with the possibility of adapting and reconfiguring – adding and removing controls ad hoc – as part of the interaction that can take place while using the interface.

3.2 Issues in Networking Controls

A direct consequence of a network-based design is that individual control elements must be more complex than equivalent controls in standard hardware equipment. A keyboard key, for example, is simply a switch that closes a binary circuit when pressed. The intelligence in the circuit is centralised in a single controlling device, which monitors the state of a large number of keys. In the network-based model, each control requires its own network interface to communicate across a shared network medium.

The choice of network technology – the underlying mechanism used by control nodes to communicate – plays a central factor in the rest of the design. A wireless solution might seem more flexible than a wired network, as nodes are not tethered to a physical location by cables or wires. However, wireless networks tend to require more complex networking circuitry, are more susceptible to noise in the communication channel, and necessitate that nodes are battery powered (raising the problem of having to keep constellations of nodes continuously charged). Conversely, contact-based networks tend to be more robust, simpler, and – in principle – can directly supply power to connected nodes.

In order to support a high degree of reconfigurability, modularity is a central concept in the design. Individual controls need to be reusable (i.e. it should be possible to add and remove controls from the network) and interchangeable (i.e. it should be possible to replace one control type with another). The design needs to support a variety of different control types with different input and output capabilities and in an extensible manner, allowing for the development of new types of controls.

A major challenge in designing user interface equipment of any kind is ensuring that it is responsive to user interaction. From a usability perspective, perceptive delays between a user performing an action and a system response can result in a frustrating or confusing to experience. This is an issue, given the network-based nature of the design; with many control devices sharing a single physical communication medium, it is necessary to consider how the system can remain responsive under different conditions.

Finally, the design must address the need to interface between the hardware equipment and software applications. This involves implementing the necessary subsystems that manage and drive the network, and creating abstractions from low-level hardware details and implementation-specific processes to high-level software representations.

The rest of this chapter presents an architecture for interactive networked controls. The structure of the following sections reflect the issues that have been raised above, following a bottom-up approach: Section 2.4 describes the networking standard that is used as the underlying communication mechanism in the implementation. Section 2.5 details a modular design for physical interface controls as network nodes. Finally, Section 2.6 deals with the implementation details of the software stack that ensures a responsive behaviour from the system and abstracts from the low-level hardware detail to provide an application-level interface.

3.3 Enabling Network Technology

The architecture for networked control uses an existing networking technology as basis for its implementation. The choice of networking technology, called the 1-Wire communications bus, is central to the design and affects many aspects of its implementation. This section provides an overview of its key concepts and operation as an introduction to the design.

3.3.1 The 1-Wire Communications Bus

The 1-Wire communications bus – also known as MicroLAN – has been developed by the Dallas Semiconductor company as a way to transmit data using only minimal wiring: a single data line (plus a second one for ground) [61]. It provides low-speed (16 kbps), and low-power devices (up to 5V) that can be parasitically powered from the data signal, allowing them to operate without requiring any additional power supply lines.

A 1-Wire network follows a master-slave model, with a single controlling master device that mediates communication between a number of slave devices on a multidrop bus. The master (which can be a PC equipped with a 1-Wire adapter, or a dedicated microcontroller) ensures that only one slave device at a time "talks" on the bus, and prevents collisions in data transmission from different devices. The bus supports bi-directional communication between master and slaves: the master can both write to slave devices and read data back.

The 1-Wire standard specifies that every 1-Wire-compatible device includes a globally unique, 64-bit address. The address encodes the particular type and capabilities of the device (see 1-Wire Components, below), and also provides a way to uniquely identify a particular device from all others of the same type.

3.3.2 1-Wire Components

Dallas Semiconductor, the company that developed the 1-Wire networking standard, also produces a range of 1-Wire-compatible components. These are commercially available as low-cost, small-footprint integrated circuits (IC). The minimum specification for one of these components is a 2-pin network and power supply interface (data/power, plus ground) and enough ROM to hold the factory-programmed 64-bit unique address.

In addition, 1-Wire components can include a number of additional functions such as temperature and humidity sensors, real-time clocks, programmable memory and I/O pins. Only the most useful components for the purposes of our discussion are listed in table 3.1. For the sake of clarity, we will subsequently refer to these devices by the mnemonic listed on the right-hand column.

1-Wire Component Type	**Features**	**Mnemonic**
DS2406	1 Kb non-volatile memory, 2 x digital I/O pins	"memory component"
DS2408	8 x digital I/O pins	"digital I/O component"
DS2450	4 x analogue input pins	"analogue input component"
DS2890	Single analogue output	"analogue output component"

Table 3.1. Some 1-Wire component types.

3.3.3 Conditional Device Searches

The basic principle of operation of a 1-Wire network is the ability of the network-master to perform a conditional search on the network, to which connected 1-Wire components respond with their unique addresses. We can consider a typical search sequence by the network master to be as follows[1]:

```
< Start of search sequence >
1. Specify search conditions
2. Retrieve address of first matching component address
3. While there are more matching components
     4. Retrieve next matching component address
< End of search sequence >
```

A search can be generic, causing connected 1-Wire components of every type and to sequentially respond with their unique address. A search can also be targeted to a particular type of component: only components of that particular type will respond with their address. The search conditions can be as specific as targeting a single component whose address is already known, in order to test for that component's presence on the bus. Searching for a component has the additional effect of "selecting" it for further

1. The details of 1-Wire signal-level communication, although interesting, are outside the scope of this overview. A good description can be found in the official protocol specification. For the purposes of the discourse the following descriptions accurately reflect the operation of a 1-Wire network, but may gloss over some of the low-level details that are not essential for understanding the design of the architecture.

communication. By using conditional searches, a 1-Wire master is able to select either groups or individual components on which to perform shared or targeted commands.

3.3.4 Activity Detection

1-Wire components are, on the most part, dependent on the 1-Wire master for their operation and need to be "told" exactly what to do at each point in time. There is one case, however, where 1-Wire components can act with some degree of autonomy: some 1-Wire device types can be configured by the network master to enter a special alarming state when some pre-programmed conditions are met. This alarming state can be used as an activity flag, to indicate that the component has experienced some change due to external stimulus.

For example, the digital I/O components can be programmed by the master to enter an alarming state when one or both of its I/O pins sense a change from a high-to-low state, or vice-versa; the analogue input components can be configured to enter an alarming state whenever the sensed voltage level on one of its input exceeds or falls bellow specified thresholds. Once a device enters an alarming state, it will remain in that state until it is explicitly reset by the network master.

The network master can carry out a special conditional search to which only devices which are in an alarming state will respond. This provides a mechanism for quickly detecting changes experienced by individual devices, without requiring the master to perform a general search and examine each component's state in turn.

3.3.5 Read/Write Operations

After selecting a specific component by using a conditional search, the network master can then perform a sequence of read/write operations to its memory. It may, for example, write to the component's internal register and configure some operational parameters. These parameters may include the conditions to enter an activity alarming state, or perform a reset to a non-alarming state after some activity has been detected. Some component types come equipped with general-purpose memory that can be used to store arbitrary data. For those component types that support some form of output, such as the digital I/O and analogue output components, the master can perform a write operation to set their output levels.

A read operation can be used to retrieve data on configuration, operational state or sensed input. A read/write operation is preceded by the network master targeting a component using its unique address, performing a write operation to specify the memory addresses that need to be read, and then reading the response as the component transmits it on the network bus. The duration of different read/write operations will vary depending on the number of bits that must be transmitted.

3.3.6 Benefits and Issues of 1-Wire for Networked Controls

The design of the 1-Wire bus and range of 1-Wire components makes it an interesting technology with which to build a system for networked control. An implementation based on 1-Wire does present some challenges – in particular, the data bandwidth of the bus is limited, so network access has to be carefully managed. On the other hand, in comparison to faster serial communication standards (such as SPI or I^2C) the 1-Wire protocol requires the simplest cabling to operate. Simple cabling requirements allow for greater freedom in exploring physical connection mechanisms, as the bus necessitates only two points of contact between devices and the network cabling.

The use of short-range wireless communication protocols, such as Zigbee or Bluetooth, would negate any cabling requirements at all. However, wireless devices necessitate more complex circuitry and consume more power. The issue of power supply would be an issue, as components would need to be self-powered and be kept continually charged. With 1-Wire, devices can be parasitically powered from the network bus and draw up to 500mA at 5V.

Other beneficial features of the 1-Wire technology include: factory-set unique ID for every 1-Wire component; limited component "interrupt" capabilities through activity detection; small IC footprints; low power consumption, and support for a large number of simultaneously connected components (~200) on a single bus.

3.4 Blueprint for Controls as Network Nodes

The architecture for networked controls accommodates a variety of different control types, with each type characterised by the set of transducers that afford particular user input/output capabilities (e.g. a "switch" control type). 1-Wire-capable IC devices are used to sample and drive the transducers, and communicate control data over the network protocol. In addition, each device contains some non-volatile memory that stores a control-description for each individual control instance. The description includes information about the technical implementation and composition of the control, as well as a unique identifier number, and assigned control type.

3.4.1 Hardware Design

The minimal specification for a control device is based on a single 1-Wire memory component, which is designated the control's *primary component*. The 1-Wire address of the primary component is used as the unique identifier for the entire control, and the 1kb of non-volatile memory of the component is used store a control description. Depending on the requirements of the particular transducer(s) that a control type may employ, the design can optionally include any combination of additional *secondary components* to sense and drive these transducers. Figure 3.3 illustrates a generic control design.

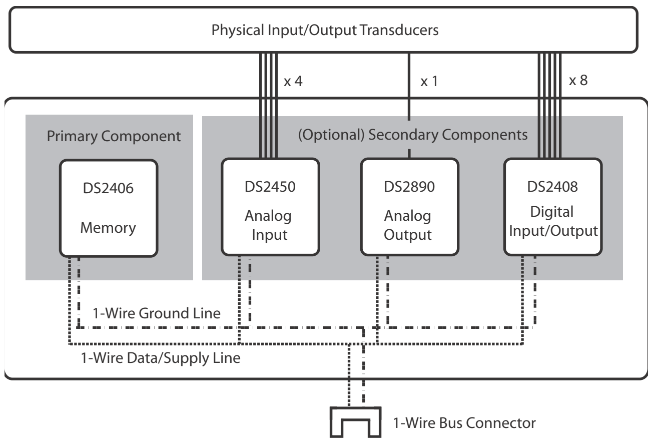


Figure 3.3: Generic design for control devices using 1-Wire components.

Analog input components are suitable for sensing transducers which output a variable voltage, such as potentiometers (e.g. rotary dials, sliders or knobs) or some basic types of sensors. Analog output components can be used to drive transducers which require a varying voltage as input (e.g. controlling the brightness of an LED). Digital Input/Output components can either drive transducers that require digital input (e.g. turning on and off the individual segments in a numeric display) or provide simple binary output (e.g. detecting a button-press).

An internal 1-Wire bus interconnects the primary component and any secondary component that might be included to support the transducers built into the control. Transducers which require power to operate can be supplied from the same bus, with some limitations. The design is generic and open-ended to allow for a wide variety of control types to be implemented, each with a distinct operational affordance.

3.4.2 An Example Control Type

A example implementation of a control of type "ROTARY_INPUT" is shown in Figure 3.4. The control offers the user an rotary potentiometer for input (a "dial" or "knob" to a user). The primary DS2506 device has the unique 1-Wire address "AB0000001."

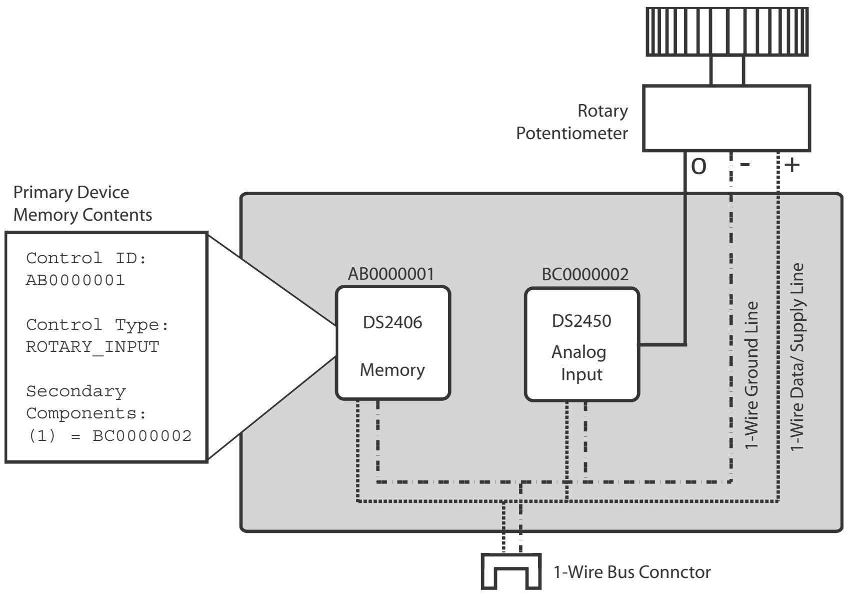


Figure 3.4: Example implementation of a "Rotary Input" -type control.

The rotary potentiometer outputs an analogue value (varying between 0 and 5 volts). In order to sense the value of the potentiometer, a secondary component has been added to the design: a DS2450, with up to four analogue inputs, and shown here with the address "BC0000002." Note that, in this case, only one of the four available inputs in the analogue input component is actually used to sample the output of the rotary potentiometer. If more than four inputs were required (for example, to support a more complex transducer) then any number of additional secondary components can be added to the design of the control.

3.4.3 Control Description

The memory of the primary component stores a control description. This information is programmed into the one-time-programmable memory of the component when it is first assembled, and provides all the details which are necessary for the system to uniquely identify the control and interpret its capabilities.

The description contains a unique identification (ID) for the control. The ID is derived from the primary component's unique 1-Wire network address. Since the manufacturers of 1-Wire components assure that each comes factory-programmed with a globally unique network address, this ensures that no two control instances will share the same ID.

The description also includes a type label which describes the control, in this case it is: "ROTARY_INPUT." Finally, the description includes a pointer to any secondary devices that might be included in the control; in this case there is only one: an analogue input component with address "BC0000002."

3.5 System Architecture

At the core of the system, and complementing the hardware design, is a software architecture that bridges between the hardware devices and software applications. The software is implemented as a stack, which serves two purposes: 1) it manages, controls and mediates communication at the network level, and 2) it abstracts from the hardware detail into object oriented and event-based software representations. Figure 3.5 provides an overview of the complete design, with a focus on the key elements of the software stack.

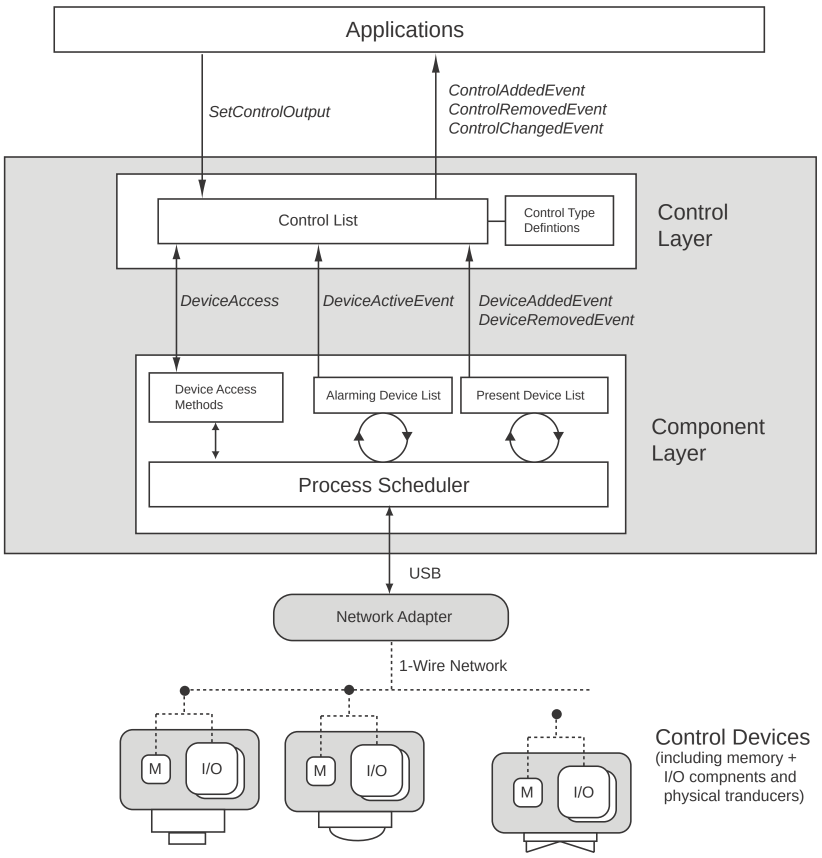


Figure 3.5: Overview of the architecture, with a focus on the software stack.

The stack is designed as a background process that runs as a service on a computer running the Microsoft .NET 2.0 Framework. A Dallas Semiconductor USB-to-1-Wire adapter provides a physical-level interface to a 1-Wire bus, to which control devices connect.

The stack is divided into two levels: A lower Component Layer and a higher Control Layer. The Component Layer deals with all the processes which are specific to the network bus and multiple network components that reside in each control device: acting as network master, monitoring the state of the network and carrying out read/write operations.

The Control Layer provides one level of abstraction above the Component Layer. This layer presents a view of the network as a constellation of composite control devices, each containing one or a number of individual memory and input/output components. It provides an object-oriented representation of the control devices as user interface elements, translating analog and digital inputs and outputs into user interface events.

3.5.1 Detecting User Interaction

User interaction is abstracted into three high-level events that represent the basic actions a user can perform with the controls: ControlAddedEvent, raised when a new control is connected to the network; ControlRemovedEvent, raised when an existing control is disconnected from the network; and ControlChangedEvent . Each event is preceded by a process that is triggered either by a user (manually adding, removing or manipulating a control device) or by an application (setting the output state of a control).

The purpose of the remaining sections of this chapter are to describe in detail the software processes that have been implemented to enable this behaviour, illustrating how the different layers of the software stack come into play.

3.5.2 Process Scheduler

One of the main challenges in using the 1-Wire bus as basis for an interactive system is its limited bandwidth. Network access is a scarce resource, and needs to be carefully managed. At the core of the Component Layer is a subsystem that schedules any communication between the software stack and the network bus. It uses a time-slotted protocol to manage three different processes: 1) A Presence Detection cycle, which detects addition and removal of primary components from the bus; 2) An Activity Detection cycle, which monitors activity in the I/O of any secondary components, and 3) Read/Write events, which execute any read/write requests that need to be carried out on individual components.

Each process is subdivided into individual frames, where one frame represents an atomic operation that can be performed on the network bus. Frames of the three different cycles can be interleaved in a configurable manner, in order to give precedence to one process over the other.

The Presence Detection cycle is the process of continually scanning the network to determine what primary components are present on the bus at any point in time. This is achieved by carrying out a conditional search on the bus (c.f §2.4.3 Conditional Device Search), where only primary (of memory type DS2406) components are targeted. A presence frame constitutes the ability to read a single component address from the bus. A complete cycle involves a single full iteration over the addresses of all present devices.

The Activity Detection cycle operates in a similar way, the only difference is that the conditional search targets only components that have experienced some form of activity in their I/O pins (c.f. §2.4.4 Activity Detection). Again, this is done sequentially, frame-by-frame, where each activity frame represents the ability to read the address of a single component that is experiencing activity.

Read/Write frames are not cyclical, and occur only at the request of the Control Layer, for example, to configure a particular component's activity detection parameters or read the values of its inputs.

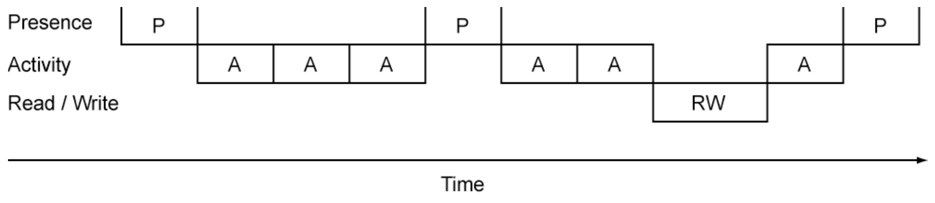


Figure 3.6: Example message protocol configuration.

Figure 2.6 shows an example sequence where the protocol has been configured with a ratio of 1 Presence Detection frame (P) to every 3 Activity Detection frames (A). Read/Write (RW) frames, when they occur, take precedence over the other two.

The ratio of P to A frames determines how quickly the system is able to detect added/removed components and active components, i.e. a high number of sequential P frames results in addition/removal of components in being detected quickly, but the timely detection of component activity suffers as a result. Conversely, the protocol can be configured to maximise the timely detection of user manipulation of controls by assigning a high A to P ratio. The exact effects that the protocol implementation has upon user interaction with the system are explored in detail in Chapter 4.

3.5.3 ControlAddedEvent

The ControlAddedEvent is raised by the software stack as a result of a user connecting a new control to the network. The steps that lead to this event being detected can be summarised in the following sequence:

1. A new control device is connected to the network by the user.
2. Its primary component is detected in a P frame by the Scheduler. A software object is instantiated to represent it.
3. If it is not already present in the Present Device List of the Device Layer, it is added to the list and a ComponentAdded event is raised with a reference to the component object in question.
4. The control layer handles the ComponentAdded event by
    1. Retrieving the control description from the memory of the primary device.
    2. Creating a software object to represent the control device of the appropriate type (containing references to both its primary component and any secondary components).
    3. Initalizing any secondary components to enter an activity alarm state when the control is manipulated.
5. A Control Added messsage is raised, which includes a reference to the control object in question.

This process is illustrated in Figure 3.7. In this example, a control of type "BUTTON," with a primary memory component of address AB00001 and a single secondary digital I/O component with address "CD00002" is connected to the network. The top of the figure shows a detailed view of the sequence of P, A and RW frames as processed by the Process Scheduler.

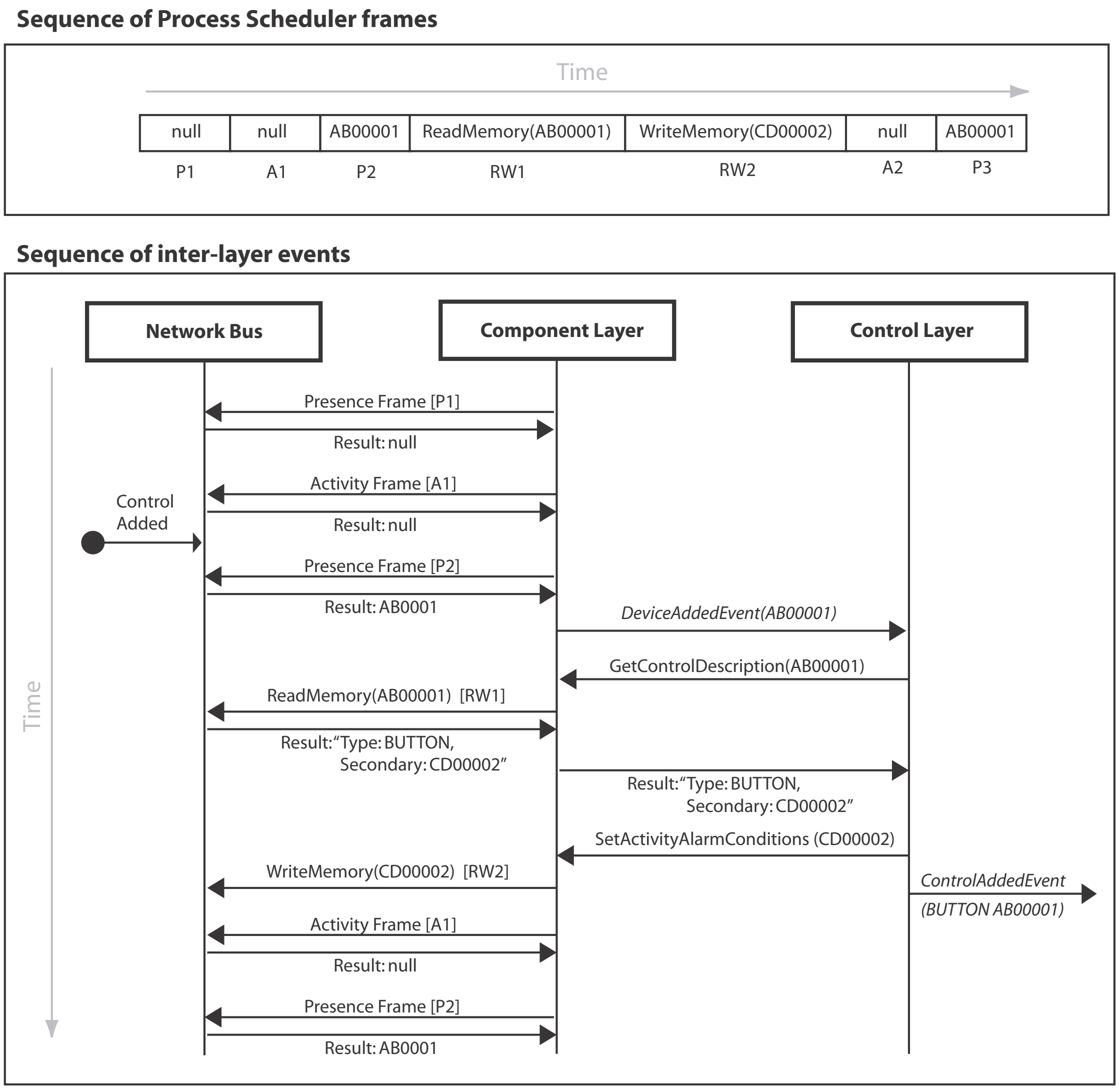

Sequence of Process Scheduler frames
Time
null
null
AB00001
ReadMemory(AB00001)
WriteMemory(CD00002)
null
AB00001
P1
A1
P2
RW1
RW2
A2
P3
Sequence of inter-layer events
Network Bus
Component Layer
Control Layer
Time
Presence Frame [P1]
Result: null
Activity Frame [A1]
Result: null
Control Added
Presence Frame [P2]
Result: AB0001
DeviceAddedEvent(AB00001)
GetControlDescription(AB00001)
ReadMemory(AB00001) [RW1]
Result:"Type: BUTTON, Secondary: CD00002"
Result:"Type: BUTTON, Secondary: CD00002"
SetActivityAlarmConditions (CD00002)
WriteMemory(CD00002) [RW2]
ControlAddedEvent (BUTTON AB00001)
Activity Frame [A1]
Result: null
Presence Frame [P2]
Result: AB0001

Figure 3.7: Sequence of events leading up to the ControlAddedEvent being raised.

3.5.4 ControlRemovedEvent

The ControlRemovedEvent is raised by the software stack as a result of a user disconnecting an existing control from the network. The steps that lead to this event being detected can be summarised in the following sequence:

1. An existing control gets removed from the network by the user.

2. By comparing the results of a complete Presence Detection cycle by the Process Scheduler against its Present Device List, the Device Layer determines that the

primary memory component of the control is no longer present on the network, and raises a ComponentRemovedEvent to this effect.

3. The Control Layer handles the ComponentRemovedEvent by finding the control which contains a primary component of a matching address in its internal Control List, removes it from the list and raises a ControlRemovedEvent to this effect, including a reference to the removed control object in question.

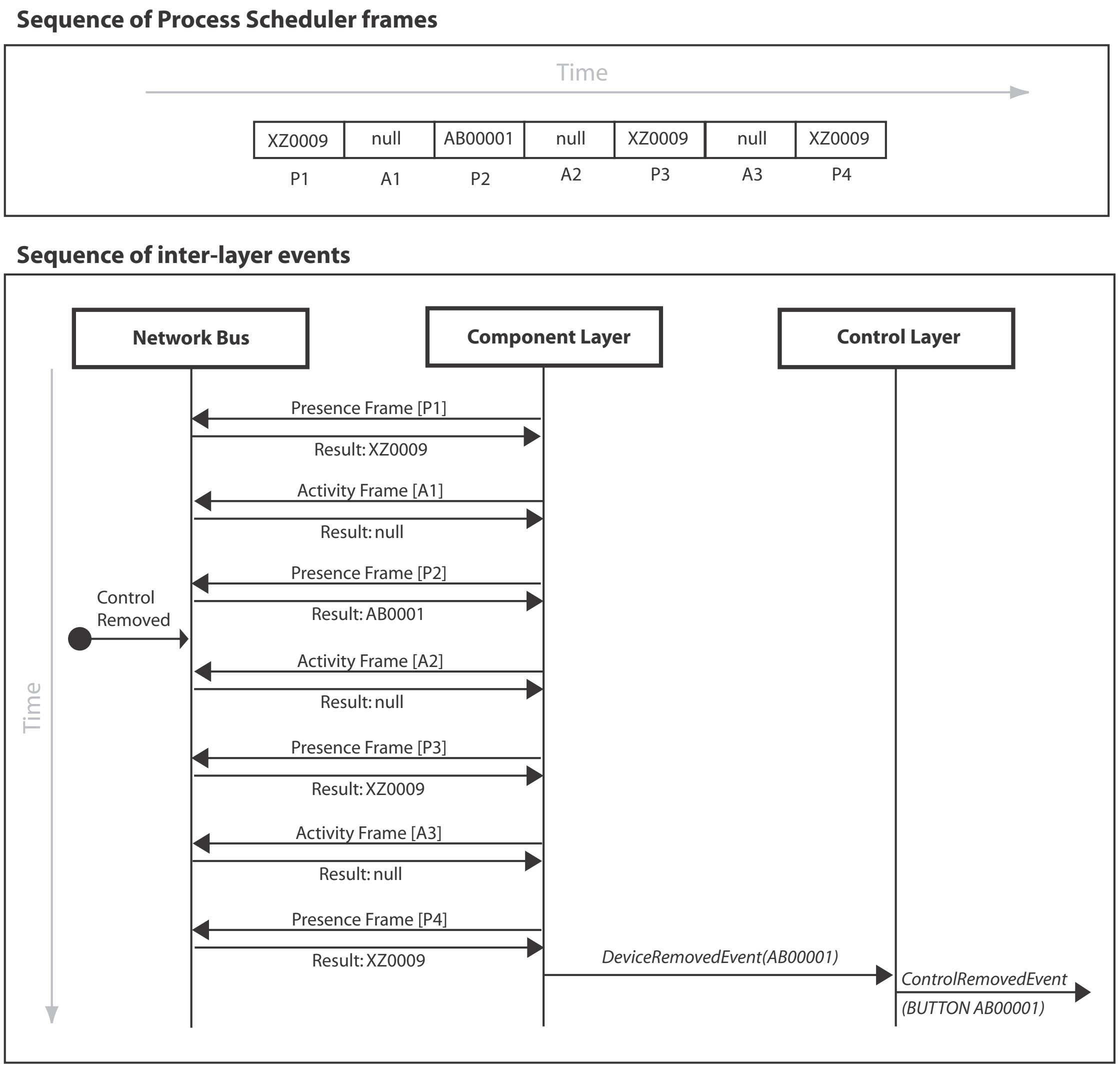


Figure 3.8: Sequence of events leading up to the ControlRemovedEvent being raised.

This process is illustrated in Figure 3.8. In this example, a control of type "BUTTON," with a primary memory component of address AB00001 is removed from the network. A second control, with primary component address "XZ00009" is also present on the network at the beginning of the sequence, and remains connected after the first control is removed. The top of the figure shows a detailed view of the sequence of P, A and RW frames as processed by the Process Scheduler.

3.5.5 ControlChangedEvent

The ControlRemovedEvent is raised by the software stack either as a result of a user manipulating an existing control from on network (e.g. a user pressing a button on the control) or as confirmation of an application's request to change the output state of a control (e.g. the application requesting for a light on the control to be turned on). The steps that stem from user interaction and lead to this event being detected can be summarised in the following sequence:

1. A user manipulates a transducer on the control.
2. A secondary I/O component on the control enters an activity-alarming state due to a change in its sensed input.
3. The alarming component is detected in an A frame by the Scheduler. A software object is instantiated to represent it, and is added to the Device Layer's Alarming Device List. A ComponentActiveEvent is raised to this effect, which contains information about the active component as well as its sensed input.
4. The ComponentActiveEvent is handled by the Control Layer by finding the control (in its internal Control List) which contains the active secondary component. Based on the sensed input of the component, the control updates its internal representation of state (e.g. sensed input == 5V => button == "pressed").
5. The activity alarm of the secondary component is reset, readying it for detection of further user manipulation.
6. A ControlManipulatedEvent is raised.

In a similar way the event can be raised by the control's state changing as a result of an application's request for output that results in a secondary component's output levels being altered.

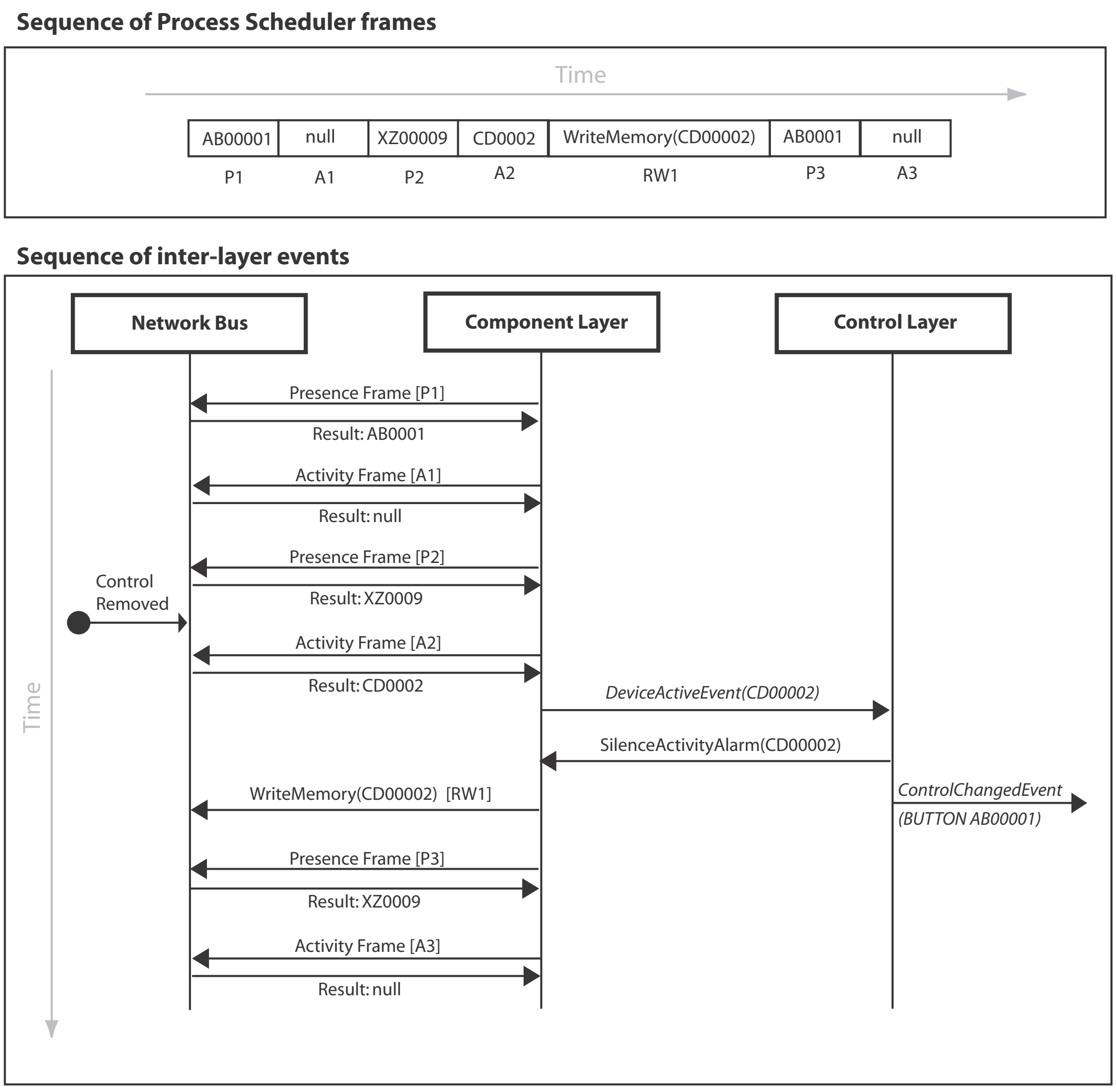


Figure 3.9: Sequence of events leading up to the ControlChangedEvent being raised.

This process is illustrated in Figure 2.9. In this example, a control of type "BUTTON," with a primary memory component of address "AB00001" and a secondary I/O component of address "CD00002" is manipulated by a user (e.g. its button transducer is pressed). A second control, with primary component address "XZ00009" is also present on the network during the process. The top of the figure shows a detailed view of the sequence of P, A and RW frames as processed by the Process Scheduler.

3.6 Discussion

This chapter introduced the idea of deconstructing physical human computer interface equipment into a set of modular controls which can be individually added, removed and manipulated by users. The concept entails implementing interface hardware as a network of controls as way to overcome the inflexibility that is usually imposed by hard-wired circuitry in traditional interface device designs. The idea is validated by a prototypical implementation based on the 1-Wire networking technology, which includes a blueprint for developing users controls of various types and capabilities as network nodes which can be added and removed from the network in an ad hoc way.

Other interface construction tools reported in the HCI literature have implemented interface elements as modular devices, and are motivated by some of the same problems that are targeted by our architecture for networked control. The Phidgets platform includes diverse input/output devices into a set of modular components that share a common USB interface [37]. Like Phidgets, our architecture supports the ability to assemble collections of control devices to form custom interfaces, and provides a standardised API to program them. iStuff also provides physical interface components, but in addition supports flexible mapping of both wired and wireless interface events to target devices during run-time [4]. Papier-Mâché provides toolkit support for interfaces that use passive and visually tracked objects as tangible input [53]. Calder [56] and d.Tools [43] are a more hardware focused toolkit providing phidget-like controls in smaller form factor targeted at product prototyping. Switcharoo is also targeted at interactive product design but uses very lightweight RFID-based input components [2]. The Triangles tangible interface is composed of reusable triangular shapes that can be assembled into structures [36]. Similar to our design, each triangle includes a small microcontroller that can communicate to a master device via a serial data bus. The Siftables project uses techniques from wireless sensor networking to develop minimal interface devices that include sensing, graphical display and wireless communication [62].

In relation to these tools, our architecture for networked controls is comparable with Phidgets and iStuff in providing active interface devices rather than support for passive objects as input, and with Calder, d.Tools and Switcharoo in targeting rapid device composition. Like the d.Tools platform, a particular concern of our architecture is that it is extensible, and able to support a wide and growing number of possible control devices. There is also a similarity with Phidgets in the set of interface building blocks provided, but our design is distinct in providing a more lightweight solution, which requires minimal wiring to network devices. The lightweight design has a direct impact on how the architecture can be deployed, and on how it can be integrated into concrete physical artefacts (c.f. Chapter 5, Physical Design).

Implementing interface equipment as a network of controls differs considerably from standard interface device design practice. The overhead caused by replacing hard-wired circuits with a network has an effect on the performance of the system, and raises questions such as: How quickly can user interaction with the system detected? Does the performance of the system scale with varying quantities of controls connected to the network? Is this performance adequate, and is the design adequate to support user interaction requirements? The next chapter focuses on characterising and evaluating the performance of our design, in order to provide answers to these questions.

Chapter 4

System Response Time

The System Response Time (SRT) of an interactive system is the period of time that elapses between a user performing an action on its interface and the moment the system can reply with a reaction. A consistently fast and stable SRT is usually considered to be a desirable characteristic in a user interface [35], [39], [5].

Delayed SRTs are usually the result of some heavy computation that must be carried out by an application before the system can elicit a response, but sometimes delays in response times can be the result of slow user interface devices and interface processes. Many interface devices, such as mice and keyboards, contribute very little to the overall SRT of a system, sampling and providing input to the system at a very high rate. Others, such as biometric scanners or vision-based gesture recognition systems, can require a considerable amount of time to capture and process input, adding a noticeable increase to the responsiveness of the system. An understanding of the performance of individual interface elements can be key to characterising the capabilities and limitations of any particular human-computer interaction technology.

Chapter 3 presented an architecture for interface equipment as a network of controls, which offers increased reconfigurability over traditional interface devices. From a system perspective, the performance of the design is characterised by the SRT period that elapses between a user adding, removing or manipulating a control on the network bus, and the action being communicated to an application as a software event.

This chapter presents an analysis of a series of experimental results that measure the SRT of the design across a number of conditions which have an effect on the performance of the system, in particular the frame ratio configuration of the Process Scheduler (cf. §3.6.2), and the control load, i.e. number of controls that are present on the bus at any one time.

The remainder of this chapter is structured as follows: We begin by describing the test rig and procedure used to carry out the evaluation of the system performance. Section 4.2 then presents a first set of results, which show how the base SRT of the system can be dynamically altered by modifying the frame ratio configuration, optimising for the detection of either addition/removal or manipulation of controls. A second set of results, presented in Section 4.3, characterises the performance of the design with respect to how the base SRT scales with an increasing load of controls on the network. In Section 4.4 we further analyse the impact of the system performance from a usability perspective.

4.1 Experimental Setup

A test rig was used to measure the SRT by logging the time elapsed between effecting some stimulus (adding, removing and manipulating controls on the bus) and detecting the resulting software events (ControlAddedEvent, ControlRemovedEvent, ControlChangedEvent). Figure 4.1 shows a diagram of the setup.

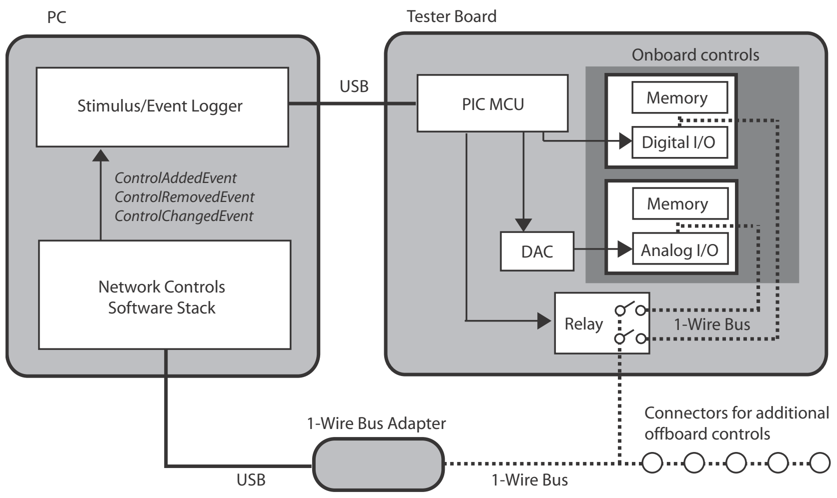


Figure 4.1: Diagram of the closed-loop test rig used to measure system response time.

The test rig is based on a standard setup of our networked controls implementation, as described in §3.6: a PC running the software stack, a USB 1-Wire bus adapter, and a 1-Wire network to which control devices can be connected. In addition, a hardware Tester Board and a software Stimulus/Event Logger application complete the test rig setup.

4.1.1 Tester Board

The Tester Board is a hardware device that can be connected to the 1-Wire bus. Internally, it contains two onboard controls that can be controlled by a PIC microcontroller (MCU). One of the controls is a generic digital input control (equivalent to a button or switch control type) and the other an analogue input control (equivalent to a rotary potentiometer, or slider). These two control types were chosen as being representative of a wide range of possible control implementations.

The MCU provides an external USB interface that allows an automated process to effect stimulus on the pair of onboard controls. The MCU is also able to simulate user input on the digital control by toggling the state of one of its digital output pins, which is directly connected to the control's digital input component. In order to simulate an analogue transducer (e.g. a rotary potentiometer) the MCU controls a digital-to-analog converter (DAC), the output of which is connected to the second control's analog input component. In addition, the controls can be individually connected or disconnected from the 1-Wire bus via a relay switch.

4.1.2 Stimulus/Event Logger

A software Stimulus/Event Logger, running on the same PC as the software stack, completes the testing loop. On the one hand, it registers itself with the software stack to receive ControlAdded/Removed/Changed software events. On the other, it communicates with the Tester Board to control the presence and input state of its two onboard controls.

The Logger software is able to test and record the SRT of the system by using the Tester Board to generate some stimulus on its its onboard controls (connect/disconnect them from the bus, or change their input conditions) and measuring the time period that elapses between the stimulus and a resultant software event being received.

4.2 SRT Performance

The configuration of the Process Scheduler (cf. §3.6.2) has an effect on the SRT performance of the system. Figure 4.2 illustrates two alternate frame-ratio configurations, one with a high P-to-A ratio (P:A = 5:1), and second with a higher ratio of A frames to P frames (P:A = 1:5)

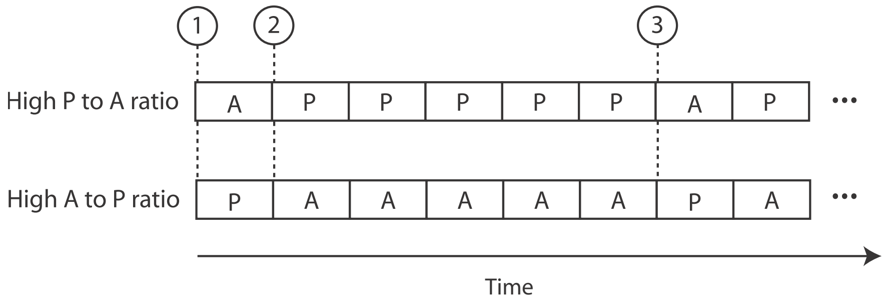


Figure 4.2: Example P-to-A frame-ratio configurations, 5:1 (top) and 1:5 (bottom).

With a high P-to-A frame ratio, the system is optimised to detect addition/removal of controls by increasing the likelihood of the event being quickly picked up in a P-frame. In this configuration, a control added at point (1) will not be detected immediately (worst-case scenario), but the delay is still minimal, with only a single A-frame being processed before the next P-frame takes place. A control added between points (2) and (3) will be detected by the very next frame after the event (best-case scenario).

A high A-to-P frame-ratio reverses the situation. A control which is added at point (1) will be immediately be picked up by the subsequent P-frame (this becomes the best-case scenario). In the worst-case scenario, an addition that takes place immediately after point (2) will not be picked up until the next P-frame, after point (3).

Detection of control removal is optimised in a similar way to the worst-case scenario of control addition, as it requires at least two consecutive P-frames to take place to confirm that the Presence Detection cycle has completed a full iteration over all present components on the bus and the control is absent from this list. A change of control state change (activity) is optimised conversely to addition/removal, with a high A-to-P ratio increasing the likelihood that activity events are detected in a timely way due to the large number of consecutive A-frames and low number of P-frames.

4.2.1 SRT for Control Addition and Removal

An experiment was carried out to determine the effect of the frame-ratio configuration on the SRT of adding and removing a single control to and from the bus. The test rig was set up to first connect, then disconnect, a single control to a bus which was otherwise empty of controls. The variable condition was the frame-ratio, which ranged from 10:1 to 1:10 Presence to Activity frames.

Every condition was tested 500 times, with the times between stimuli delayed by a random amount between 0 and 4 seconds to accurately simulate a user adding or removing a control at an arbitrary point in the frame sequence. Figures 4.3 and 4.4 show the results of the tests, with the error bars delimiting the 5th and 95th percentile of the samples taken.

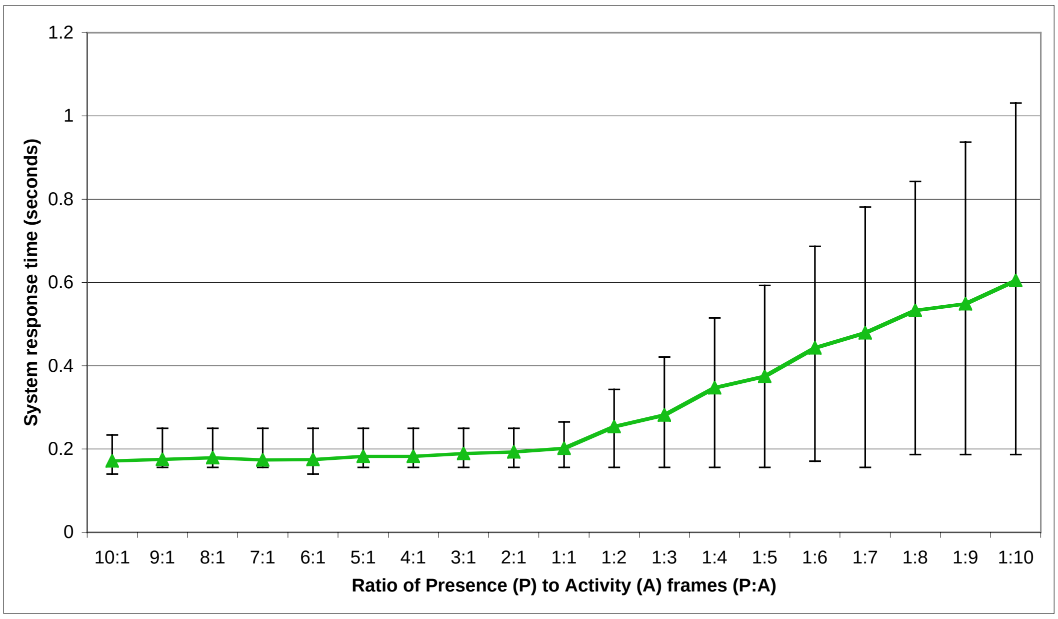


Figure 4.3: Time to detect addition of a control, across varying protocol configurations.

Figure 4.3 shows how the time to detect a control addition increases slowly from an average of 170 milliseconds from a 10:1 configuration towards an average 210 milliseconds in a 1:1 ratio. Beyond this point, as the ratio of interleaved A-frames is incremented, the average time to detect addition of controls increases linearly at a much steeper gradient, at an average rate of 45 milliseconds with every additional A-frame, and peaking at 604 milliseconds for a configuration of 1:10.

The error bars give an indication of the SRT variability. In any given configuration, there is a chance that the control addition will be detected at a minimum delay of 140 milliseconds if the stimulus takes place just before the occurrence of a P-frame. With high P-frame ratios, the worst-case scenario peaks at 250 milliseconds, for the case where the stimulus takes place just before the occurrence of a single interleaved A-frame. As the ratio of P-frames is decreased, and the number of A-frames is increased, the difference between the best- and worst-case scenario increases and the SRT variability becomes more pronounced, with an SRT for control addition taking anywhere from 140 to 1031 milliseconds for a configuration of 1:10.

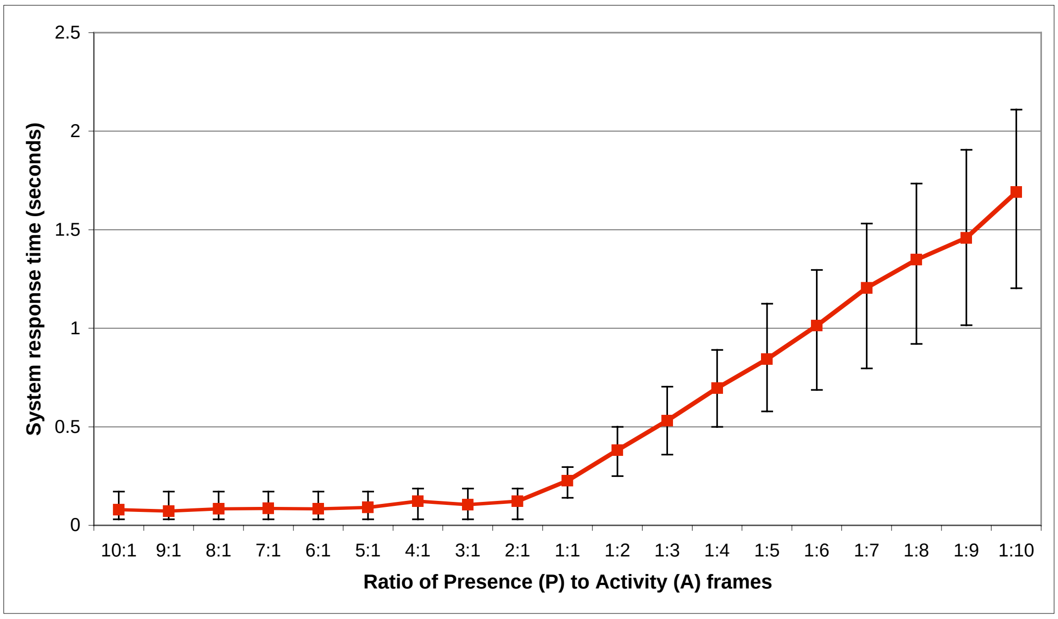


Figure 4.4: Time to detect removal of a control, across varying protocol configurations.

The graph in Figure 4.4. shows a similar pattern for the SRT of control removal. At high P-frame ratio configurations, detection of control removal is on average slightly faster than control addition, with a minimum average delay of 49 milliseconds in a 10:1 configuration. This is due to the fact that after a control is first added, there is a sequence of Read/Write (RW) frames that takes place as the components are configured for operation and before the ControlAddedEvent is raised (cf. §3.5.3 ControlAddedEvent).

When the number of A-frames exceeds the number of P-frames, the average SRT of control removal increases at a higher rate than that of control addition, peaking at an average of 1.69 seconds in a 1:10 configuration. This is due to the fact that a full Presence Detection cycle must be completed before the system can determine that a component is absent from the bus (cf. §3.4.4 ControlRemovedEvent). For the same reason, as the lower bounds of the error bars show, the best-case detection times increase as additional A-frames are introduced. At high A-frame (and low P-frame) configurations the SRT of control removal is longer to detect on average, but has lower overall variability.

4.2.2 SRT of Control Manipulation

A second experiment was performed out to determine the effect of a variable frame-ratio configuration on the SRT of control manipulation. For this experiment, the test rig was set up to periodically modify the input on a single control, present on a network which was otherwise empty of controls. The first variable condition was the frame-ratio, which ranged from 10:1 to 1:10 Presence to Activity frames. A second variable was the type of control: one of the controls used analogue input, and the other digital input.

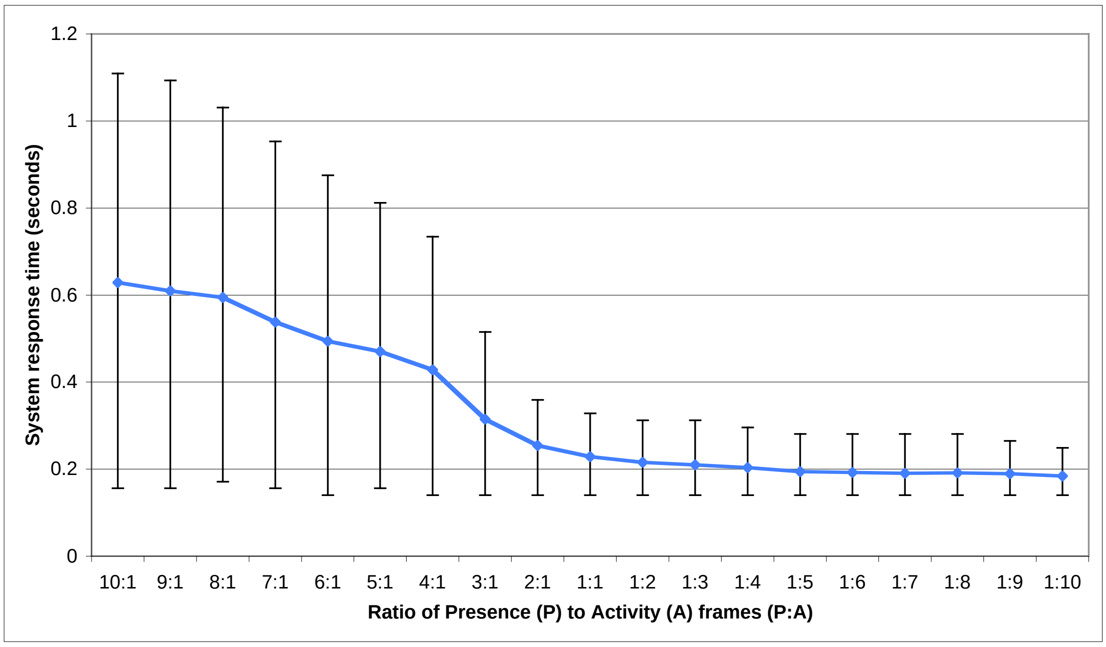


Figure 4.5: Time to detect analog control manipulation, across varying protocol configurations.

As in previous experiments, every condition was tested 500 times, with the times between stimulus delayed by a random amount between 0 and 4 seconds to simulate a user manipulating a control at an arbitrary point in the frame sequence. Figures 4.5 and 4.6 show the results of the tests, with the error bars delimiting the 5th and 95th percentile of the results.

The SRT of analogue controls manipulation performs conversely to control addition/removal over varying frame-ratio configurations. In a 10:1 configuration, the SRT averages 628 milliseconds, with an even distribution between 1100 and 156 milliseconds. As the ratio of A to P frames increases, the SRT improves considerably, and its variability decreases, with a minimum average delay of 184 milliseconds in a 1:10 configuration, with 90 percent of the results distributed between 140 and 250 milliseconds.

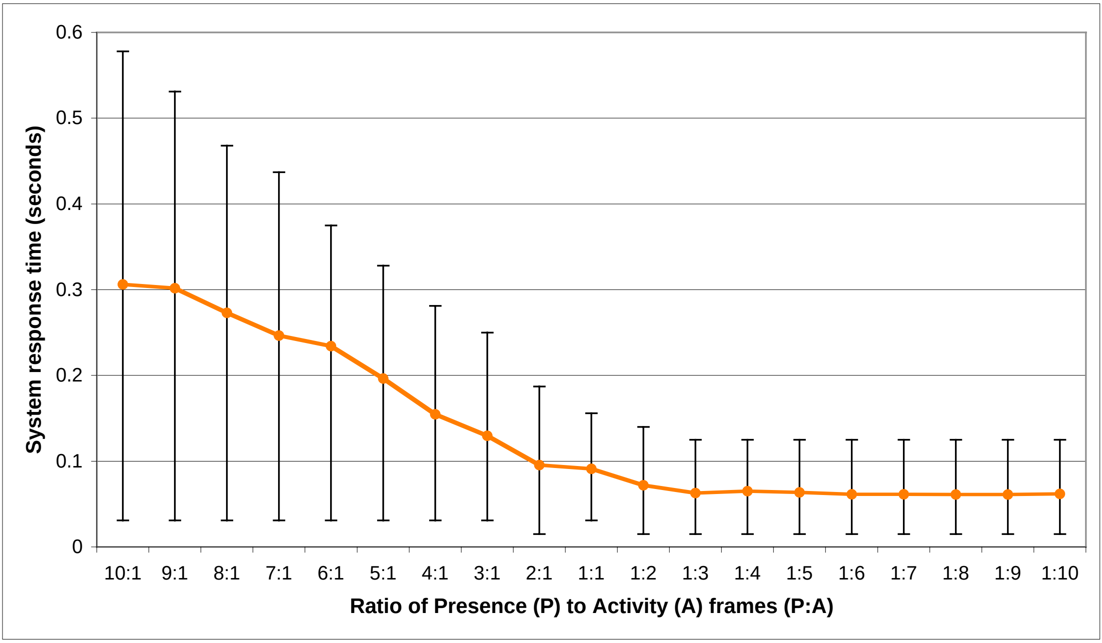


Figure 4.6: Time to detect digital control manipulation, across varying control configurations.

The SRT for digital controls performs similarly to analogue controls, but with an overall faster response times: averaging 306 milliseconds in a 10:1 configuration, and 62 milliseconds in a 1:10 configuration. This is due to the fact that fewer read/write operations are simpler for digital components (cf. §3.5.5 ControlChangedEvent).

4.3 Scalability Performance Results

An important characteristic of any system of networked devices is its ability to scale, i.e. how its performance is affected by increasing quantities of devices on the network. In this section, we present the results of tests carried out to determine the effect of varying numbers of present controls on the bus, and how this variable affects the base system response time.

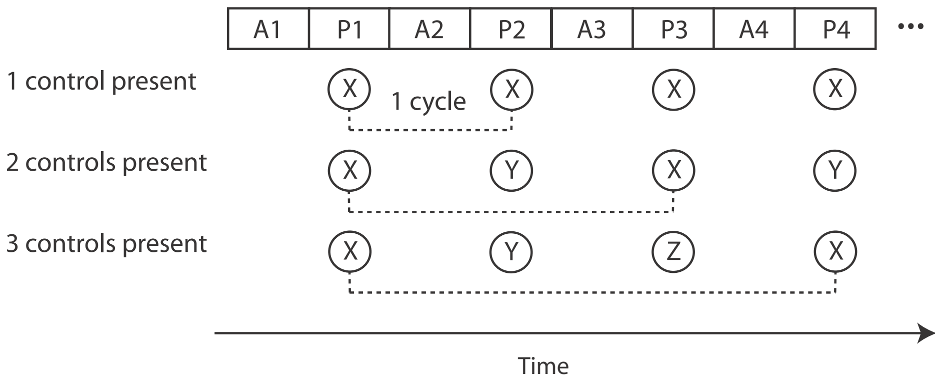


Figure 4.7: Example presence detection cycles with varying numbers of present controls.

Figure 4.7 illustrates a frame sequence with a 1:1 ratio of P-to-A frames, in the condition where only one control (X), two controls (X and Y) or three controls (X, Y and Z) are present on the network bus. As the number of present controls increases, it takes longer to complete a full presence detection cycle (cf. §3.6.2). The activity detection cycle is not directly affected by the number of present controls, but instead by how many controls are simultaneously experiencing activity at any one point in time.

4.3.1 Scaling of Control Addition and Removal

A series of tests were carried out to determine the effect of variable numbers of present controls on the SRT of adding and removing controls. The scheduler was configured with a 10:1 P-to-A frame ratio – optimising it for presence detection – and the test rig was set up to first connect, then disconnect, a single control to a bus. The variable condition was the number of additional present controls, which were added to the bus increments of 5 at a time.

As in previous tests, every condition was sampled 500 times, with the times between stimulus delayed by a random amount between 0 and 4 seconds to simulate a user adding or removing a control at an arbitrary point in the frame sequence. Figures 4.8 and 4.9 show the results of the tests, with the error bars delimiting the 5th and 95th percentile of the samples taken.

As the number of controls increased from 1 to 20, the SRT of control addition increased from its base average value of 171ms to an average 513ms. The bottom 5th percentile error bars show that, consistently across different quantities of present controls, it is still possible for a control addition to be detected at the base rate (if the control happens to be added just before the P-frame where it is detected) but the likelihood of this happening decreases at the list of device that the presence detection cycle has to iterate through gets longer.

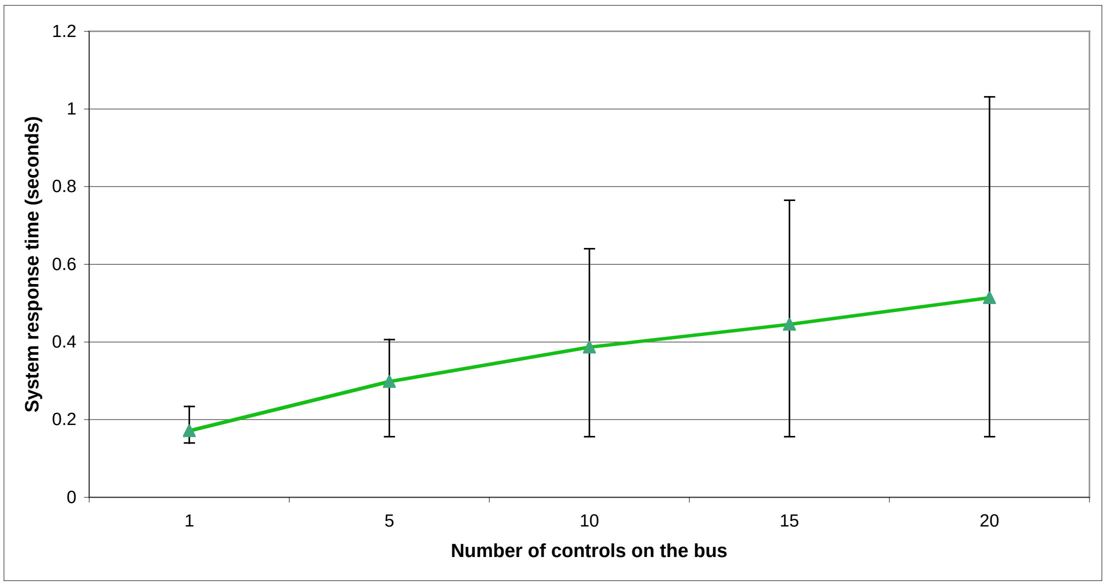


Figure 4.8: Time to detect control addition, given a varying number of controls present on the bus.

The SRT of control removal is affected more drastically with increasing quantities of present controls, taking as long as 2.38 seconds on average to detect a removal with 20 other controls present on the bus. The best-case detection time, as indicated by the lower bounds of the error bards, also increases with the number of controls. This is due to the fact that a full Presence Detection cycle must be completed before the system can detect whether a previously present control is now missing from the list, and has therefore been removed.

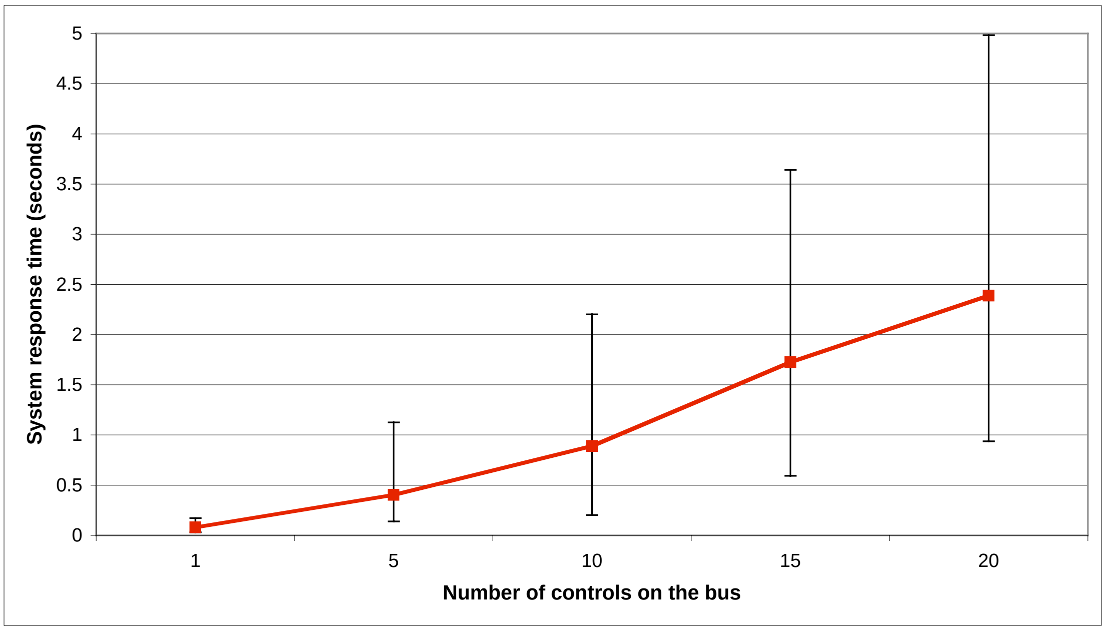


Figure 4.9: Time to detect control removal, given a varying number of controls present on the bus.

4.3.2 Manipulating a control

A test was carried out to determine how the performance of control manipulation detection scales with increasing numbers of controls. The protocol scheduler was configured with a 1:10 P-to-A frame ratio – optimising it for activity detection – and the test rig was set up to periodically manipulate a digital control which was always present on the bus. The variable condition was the number of additional present controls, which were added to the bus increments of 1.

The SRT of control manipulation scales differently from control addition/removal, as shown by the results of Figure 4.10: rather than increasing over time, the SRT remains consistent at the base rate of an average 60 milliseconds, with 90 percent of the results falling between 15 and 125 milliseconds in an even distribution.

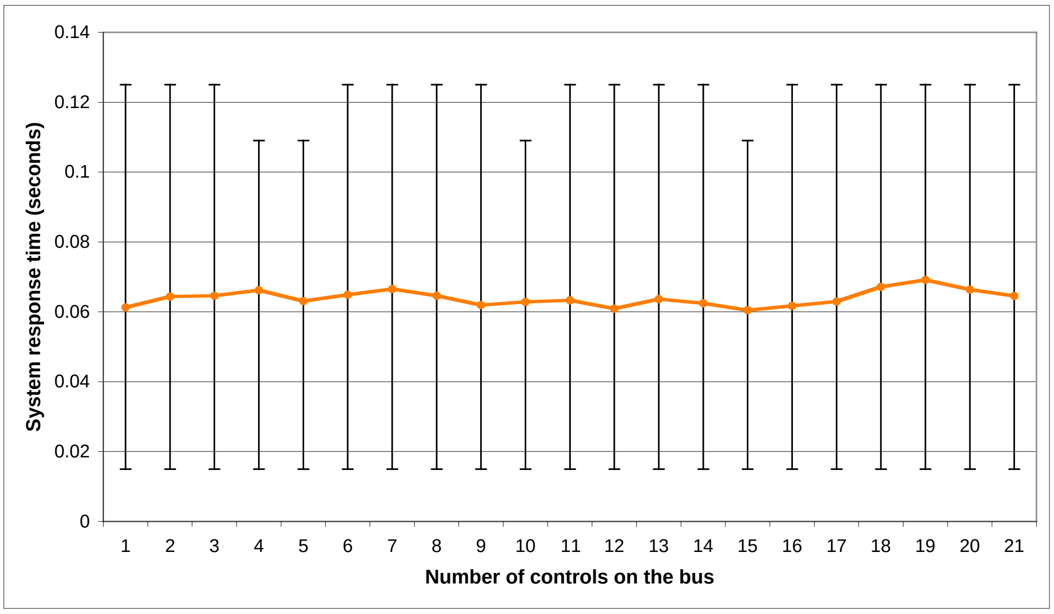


Figure 4.10: Time to detect a single control manipulation, given a varying number of present controls on the bus.

This is explained by the fact that the activity detection cycle is not affected by the number of controls which are present on the bus at any point in time, but rather how many controls are experiencing activity simultaneously. In other words, independently of how many controls are connected, the time to manipulate a single control remains consistent.

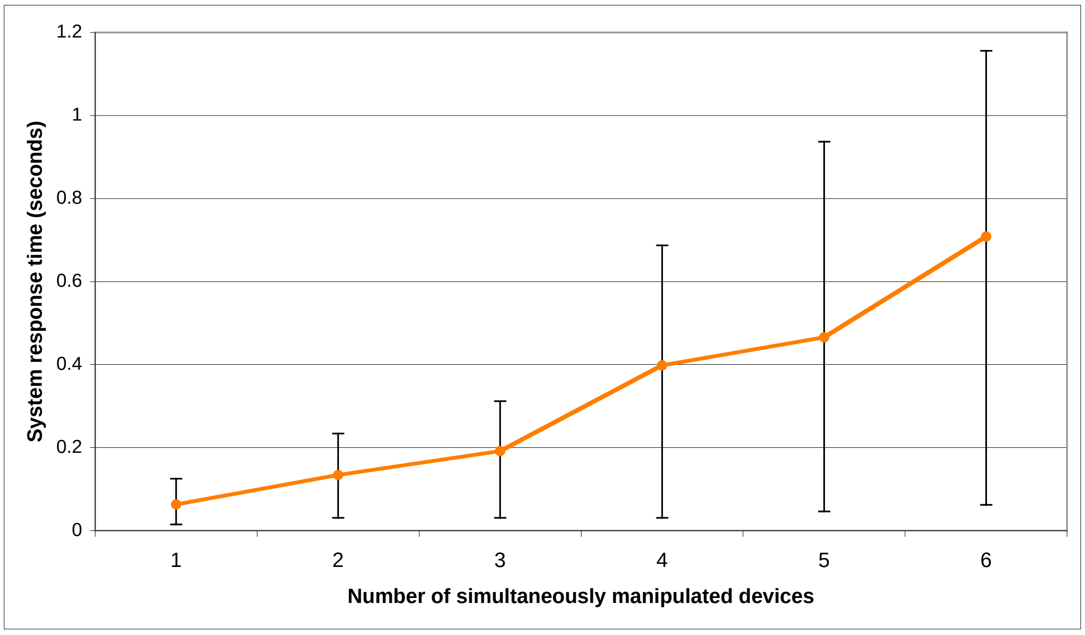


Figure 4.11: Time to detect manipulation of a single control, given an increasing number of simultaneously manipulated controls.

We can, however, expect that manipulating several controls simultaneously will cause the activity detection cycle to be longer, and increase the average time to detect manipulation of any one control. In order to test this behaviour, a final experiment was performed where the scalability of multiple simultaneous manipulations was tested. The experimental setup was the same as in the other conditions presented above. Figure 4.11 shows the results of manipulating from 1 to 6 controls simultaneously. As expected, the time to detect the target control increases with simultaneous manipulations. The SRT to detect manipulation increases by an average of 130 milliseconds with every additional control. For 90 percent of the results, the variability in SRT remains constant for the minimum delay (an average of 36 milliseconds), while the maximum delay increases by an average of 120 milliseconds per every additional control.

4.4 Human Performance

The human perceptual system places a limit to how fast people can perceive a delay between two discrete events, such as a user performing an action on a computer and receiving a system reaction. System response times below a certain latency are imperceptible to users. Naturally, a first question in considering the effect of SRT on user interaction is: at what point does the latency of response time become noticeable to the user?

4.4.1 Limits of Human Perception

Two distinct events happening in a quick enough succession will be perceived by a human as having taken place simultaneously (fusing into a single event), and can give the impression that one caused the other. Card et al reported the results of an experiment carried out by Michotte in 1963 [17] that measured human perception of delay as it relates to this concept of causality. The results of Michotte's experiment indicated that if a delay between two stimuli is longer than 50 milliseconds, the illusion of simultaneity starts to break down and a subject will begin to perceive a "delayed" causality between the two events. For delays greater than 120 msec, the illusion breaks down completely and the two stimuli start to be perceived as independent events. As a result, a latency in the vicinity of 50 msec has sometimes been quoted as a "safe" benchmark below which any latency in SRT is imperceptible to a user. (e.g. in [41]).

A user's experience with system response times is usually more complex than simply perceiving a delay between two independent stimuli. When interacting with a system the user not only perceives events, but also generates them by acting on the system and providing input. Any perception of latency occurs if the gap between an action (which must be decided upon and enacted) and a reaction (which must be perceived and processed) exceeds a certain threshold. The perception of SRT therefore involves not only the human perceptual system, but its motor and cognitive systems as well. Card et al. apply their Human Processor Model [17] to the task of estimating the time between a symbol appearing on the screen and a user pressing the spacebar, and find that reaction times for this simple task is in the range of 100 to 400 msec.

4.4.2 Measuring Human Perception of Delay

In order to gather further confirmation as to the human perception of SRT latency, we carried out an experiment to measure a subject's ability to perceive a SRT delay when performing a simplified interaction task: pressing a button and having a light turn on in response.

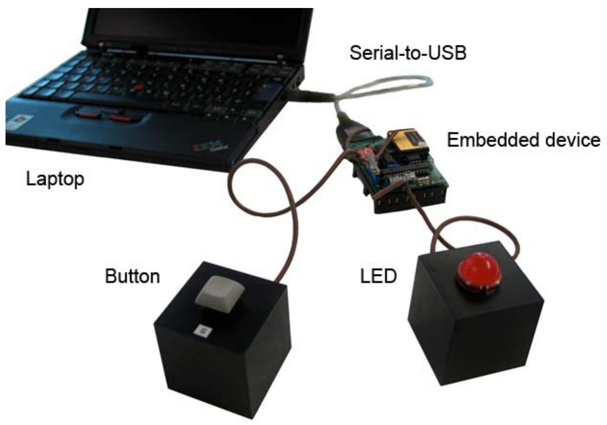


Figure 4.12: Test apparatus to measure human perception of SRT

The experimental setup is shown in Figure 4.12: A keyboard-style button and a wide-angle, high-brightness LED are wired up to an embedded device, which is equipped with a serial interface to a laptop PC. The embedded device includes a PIC microcontroller that is programmed to turn the LED when it registers a button press. The microcontroller can be configured (from the PC, via the serial interface) to introduce a delay between the detection of the button press and the turning on the LED. This test rig was developed in order to have accurate control over the timed delays between input and output, which would have been difficult to guarantee if using a general-purpose PC with a non-real-time operating system.

4.4.3 Experimental Procedure and Results

The experiment consisted of asking a participant to arrange the button and LED on a table in such a way that the button could easily be pressed with their dominant hand while maintaining a clear view of the LED. It was then explained that whenever the button was pressed, the LED would always turn on, and that in some occasions the LED would turn on immediately following the button press, and sometimes there would be a gap between the button press and the LED turning on. Participants were then told that they would be asked to press the button a number of times and, while doing so should be looking closely at the LED to determine if they perceived it to turn immediate or after a gap of any duration.

Participants were given 10 practice conditions with randomised delays between 10 and 1000 msec to get used to the test procedure. They were then tested and their responses ("gap" or "immediate") recorded for a series of specific time-intervals. We expected that the perception of delay would start to become apparent in the range of 50 to 200 milliseconds. We therefore tested in increments of 20 msec between 10 and 210 msec (at 10, 30, 50, 70, 90, 110, 130, 150, 170, 190 and 210 msec). We also measured some longer delays at higher intervals (at 250, 300, 400 and 1000 msec) where we expected the perception of delay would be more clear-cut.

It was anticipated that there would be some short-term learning effect between conditions (e.g. a delay of 300 msec after a 1000 msec delay might be more likely to be perceived as an "immediate" reaction, and a 300 msec delay after a 10 msec delay more likely to be perceived as a "gap"). In order to minimise this effect, each participant was tested twice for every condition, and the order in which each condition was randomised between subjects.

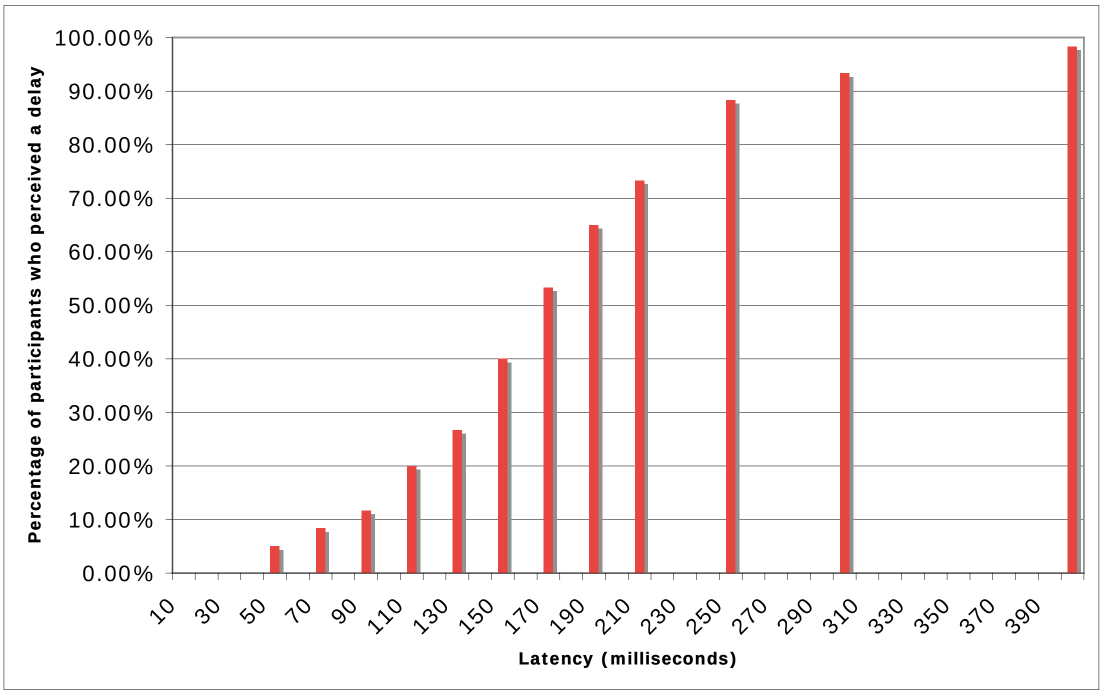


Figure 4.13: Experimental results.

In total, 30 participants took part in the experiment (12 women, 18 men in the age range of 19 to 45), and the collected results are presented in the graph in Figure 4.13. No participant perceived a delay below a SRT of 50 msec. About 50% of participants perceived a delayed response with a SRT of 170 msec, and the majority of participants (98%) perceived a delay with a SRT of 400 msec. Although the data is noisy due to the limited sample size, we believe that these results give a valid indication of the limits of human perception of SRT delay. This is validated by the fact that results are in line with Card et al's Human Processor Model for reaction times in a simple interaction task, and Michotte's findings into perception of multiple event fusion.

4.5 Effect of SRT Delays and Variability

While we have so far discussed how SRT might be perceptible to the user, we have not yet touched on the implications of long or variable response times can have on the process of human computer interaction. The actual effect of perceptible SRTs is a complex issue, and several studies have examined the relationship between response times and the quality of user interaction ([12], [5], [39], [35]). The key factors that have been explored include, firstly: at what point does a delay in SRT become an issue for the user, affecting their satisfaction or ability to perform? And, secondly: what is the impact of variability of response times on the user? Investigations in this area measure the loss of productivity, increase in numbers of errors and perceived user (dis)satisfaction that can arise when the response times and SRT variability exceed certain thresholds.

The results of different studies are varied, complementary and sometimes contradictory, reflecting the complexity of the issue. For example, in exploring the effect of overly long response times and excessive variability of response time, Butler found that very slow computer response times did not appear to degrade user performance by any large amount (variability in response times had a slightly stronger effect on performance) [12]. But while the effect of long SRTs might not directly affect performance, research into the impact of system response time on user anxiety Guynes found a strong positive relationship between increased system response time and user frustration when interacting with a system [39].

Other research has proposed SRT guidelines for designers of interactive systems. Goodman et al. [35] quote Foley & Wallance [30] who propose the idea of different levels of appropriate SRTs, which are dependent on the nature of the task at hand (lexical, syntactic or semantic), with proposed values of 50 msec, 1 to 4 sec, and as much as 10 seconds respectively. Schneiderman proposes that acceptable variations on SRT can be generally regarded as being in the range of plus or minus 50 percent of the mean. He also reports the findings of several studies into SRT, and summarises the three key factors influencing response time expectation as follows [5]:

1. Previous experiences are critical in shaping expectations.
2. There is enormous variation in response time expectations among individuals and across tasks.
3. People are highly adaptive. Although they may be able to accommodate long and variable delays, their performance and satisfaction are likely to suffer.

These conclusions are echoed in one form or another in most research related to SRT: lower SRT and variability in SRT are probably better, but the maximum acceptable levels are highly dependent on the experience and expectations of the user, as well as the task at hand. In short, the suitability of the response times of an interactive system are only fixed in the context of a concrete application.

4.6 Discussion

We can summarise the results of experiments into both system and human performance in Figure 4.14, which shows the SRT across varying configurations against the backdrop of the human ability to perceive SRT delays.

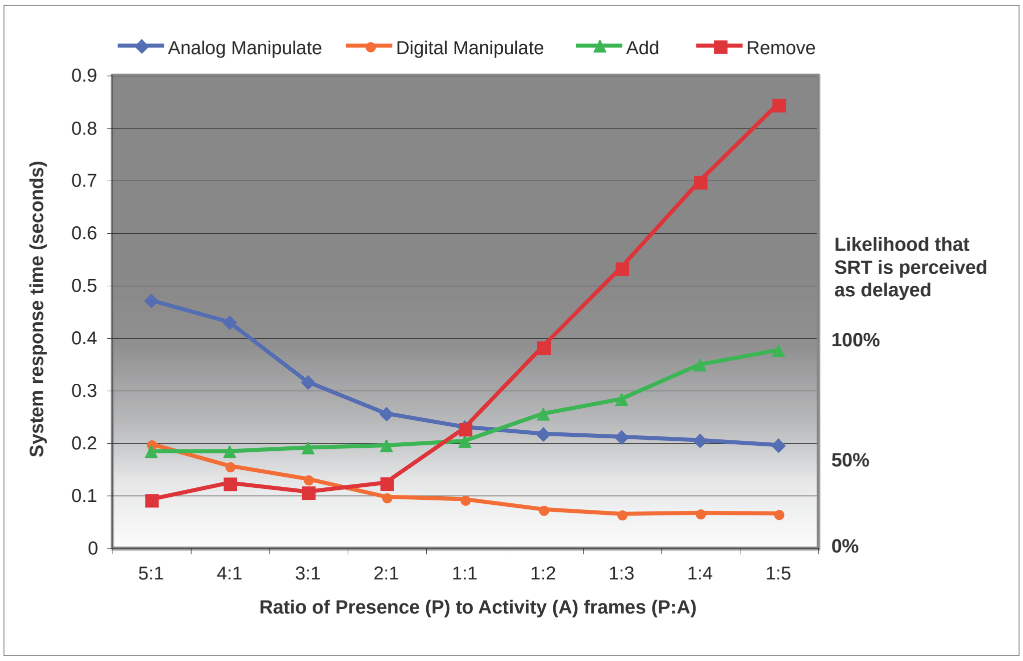


Figure 4.14: Comparison of base SRT for control presence and manipulation.

To sum up the key features of the results: the system can be optimised to either detect addition/removal of controls, or control manipulation; the SRT for control addition/removal increases both in latency and variability with increasing numbers of controls on the bus, but manipulation of a single control scales consistently and with a low variability; and, the overall SRTs are within the bounds of human ability to perceive delays with a consistently low variability, particularly when the system is optimised for this case. How this affects the user and their perceived quality of interaction will be highly dependent on the task at hand, the expectation of the user and the design of the application.

There does exist a clear trade-off between the two interaction processes of addition/removal and manipulation of controls. This may be an important fact to communicate to interaction designers working with the technology. By exposing this characteristic, application designers are able to tailor the SRT performance for the system for the particular task at hand. For example, an application might optimise the system for detection of control presence during a configuration phase – allowing the user to quickly change and explore different control configurations – and later optimise it for detection of manipulation events during regular use. If detection of both addition/removal and manipulation of controls plays an important role in the interface, a suitable balance may be found in a 1:1 Presence to Activity configuration. It is also conceivable to implement an adaptive optimisation strategy, which keeps track of usage patterns and automatically tailors the system for the style of interaction exhibited by a particular user of an application.

The particular characterisation of system response time presented in this chapter is tightly bound with some system design choices, particularly the use of 1-Wire communications bus which only supports relatively low communication rates and very limited bandwidth. However, the issue of delayed and fluctuating SRTs is likely to be present in any design for networked control, and technical solutions will trade off with other aspects of the technology. For example, it would be possible to use alternative channel of communication to transmit control data (e.g. a wireless channel, such as Bluetooth). While this would address the problem of providing a consistently fast SRT, it could very likely result in a trade-off with other practical aspects of the design, such as increase in device complexity, higher power consumption or an increase in per device cost. The design for an architecture of networked controls balances several issues and presents a solution which is practical beyond a proof-of-concept: the trade-off is a limitation in its response time. By carefully characterising and understanding this key issue, we provide a way for application designers and users to make appropriate use of the technology.

Chapter 5

Physical Design

The premise of this thesis is to achieve flexibility in the physical interface hardware, and part of this effort entails the development of a design that supports and encourages physical adaptation, construction and arrangement of interface devices. This chapter presents a physical interface design, called VoodooIO, that allows users to put compose and adapt interfaces out of individual controls devices. From a system standpoint, the design allows controls to physically connect and disconnect from a network, based on the system of networked controls introduced in Chapter 3. From a user perspective, the design supports a high degree of flexibility and ease of use in distributing and arranging controls within a physical space.

5.1 Key Concepts

The physical design of the VoodooIO interface can be manually shaped, tailored and physically modified to a user's requirements – rather than as a computer interface device with a rigid shape, a predetermined selection of controls or fixed layout. The main components of the design are:

- The *substrate* is a flexible material designed to support devices in two ways: providing a surface area for control arrangement, and providing a physical carrier for power and data transmission. For this purpose it is composed of conductive layers separated by isolative material. The design involves two conductive layers to create the areal equivalent of a two-wire cable, suitable for the transmission of 1-Wire network signals (c.f. §3.5). The material can be cut to shape and used to augment existing surfaces, objects or areas of the architecture.

- The *control set* is made up of a selection of physical input and output controls. Internally, they are implemented as network nodes (c.f. §3.6) and are equipped with special pin-like connectors to allow them to fasten onto the substrate material and connect to the internal conductive layers.

Figure 5.1 depicts an example deployment of the substrate material in an office environment: a sector of the wall has been covered in the material, and further sections have been affixed to the side of the monitor bezel, and chair armrests. The façade of the telephone unit has been extended with an additional section, as has the keyboard - notice how the material has been shaped to fit its curved edge, extending the keyboard and integrating into the desk arrangement. Individual sections of deployed substrate are interconnected, allowing controls to be located and arranged anywhere within the highlighted areas.

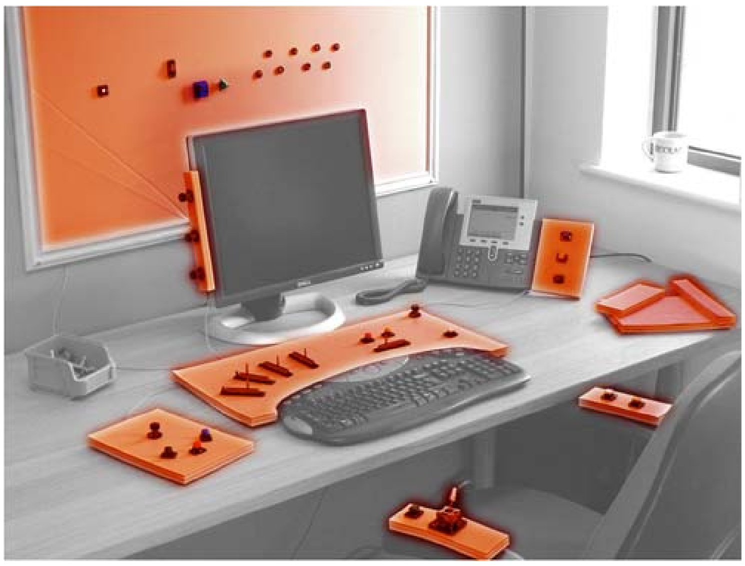


Figure 5.1: An example deployment of the substrate material and control set.

The substrate allows the user interface to extend beyond the desktop, making use of the architectural elements that surround it and existing furniture and objects within it. The key benefit of the design is the support for free placement and arrangement of controls on surface areas augmented with the substrate material.

5.2 Physical Attachment = Digital Connectivity

The concept of surface-based networking has been explored in a number of research projects ([78], [55], [57], [85]) where the motivation is to combine the benefits of wired and contact-based networks – such as the ability to supply power to connected nodes – with the flexibility associated with wireless networks. Traditionally, wired networks use cables and a plug-and-socket connection mechanism to connect nodes to the network bus. In surface networking (illustrated in Figure 5.1) the surface becomes the medium for transmission of network signals: devices are connected to the network by simply by placing or attaching on the active surface.

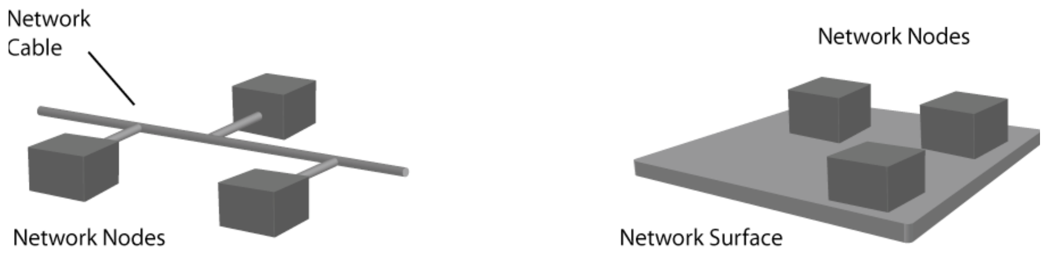


Figure 5.2: A traditional cabled network (left) vs. a surface-based network medium (right).

The VoodooIO design builds on a concept for connecting devices to a conductive surface by means of pin connectors, originally conceived in the Pushpin Computing project [57] (c.f. Figure 5.4). The connector mechanism was later appropriated in the Pin&Play project [85], which explored the use of a layered conductive surface to act as a planar network medium (c.f . Figure 5.3).

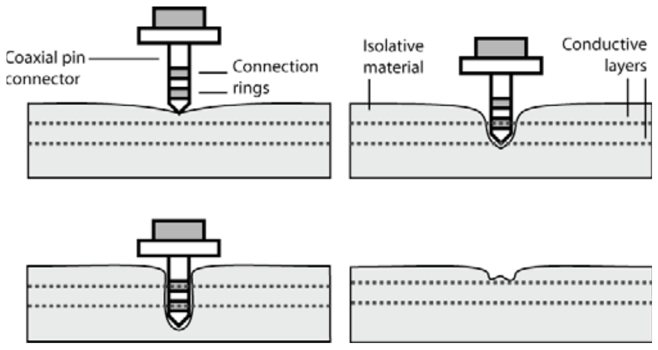


Figure 5.3: Pin&Play uses a pin-like connector to pierce through the surface and make contact with the conductive layers embedded within.

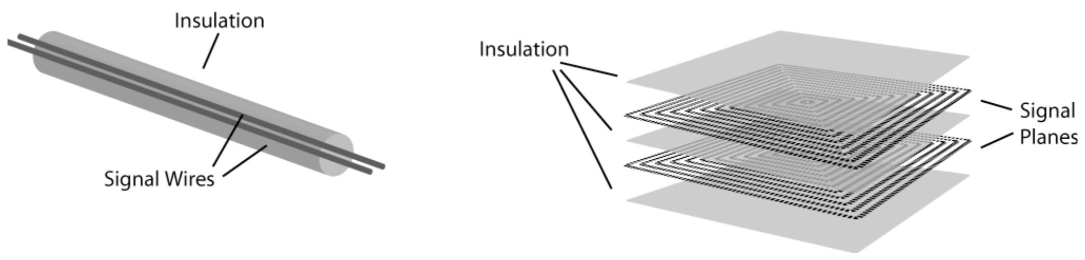


Figure 5.4: The Pin&Play technique uses a network medium made of conductive layers embedded within insulating layers (right) – a planar equivalent to the wires embedded inside the insulated cables traditionally used in networking (left).

The VoodooIO design builds on the innovations of the Pin&Play project, and develops them as physical form factor to networked control concept described in Chapter 3. The rest of this chapter documents the development of a novel material for planar networking – called the substrate material, described in Section 5.3. Control devices are equipped with coaxial pin connectors that can attach and connect to the substrate (c.f. Section 5.4). Using this design as a blueprint, a number of concrete different physical interface controls are developed (c.f. Section 5.5). The result is a fully-functional design for physically flexible user interfaces, where the act of physically attaching a control to a substrate has the effect of connecting it logically to the interface.

5.3 Substrate Material

The substrate is a novel composite material: a laminate of flexible rubberised sheets with embedded conductive layers that allows the propagation of 1-Wire network signals across an area (c.f. Figure 5.5). The name stems from its enabling role as an underlying layer on which controls can be placed, doubling as a physical construction material and physical medium for network signals.

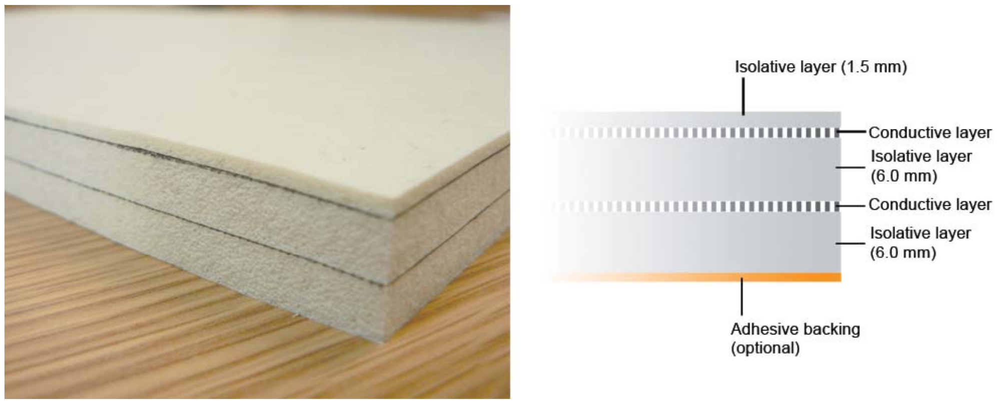


Figure 5.5: Substrate composition: conductive layers are embedded within an isolative material.

5.3.1 Isolative Layers

The bulk of the substrate is made up of silicone foam rubber sheets, which act as insulation between the conductive layers. Different materials were evaluated for their suitability – including polyethylene, neoprene and ethylene propylene rubber sheets – and, while they are all comparably suitable for providing isolation between the conductive layers, the silicone foam rubber proved to be the most resilient to penetration by the sharp pin-connectors (c.f. Section 5.4 Substrate Material Properties). In addition, silicone foam rubber provides an aesthetically pleasing look and feel to the surface of the substrate.

Figure 5.6: Silicone foam rubber sheets of varying thickness.

The sheets used in the substrate are manufactured by Silicone Engineering Ltd. (UK), who supply the material in various thickness (1.5 mm, 3.0 mm and 6.0 mm) and in a variety of colours. They also supply the sheets with or without adhesive backing with a peel-off paper covering.

As illustrated in Figure 5.5, the design of the substrate uses a 6.0 mm silicone foam rubber sheet at its base (with optional adhesive-backing to allow the substrate to be affixed to a surface). A second 6.0 mm layer provides an isolation between the two conductive layers, while a third – 1.5 mm thick – layer acts as the top surface and protects the conductive layer from direct touch.

5.3.2 Conductive Layers

The conductive layers used in the substrate are made up of a conductive fabric manufactured by SwissTulle Plc (UK). The fabric is produced as a bobbinet-style tulle, that is weaved into a hexagonal mesh which is regular and clearly defined (c.f. Figure 5.7).

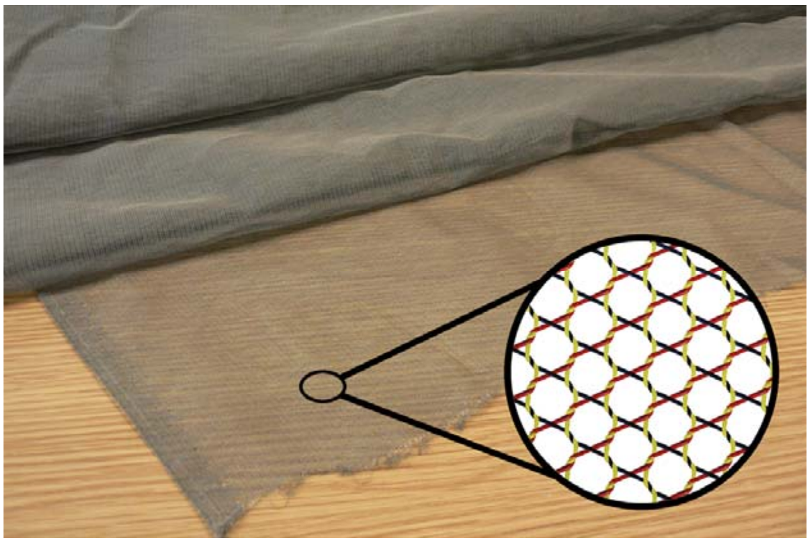


Figure 5.7: The conductive layers are made from a conductive fabric.

The SwissTulle product used in the substrate manufacture is the 166NS46 fabric, which is weaved with silver diagonal (weft) yarns around a nylon vertical (warp) yarn to make it electrically conductive across its surface (c.f. Figure 5.7). The extremely low resistance of the fabric (< 3 ohm per meter) make it a good medium for the transmission of network signals. The bobbinet weave, coupled with the elasticity of the nylon yarns, allow the fabric to be highly resistant to deformation and tearing.

The fabric performs much better than any other conductive materials that were tested for the purpose. Solid conductive sheets – such as tinfoil – provide an even lower electrical resistance, but they suffer severe deterioration when pierced by the pin connectors, which leave permanent and irreparable holes in the sheet. Other conductive fabrics were also found to be unsuitable, being either too tight in their weave (leading to tearing or difficulty in piercing), too loose (meaning that the pin connectors would not make contact) or providing too much electrical resistance to be use for signal transmission .

5.3.3 Bonding Process

Bonding the silicone foam rubber sheets and conductive fabric into a single laminate proved to be a materials challenge in itself. Initial prototypes tested a variety of glues to bond the materials together, but glue residues between the layers would mean that, once inserted, the pin connectors would be covered in a fine coating of glue and become unable to make contact with the conductive layers. A second approach attempted to stitch the layers together at the edges, but this led to lack of stability in the material and poor connectivity performance. A third approach involved suspending the conductive layers within a frame, and pouring liquid the silicone in liquid form to encase the fabric in a single piece of silicone foam rubber. Although promising, this approach was time consuming and delicate, as the curing process had to be carefully managed to ensure that the distance between the conductive layers was evenly preserved throughout the substrate.

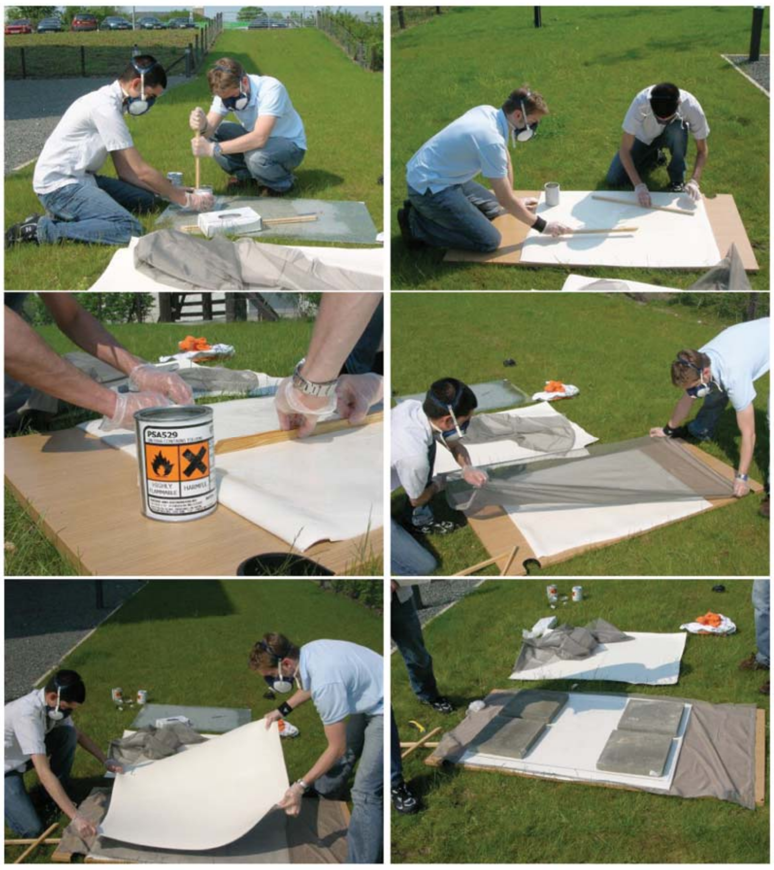


Figure 5.8: Substrate bonding process: layers of silicone foam rubber are alternated with layers of the conductive fabric, and bonded together using a silicone adhesive.

The solution to the bonding problem was found with the help of a company specialising in silicone adhesives - Techsil Ltd. (UK), who produce a liquid, pressure-based silicone adhesive (PSA 529) that can be used with a catalyst (SRC 18) to bond the silicone sheets at room temperature, leaving no residue between the layers. The bonding process is illustrated in Figure 5.8: a 6.0mm-thick layer of silicone sheeting is laid out, and its top surface coated with the adhesive; a conductive sheet is stretched over the sheet, and sandwiched between a second 6.0mm silicone sheet. The porosity of the fabric allows the adhesive to flow through it and affix to the second silicone sheet. The process is then repeated with a second conductive sheet, and the final 1.5mm silicone sheet. Weights are placed on top of the completed laminate to provide even pressure across its surface, which is then left to cure for 24 hours at room temperature.

5.4 Substrate Material Properties

The substrate composition and manufacturing process result in a material with some novel properties as a network medium. For a start, the material is highly flexible, it can be bent and cut into custom shapes and forms (c.f. Figure 2.9).

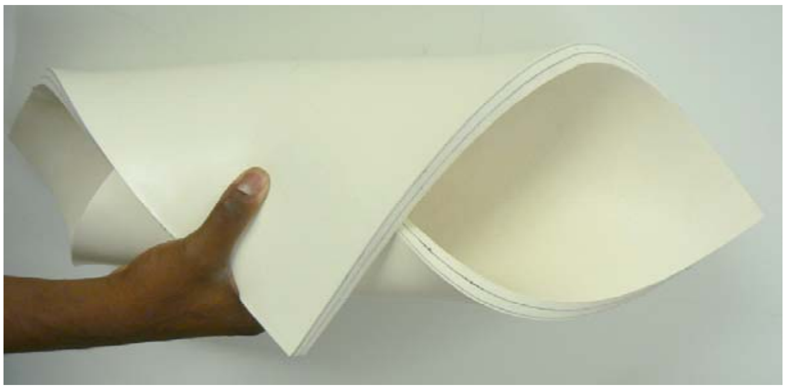


Figure 5.9: The substrate material is highly flexible.

Due to the combined properties of the silicone foam rubber and bobbinet-weave of the conductive sheets, the substrate is also largely self-healing and resistant to punctures, much like the cork sheets used in notice boards. The left column of Figure 5.10 shows how, after being pierced by a sharp pin which is then removed, the surface of the substrate closes over leaving almost no visible mark (a circular area has been marked with increased contrast to highlight the mark which is hard to see with the naked eye).

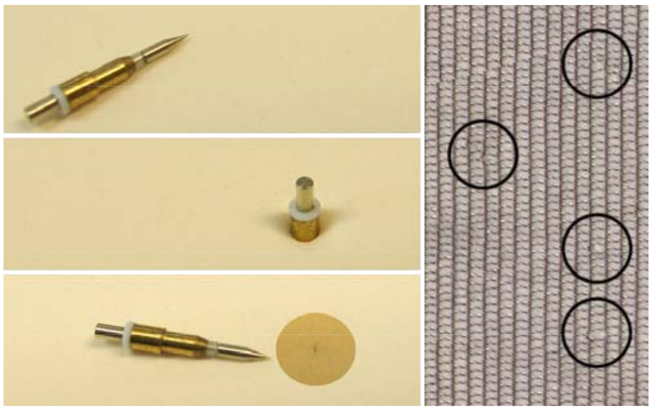


Figure 5.10: The substrate is self-healing and resistant to punctures by the pin connectors.

The right-hand image of Figure 5.10 shows a highly magnified scan of the conductive fabric after being repeatedly pierced with a pin connector. Although the girth of the pin is several times thicker than the pitch of the weave, the fabric does not snap, tear or even deform excessively. Instead, the weave allows the pin to pass through it – the threads run and expand the pitch, temporarily accommodating the pin and providing a secure electrical connection to it. Removing the pin leaves only minimal deformation of the weave. While exhaustive material destruction tests have not been performed on the substrate, experience has shown that the material can be pierced repeatedly in the same location without it presenting any serious signs of degradation.

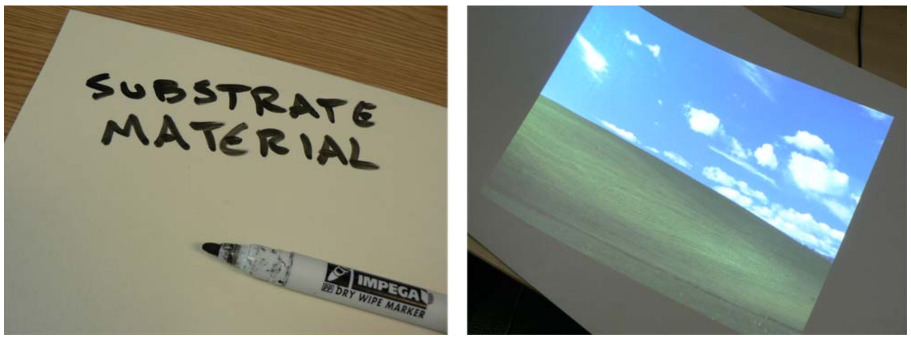


Figure 5.11: The white surface of the substrate serves as a writing and projection surface.

While the white surface of the substrate was chosen partly for aesthetic reasons, it can also serve practical functions, as it doubles as a temporary writing surface (using dry-white markers) or a projection screen (c.f. Figure 5.11). An optional adhesive-backing allows the substrate to be permanently affixed to surfaces walls, objects and furniture (c.f. Figure 5.12).

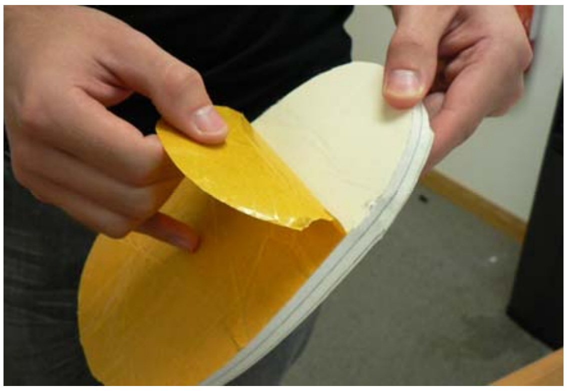


Figure 5.12: The adhesive backing can be used to stick areas of subsrate to existing surfaces.

5.4.1 Electrical Considerations

Due to manufacturing constraints, the largest area of substrate that has been produced is no bigger than two meters square. However, it is conceivable that larger areas can be produced and still be effective as medium for network signals. One limiting factor is the resistance of the medium, determined by the conductivity of the conductive fabrics. A second factor that may come into play (especially with large substrate areas) is the capacitance of the material. The substrate is, in effect, a large capacitor made up of two large parallel, conductive plates. Capacitance can be calculated by the following formula:

$$C = (E0 \times Er \times A) / d$$

Where C is the capacitance, E0 is a universal constant, Er is the dielectric constant of the material between the conductive plates (the silicone foam rubber, in this case), A is the area of the plates and d the distance between the plates.

For a two-square meter area of substrate the capacitance is negligible: the dielectric constant for the silicone foam rubber is relatively low (2.9, with 0 being a vacuum, and water having a value of about 80) and the distance between the plates relatively high (6.0 mm). However, for larger substrate areas (a high A value in the formula above) it is conceivable that the capacitance will have an effect on its slew-rate (how quickly it is possible to charge or discharge the medium from a "high" to "low" state, or vice-versa).Establishing the limitations of the maximum size of a substrate area that can act as a network medium is outside the scope of this work, and is left open to future research. As it stands, the design constitutes no limitations for the range of interactive applications that motivated the design.

5.5 Coaxial Pin Connectors

The connection of controls to the substrate is made using pin connectors. The connector pins are a custom component, designed and manufactured to specification and especially for this purpose by the engineering firm Cambion Ltd (UK).

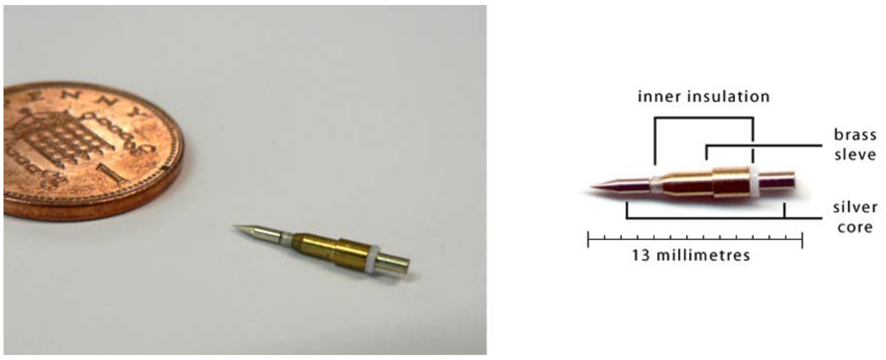


Figure 5.13: Coaxial pin connector design.

The pin connectors are small, not much bigger than a normal pushpin, and have a coaxial design to facilitate independent contact to the two conductive layers of the substrate (cf. Figure 5.13): an outer brass sleeve is separated from an silver core by an inner section plastic tubing.

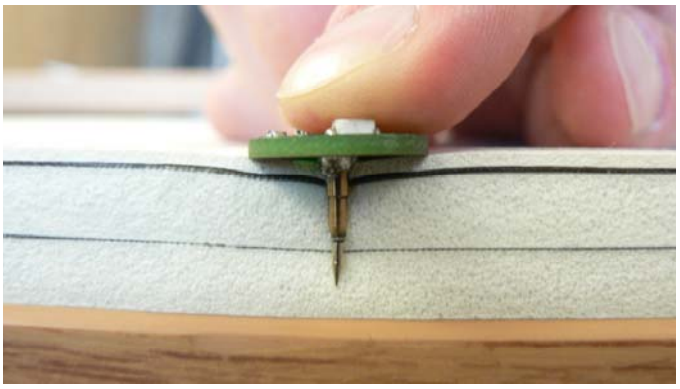


Figure 5.14: The outer and inner sections of the pin connector make independent contact with the top and bottom conductive layers of the substrate, respectively.

The blunt end of the pin is designed to be soldered as a through-hole component to a printed circuit board (PCB). The sharp end of the pin is tapered, facilitating its entry through the various substrate layers. When the pin is fully depressed into the substrate, the outer and inner sections make independent contact with the top and bottom conductive layers, providing an electrical pathway between these and the PCB (c.f. Figure 5.14).

5.6 Control Set

Using the blueprint for controls as network nodes described in Section 3.6, and given the substrate material as a network medium, it possible to design a set of user controls equipped with coaxial pin connectors to form the basic building blocks of a physically user interface.

Everyday appliances, computer input devices and other equipment intended for user interaction are make use of a variety of physical controls in their interface (c.g. Figure 5.15). By deconstructing these interfaces into individual controls components, it is possible to note that a limited amount of control types are reused across designs. Albeit in many different form factors, and with some variation as to physical design and input capabilities, most physical interfaces are composed of simple transducers such as buttons, switches, sliders, knobs, dials and joysticks are the basic building blocks of physical interfaces.

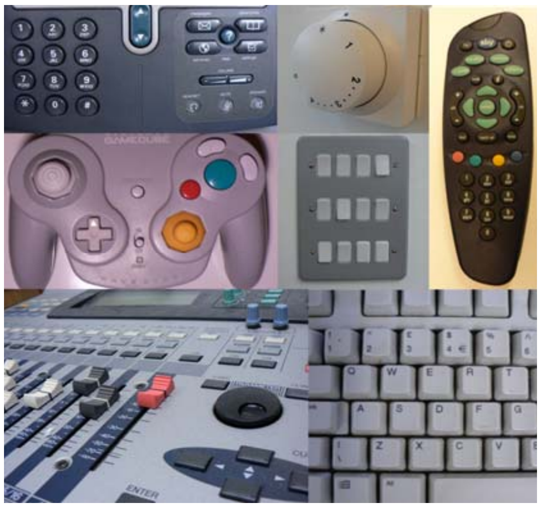
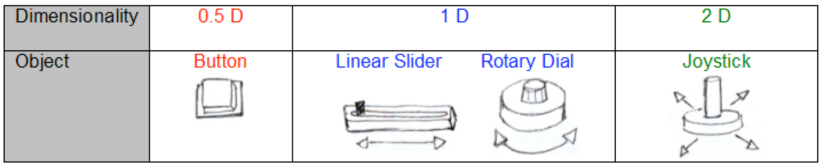


Figure 5.15: Examples of everyday interface devices and constituent physical controls.

There have been a number of attempts to capture and categorise the design space of input devices and controls, most notably in Card et. al's morphological analysis of the design space [16], and Buxton in [13]. For the sake of simplicity, we can broadly categorise these atomic types of input controls by their dimensionally of input, or how many axis of input they provide (c.f. Figure 5.16).



Dimensionality	0.5 D	1 D		2 D
Object	Button	Linear Slider	Rotary Dial	Joystick

Figure 5.16: Controls categorised by their dimensionality of input.

Controls such as buttons or switches provide binary input, while analog controls such as sliders or dials allow users to express input along a single axis. Joysticks combine two analogue axis in a single control to provide two-dimensional input. Although higher dimensionalities are possible – such as with the Flock of Birds device, which provides 6 axis of simultaneous input – their use is rare in everyday physical interfaces. More likely, these types of basic controls are combined to provide some form of compound control with a higher dimensionality of input (e.g. the arrow keys on the keyboard, which provide two-dimensional input), as illustrated in Figure 5.17.

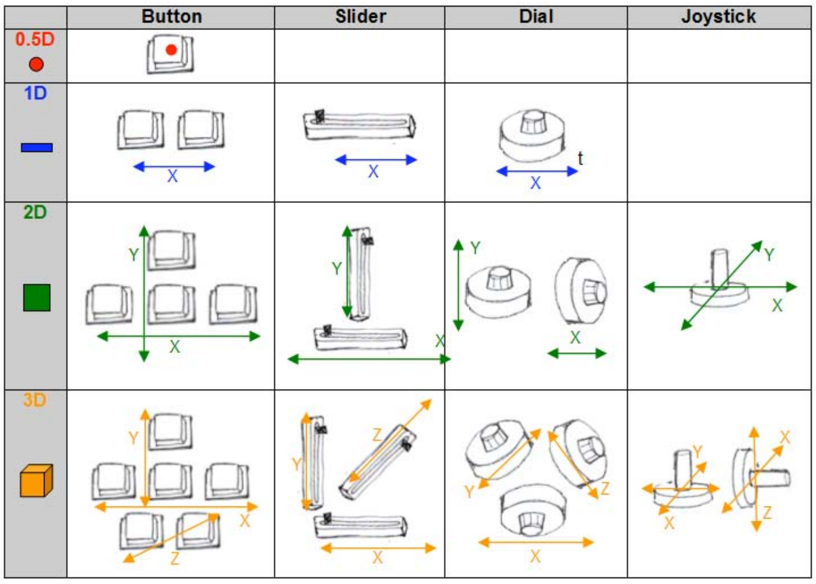


Figure 5.17: Basic control types can be combined into compound controls.

There are other important variations - chiefly in the mechanical behaviour of controls - which determines their suitability for a certain task. One notable feature is the distinction between "put-and-stay" control types, which retain their input state after being manipualted, and "self-centering" or "momentary" controls, which are usually spring-loaded to return to a neutral state after being handled. Other characteristics include resolution, range and discretised-versus-continuous input.

5.6.1 A Selection of Input Controls

Some of the control types that have been implemented for the VoodooIO control set are shown in Figure 5.19. Each is control is a network node that follows the circuit blueprint laid out in Section 3.4. All controls are equipped with one or more coaxial connector pins, wired up in parallel to provide a robust connection to the substrate (both mechanically and electrically).

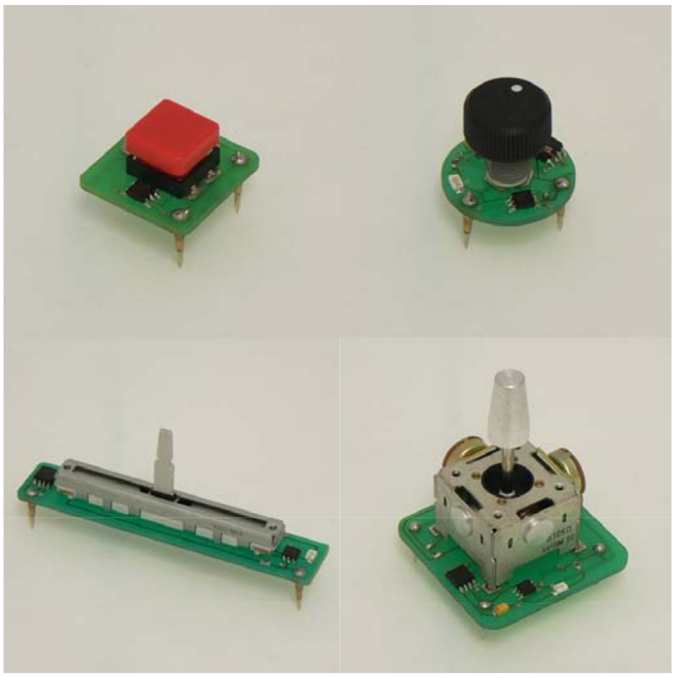


Figure 5.18: Some of the implemented control types included in the control set: button (top left), dial (top right), slider (bottom left) and joystick (bottom right).

In addition, each includes a digital or analogue input transducer (e.g. a button, a rotary potentiometer "dial", a linear potentiometer "slider," or a two-axis joystick). All of these controls also include a small output LED – referred to as the control's "beacon" – which can be used to provide localised feedback on every control.

The basic control set cover the basic dimensionalities of input discussed above, and the control set includes some additional variations of each type, such as a "put-and-stay" and "self-centering" version of the joystick and button (i.e. a latching switch and a push-button) as well as a "stepped" and a "smooth" version of the slider, and a version of the dial with continuous rotation versus one with a minimum and maximum boundary. By providing a generic blueprint for control design, it is expected that the control set will grow over time as new control types and variations are included.

5.6.2 Output Controls

Some basic output controls have been implemented for the VoodooIO control set: a single digit numeric display and a RGB LED, capable of displaying 256 shades of colour (c.f. Figure 5.20).

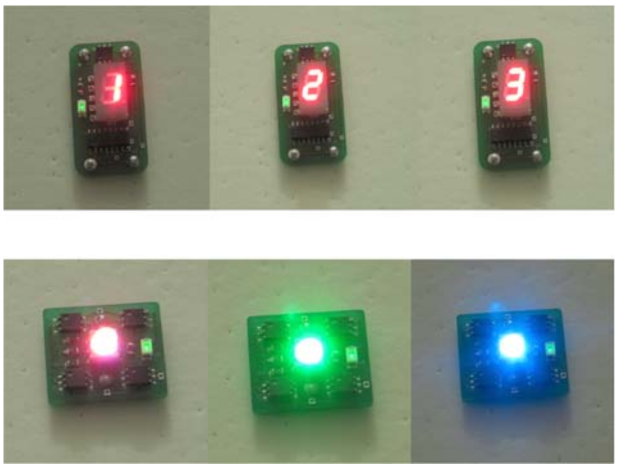


Figure 5.19: Some output controls: a numeric display (top) and a RGB LED (bottom).

More complex output devices – such as video screens or audio speakers – are not readily supported due to the limited bandwidth of the network bus and low-voltage power supply. For applications where complex output is required, VoodooIO can be interfaced with traditional displays and audio systems (c.f. Chapter 6, Application Support Tools). The intention is not to replace all existing interface devices with VoodooIO components, but rather to provide a flexible control platform that can be integrated with existing devices and complement their use.

5.6.3 Substrate Connector

The substrate connector is not a control per se, although it is similar to the above controls in design. Rather than being equipped with an input/output transducer, the connector includes a RJ-11 socket – similar to the standard telephone sockets – connected directly to the coaxial pin connectors mounted underneath (c.f. Figure 5.21). The connector is used as a means to attach a piece of substrate to a network adapter, acting as a bridge between the coaxial-pin and RJ-11 connector mechanisms. Like a control, it includes a primary memory component – effectively providing the substrate area it is connected to with its own unique identification on the system.

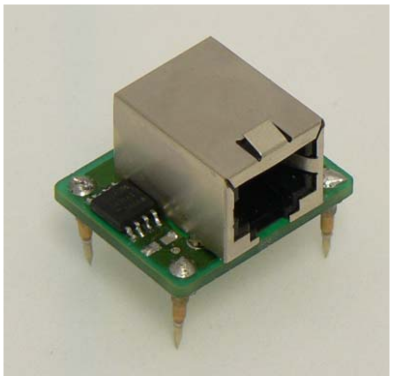


Figure 5.20: The substrate connector.

5.7 User Experience

As a summary of the contents of this chapter, this section provides an overview of the experience of the physical design from the point of a hypothetical user: To begin with, a user is presented with a sheet of substrate in its "raw" rectangular form, as it is produced (Figure 5.22, left). The substrate can be cut into an arbitrary shape using a sharp knife (Figure 5.22, center, right).

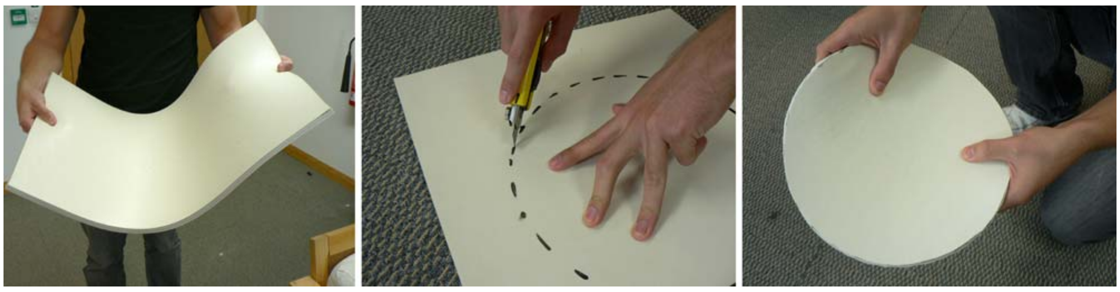


Figure 5.21: The substrate can be flexed and cut to shape.

The substrate defines the area of the interface, and can be viewed as the physical equivalent of a screen or window framing the interaction with a computer system. The user literally cuts-out planar sections of interface façade out of the substrate material, which can be distributed and placed around the environment and used to augment the surface of walls, objects, furniture and other architectural elements. The substrate can be permanently affixed in place by peeling off the adhesive backing (Figure 5.23).

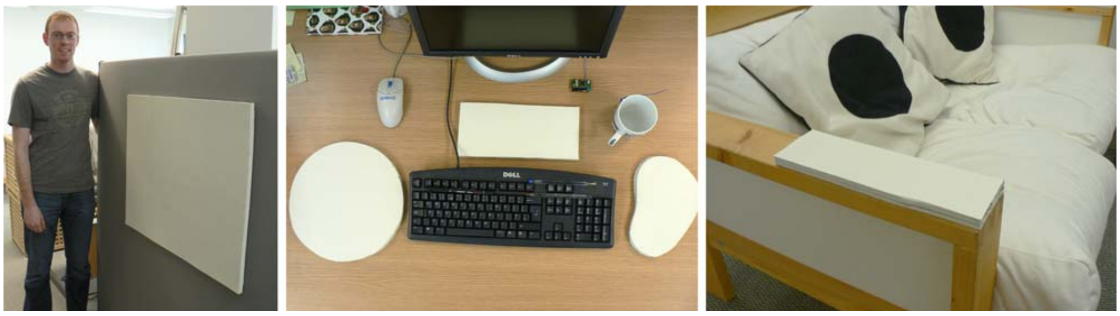


Figure 5.22: Augmenting surfaces with the substrate material.

The user then attaches a substrate connector in a convenient location on the substrate, generating a socket that allows the substrate to be connected to the network adapter (Figure 5.24).

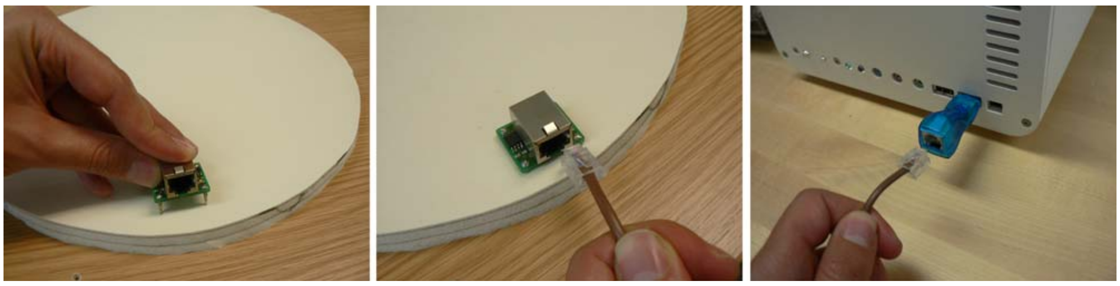


Figure 5.23: Connecting substrate areas.

Multiple substrate connectors and additional cables can be used as interconnects to extend the network signal from a single adapter across multiple substrate areas (Figure 5.24).

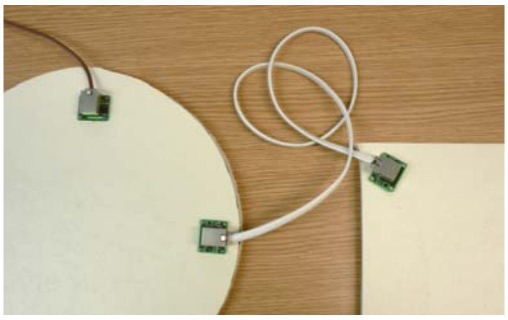


Figure 5.24: Interconnecting substrate sections.

With the substrate shaped and deployed, the user can take controls and freely arrange them on any of the substrate areas. To add a control, the user simply pushes its connecting pins into the surface. Pins make it easy for users to attach or detach a control, and as pin connectors do not involve sockets, users can place devices anywhere on a substrate (Figure 5.25).

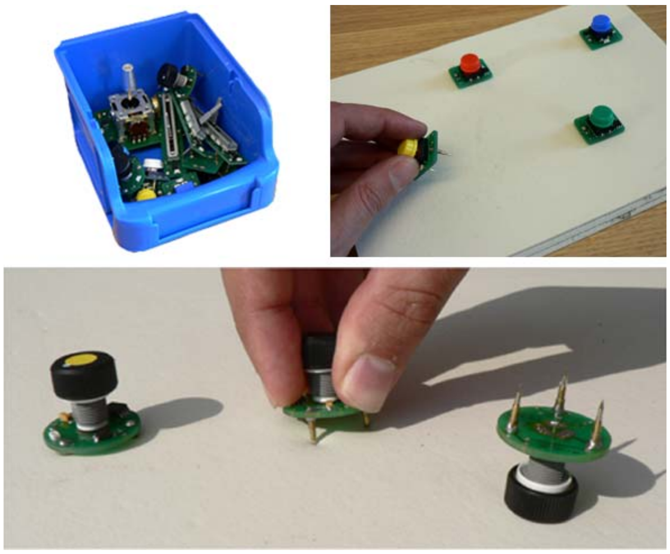


Figure 5.25: Adding and removing controls.

The user is free to continually add, remove and rearrange controls on the substrate. A newly added control is promptly available for interaction – as soon as it has been correctly detected and initialised by the system, the control's beacon flashes twice to inform the user of the fact, reinforcing the property that physical attachment equals digital connectivity.

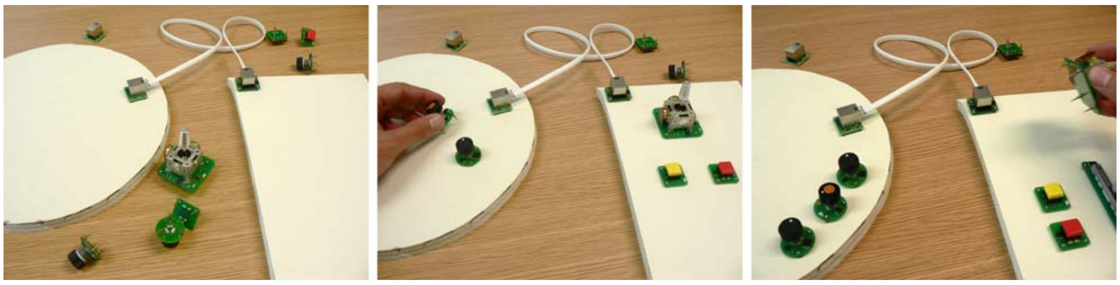


Figure 5.26: Rearranging the control configuration and layout.

From a system perspective, there is no distinction between the act of configuring the arrangement and composition of controls, and that of using and manipulating them. The user is free add, remove, rearrange and manipulate controls ad hoc: all these actions can be detected and interpreted as interaction modalities (c.f. Figure 5.26).

5.8 Discussion

The VoodooIO design gives a concrete physical shape to the architecture for networked controls that was presented in Chapter 3. The design addresses several of the challenges involved in making the physical interface flexible, in a way that affords a high degree of flexibility and user adaptation.

A property of relevance for flexible use is the unconstrained device placement as provided by the VoodooIO substrate-connector design. The behaviour of a connection is the same whether near the edge or near the centre. Devices need not be aligned to any topology but can be aligned very precisely if the user wishes to do so.

Due to their pushpin characteristic, attachment and detachment of controls from a substrate is effortless but controls are held firmly in place while they are attached. The choice of connector places emphasis on the act of pinning, which has strong associations both with “temporarily fastening” and with “holding fast”. This provides for flexible adaptation and also for “freezing” of physical interface configurations: an important feature that allows users to learn and internalise interface arrangements in extended periods of use [84]. The idea of a pushpin connection mechanism was first introduced in the Pushpin computer project as a way to power sensor nodes through a conductive surface [57] It was later adopted by the Pin&Play project as a technique to network devices that are attached to surfaces [85], and subsequently in the design of the POUTS actuated pin-board [65].

Other connection mechanisms that allow users to easily assemble and connect interface components have been explored in a number of projects. The Triangles tangible interface uses triangular pieces with magnetic edges, which let any one piece connect to another and double as a serial data link. The Networked Surface is an infrastructure that supports high-speed data networking between objects placed on a flat surface [78]. The Magic Surface also provides bi-directional communication with objects through a surface, achieved in a contactless manner by using a magnetic-based technique [55]. The VoodooIO designs shares the common with all these, that the act of physically connecting objects also results in a digital connection being established.

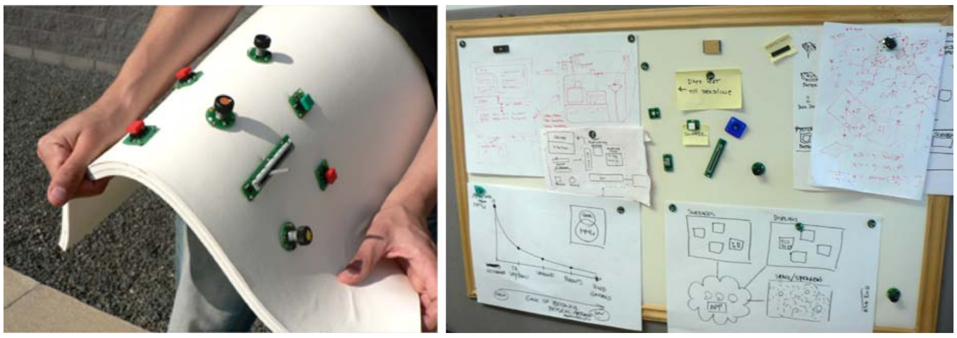


Figure 5.27: Physical flexibility allows users to appropriate the interface and integrate it into their environment in interesting ways (e.g integrated into a notice board, right).

The physical interface remains malleable and flexible during its use, and can be appropriated, adapted and personalised by users in novel and unexpected ways (c.f. Figure 5.27). The substrate design and implementation allows for construction of interactive surfaces in different form factor that are not necessarily rectangular or planar. The substrate material can also be laid out around three-dimensional objects, or embedded invisibly with other types of surface. In this respect, VoodooIO shares a similar philosophy to user interface designs that integrate with non-computer centred practices and encourage interaction away from the desktop. Examples of such interfaces include the Collaborage system to track objects arranged on walls [Moran 99 Collaborage]; the Designer's Outpost that supports working with mixed paper and digital media on large screens [54], and work exploring how paper-based collaborative scheduling practices can be supported with computation [40].

The physical design of VoodooIO presents a novel solution to the problem of making the physical interface flexible, using the architecture for network controls described in Chapter 3 as a technical basis for the design. The theme of the next chapter is how the VoodooIO system is exposed for application development: it introduces the application programming interface model, and a set of software tools that act as glue between the physical interface and computer applications.

Chapter 6

Application Support Tools

Conventional software applications are designed to be used with standardised human interface devices such as keyboards and mice. The VoodooIO design differs from traditional interface devices to the extent of requiring a bespoke set of software tools to support application integration and development. This chapter introduces a number of tools to support application development, which are targeted at users and developers with varying levels of programming expertise: a software infrastructure that provides an application-level interface to the underlying system; a collection of libraries and modules that provide support for application development in existing programming platforms, and high-level configuration tools allow end-users to configure VoodooIO for use with conventional applications.

6.1 Application Infrastructure

It is envisioned that the physical interface to ubiquitous computing applications will need to be distributed throughout the environment, and accessible by multiple applications running on a variety of computing platforms and systems [95].

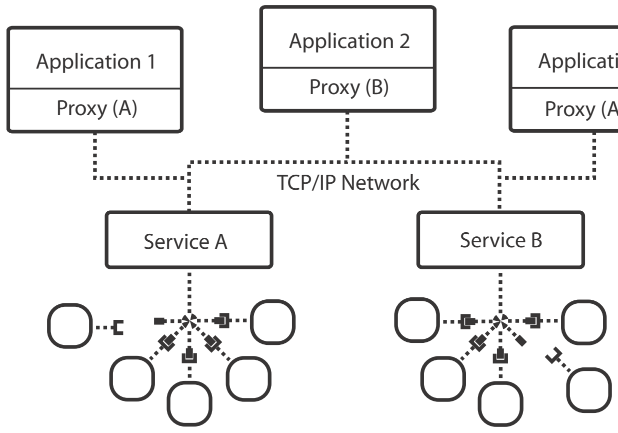


Figure 6.1: The application interface follows a service/proxy model.

To this end we have developed a software infrastructure that acts as a application-level device driver to the VoodooIO system, and allows applications to access the hardware remotely across a network in a platform-independent way. The application infrastructure follows a service/proxy model, where the physical interface is exposed as a service on a network that client applications can access through a proxy object (cf. Figure 2.12).

Rather than thinking of the physical interface as a peripheral device for a computer, each individual network of controls is treated as an interface service. Applications can connect to this service via a proxy which provides transparent connectivity to the service over a standard TCP/IP network. The purpose is to decouple applications from the physical interface in such a way that the two need not be co-located in the same machine, room or even country. A service can act as a shared interface resource, allowing multiple applications to connect to it; conversely, a single application can connect to multiple services at once via multiple Proxy connections.

Services are stateful, retaining localised representations of a VoodooIO network of controls. Applications can connect and disconnect to a Service ad hoc without interrupting its operation, or interfering with other applications. When an application connects to a service via a proxy, the proxy is updated with the current state of the service to provide the application with the current view of its control composition: what controls it contains at that moment in time, as well as their respective values. Any subsequent user-interaction with the user controls (adding, removing or manipulating controls) is transparently propagated to all connected applications via their proxies. Any output that one application effects on the controls, and which changes their state, is also propagated to other connected proxies so they can update their internal representation accordingly. All communication at this level is carried out via a custom ASCII-based protocol over a TCP/IP network.

6.1.1 Service/Proxy Implementation

A service implementation is outlined in Figure 6.2: it consists of a 1-Wire adapter, attached via USB to a PC which runs the Software Stack (c.f. §3.7) coupled with a TCP/IP asynchronous server. The server accepts connections from client proxies and encodes interaction events (ControlAddedEvent, ControlRemovedEvent, Control-

ChangedEvent) into an ASCII text message, which is transmitted via TCP to all connected proxies.

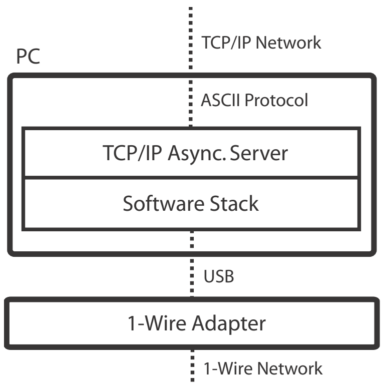


Figure 6.2: Service implementation: the Software Stack is coupled with an TCP/IP server to propagate interaction events to connected proxies.

Request messages from proxies are parsed and forwarded back to the Software Stack for processing. The VoodooIO service is implemented in the C# language for the .NET platform, and can be run a background process on a computer running a Windows operating system. It should be possible to port the implementation to other platforms, and it is conceivable that it could eventually be integrated into the adapter as a single embedded device.

Applications communicate with services via a proxy. A typical proxy implementation is illustrated in Figure 6.3.

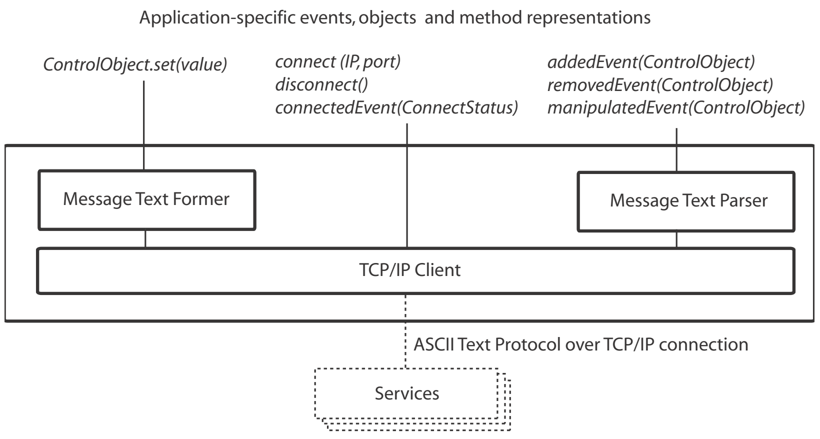

Application-specific events, objects and method representations
ControlObject.set(value)
connect (IP, port)
disconnect()
connectedEvent(ConnectStatus)
addedEvent(ControlObject)
removedEvent(ControlObject)
manipulatedEvent(ControlObject)
Message Text Former
Message Text Parser
TCP/IP Client
ASCII Text Protocol over TCP/IP connection
Services

Figure 6.3: Typical Proxy implementation, providing a platform- and application-specific interface to a Service.

The proxy provides an application with a transparent mechanism to connect and communicate with one or multiple services. Incoming interaction messages from services are parsed and translated into context-specific events or method calls. Output requests are formed as messages and forwarded to any connected Services.

By using a proxy, it is possible to decouple the underlying service implementation from heterogeneous applications, operating system platforms or development languages. Proxies can be implemented to be specific for as a "hook" to an existing application or as libraries that provide programmatic access to a service from a development environment. The protocol used to communicate between services and proxies is included as an appendix to this thesis.

6.2 Developer Tools

Development libraries provide a language-specific programming interface through which to access services. A development library implements a proxy and provides an object-oriented representation of controls which are accessible through a documented API. The UML diagram for a typical library implementation is shown in Figure 6.4.

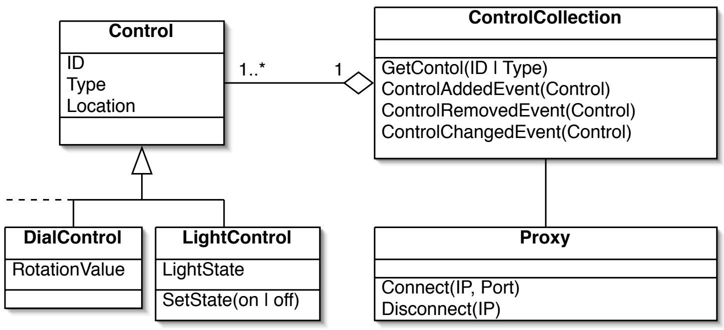


Figure 6.4: An object-oriented library representation

A proxy object encapsulates a TCP/IP client and the processes to encode/decode messages in the ASCII text protocol. Multiple proxy objects can be instantiated to support parallel connections to multiple services. A ControlCollection object provides an access point for the application to access control objects, and raises events when controls are added, removed or have changed their state. Control objects represent particular instances of physical control devices, and encapsulate device-specific methods and properties.

Development libraries have been developed for a variety of programming languages, including C# for the .NET platform, C++, Python, Java, the Processing open-source language and the Adobe Flash content production environment. The rest of this section uses the Adobe Flash integration to illustrate how libraries can be used to extend existing development platforms, and allow developers access to the technology from within a familiar development environment.

6.2.1 Example Library: Adobe Flash

Adobe Flash is a content creation platform intended primarily for the production of interactive online content and animation, but also used widely by product and interaction designers as a way to prototype functional products and interface concepts. The Flash development environment is centred around the concept of a stage on which graphical components can be arranged, and a object-oriented scripting language (ActionScript) that allows programmatic behaviours to be associated with virtual objects. We have developed a library package that, when installed, extends the Flash graphical stage onto a physical substrate stage, and allows physical controls to be associated with functionality by using ActionScript [82].

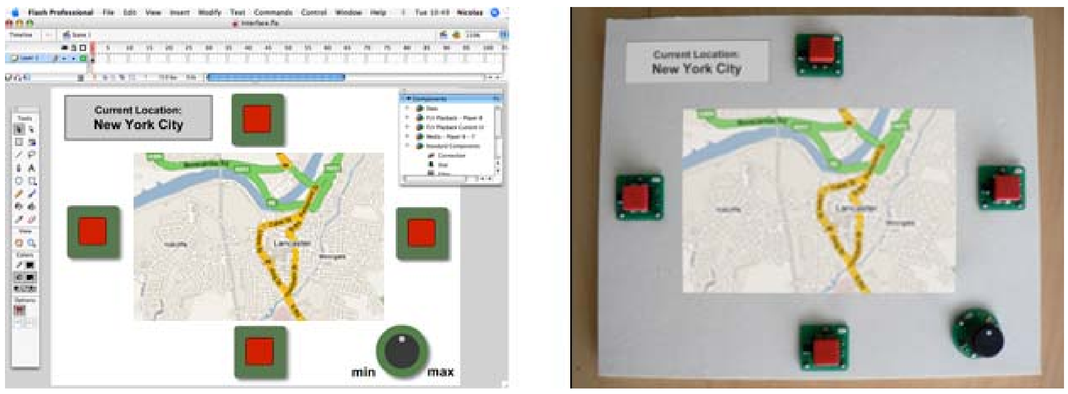


Figure 6.5: The Adobe Flash development libraries extend the graphical stage and controls with a physical equivalent.

The graphical and physical stage are closely coupled, with physical controls represented by virtual counterparts on the Flash stage, and with Flash programmed output visually overlaid on the physical stage. Figure 6.5 shows the two stages side by side for design of a map navigation interface. The physical stage serves as an arena in which designers can work with paper, foam and other materials around the controls, and the graphical stage provides the environment in which controls can be associated with functionality and interactive behaviour. The two stages are kept tightly synchronized, to allow dynamics such as rapid change of the behaviour of a control (affected on the graphical stage) and immediate testing by manipulating the respective control (on the physical stage).

Installing the library introduces a set of new components to the Flash component repository (cf. Figure 6.6). These are:

- a connection component through which a Service can be selected (connections to multiple services are possible, facilitating programming of distributed physical interfaces),
- a component each for the available physical control types (each with predefined ActionScript event handlers for the three core events: added, manipulated, removed),

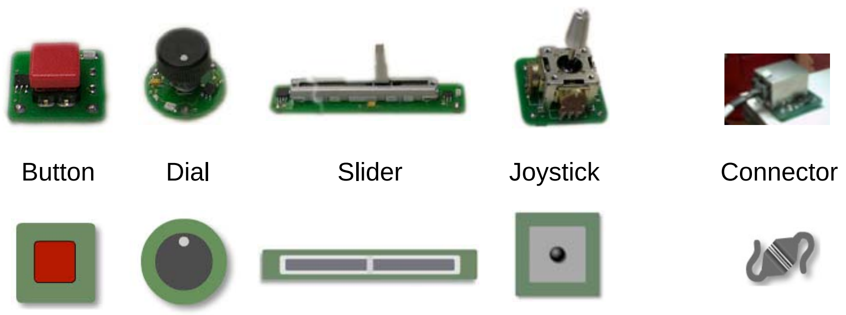
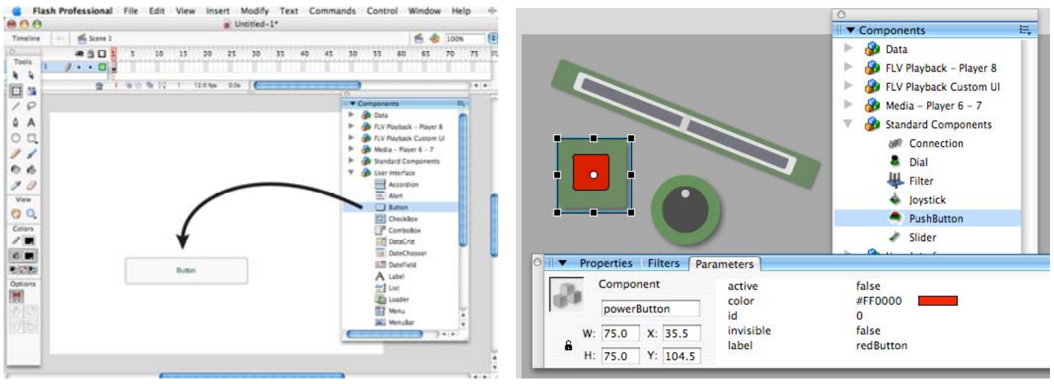


Figure 6.6: Physical controls (top) vs. Flash components (bottom).

In addition, a filter component allows filtering of events (for example to filter events from a particular service if multiple connections are made). Users interact with physical controls as they do with any other Flash component, using the standard mechanisms for instantiation of a component, arrangement on or off the stage, and setting of properties and parameters through graphically inspectable panels (cf. Figure 6.7).



Component powerButton

W: 75.0 X: 35.5
H: 75.0 Y: 104.5

active	false
color	#FF0000 🟥
id	0
invisible	false
label	redButton

Figure 6.7: Users interact with controls as they do with any other Flash component.

Physical controls and corresponding graphical components can be brought onto their respective stages independently of each other, in no prescribed order. Associations can at any time be made and changed, either by using the unique ID that each physical control has built-in, or by using a name that may be pre-programmed for a control, or interactively assigned.

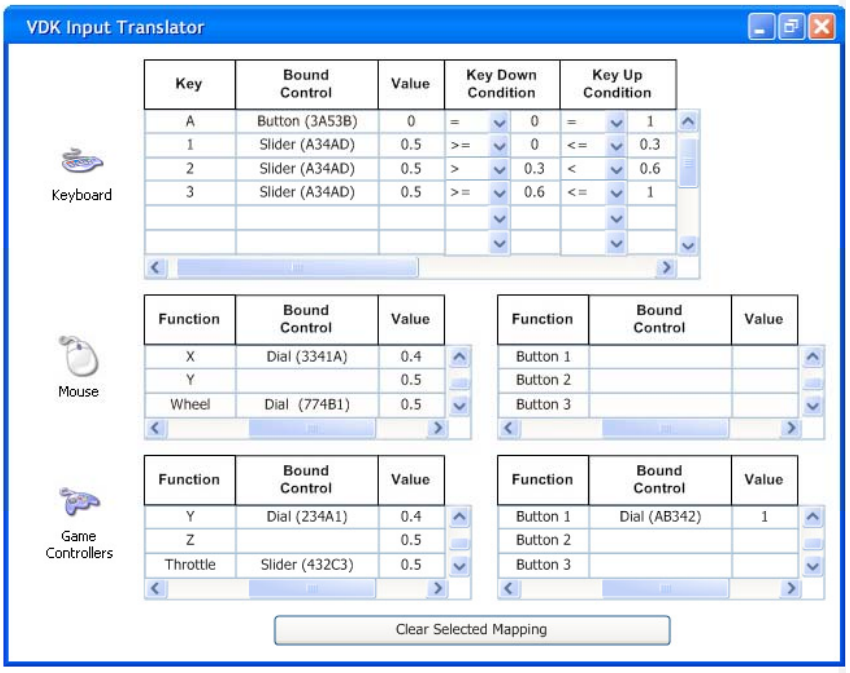
6.3 End-User Mapping Tools

In addition to the development libraries, we have developed a set of tools that act as "glue" between the physical interface and conventional applications. These tools are targeted at a wider user group than application developers: using them involves configuring parameters or training the software, rather than programming or writing scripts. These tools provide an "out-of-the-box" mechanism to hook into conventional application functionality, providing a way to perform ad hoc mappings between physical controls and application functions.

6.3.1 InputMapper

Figure 6.8 shows the configuration panels of one of the tools: the InputMapper allows users to map control events – adding, removing or manipulating a control – onto standard human interface device actions. A mapping is specified by associating a specific control (using its unique ID) with a particular action. For example, the act of pressing a button control can trigger a simulated keyboard key press, or a joystick control can be mapped to the position of the mouse cursor on the screen. It is also possible to specify simple conditions, such as: "if the position of slider control X is less than

50%, trigger action A, if the position is more than 50%, trigger action B." Multiple actions can be associated with a single control.



Key	Bound Control	Value	Key Down Condition		Key Up Condition	
A	Button (3A53B)	0	=	0	=	1
1	Slider (A34AD)	0.5	>=	0	<=	0.3
2	Slider (A34AD)	0.5	>	0.3	<	0.6
3	Slider (A34AD)	0.5	>=	0.6	<=	1

Function	Bound Control	Value
X	Dial (3341A)	0.4
Y		0.5
Wheel	Dial (774B1)	0.5

Function	Bound Control	Value
Button 1		
Button 2		
Button 3		

Function	Bound Control	Value
Y	Dial (234A1)	0.4
Z		0.5
Throttle	Slider (432C3)	0.5

Function	Bound Control	Value
Button 1	Dial (AB342)	1
Button 2		
Button 3		

Figure 6.8: The configuration panel of the InputMapper end-user configuration tool.

6.3.2 MIDIMapper

The MIDI specification (Musical Instrument Digital Interface) provides a standardised way for electronic instruments and musical software to transmit event messages, such as sequences of notes and pitch/volume changes. With the MIDIMapper, users can perform associations that allow MIDI messages to be constructed and sent from a virtual MIDI port. For example, a slider control can be mapped to a pitch-modification on a specified channel, which is issued every time the slider is manipulated. As with the InputMapper, it is also possible to associate control addition/removal with standard MIDI control events, for example: mapping the addition of a button control to trigger a Note-On message (normally sent when a musician presses an instrument key), and its removal to a respective Note-Off message.

6.3.3 Mapping by Example

Both the InputMapper and MIDIMapper tools allow users to configure the behaviour of the physical interface on conventional applications, hooking into their functionality by simulating events that can be received by the application's interface (such as keyboard input or MIDI messages). Although targeted at non-expert users, these tools still require some level of skill to configure – particularly when specifying complex conditional mappings. In an effort to lower the threshold required to create custom mappings, we also developed a mechanism for end-user mapping that allow users to map-by-example. Figure 6.9 shows an alternative InputMapping interface (restricted to keyboard input only) which interactively guides users through the process of creating a mapping between an input control and a keyboard action.

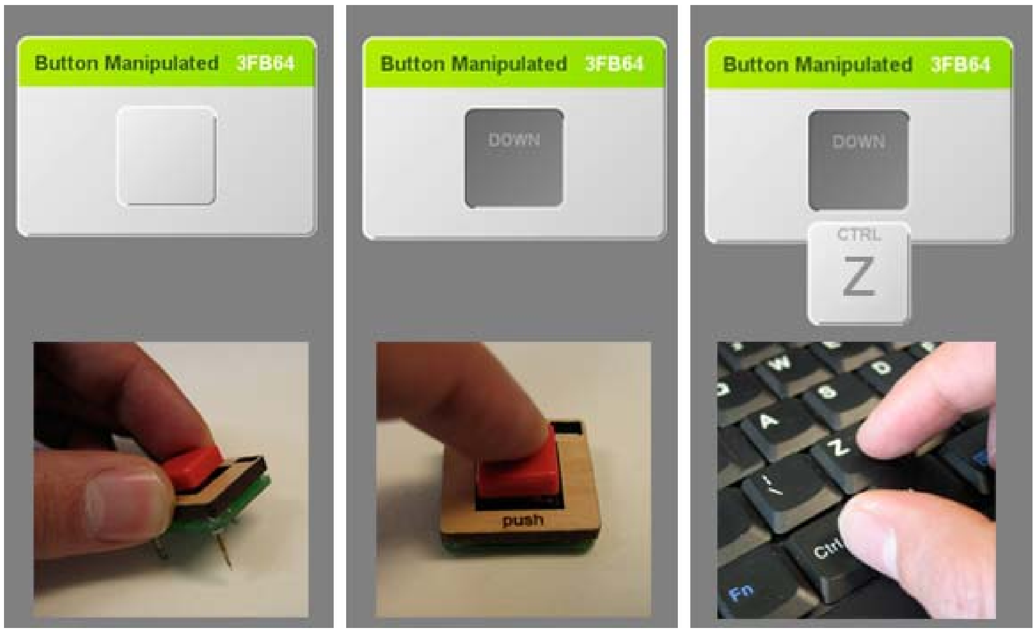


Figure 6.9: Mapping-by-example.

The graphical interface allows users to create mappings by example, as a three-step process:

1. In the first step, the user is asked to select an input control by attaching a new control to the substrate, or manipulating an already present control. A graphical timer is displayed to indicate that the user has some time to change their mind and select a different control, before the selected control is nominated to be mapped; this is reflected by a graphical icon which represents the choice of control that appears on the left hand of the GUI.

2. Once a control has been selected, its icon moves to the middle of the screen, and the user is asked to set the control to a particular state (e.g. press/release a button, or turn a knob to a specific position). Once the user stops manipulating the control, a second timer countdown begins; when it expires, the icon moves to the right of the screen to indicate that the second step has been completed.
3. As a final step, the user is prompted to perform any keyboard commands that they want to associate with that control (and which will be recalled whenever the control is set at that particular state). The user may enter a single keystroke, or a sequence of key-presses.

Once the process has been completed, the mapping GUI disappears from view and the mappings are put into operation. In this way a user can, for example,to program a VoodooIO button-press to effect an application shortcut which needs to be accessed frequently, or a string of text that needs to be entered often (e.g. the keystrokes Control+Z for "undo", or a user's e-mail address), thus generating physical access to the shortcut/text via a single button press. The process can be repeated multiple times with a single control, for example: a mapping for a rotary knob control, where turning the control counter-clockwise generates virtual key-presses for the keys "L", "E","F", and "T"; and turning the knob in a clockwise direction causes key-presses for the "R", "I", "G", "H" and "T" keys to be input.

Like the passage of time, the process of creating a mapping is sequential and unidirectional. The timer-based design of the tool leads the user from one step to the other, detecting when a user has completed a step and prompting for them to perform the next. Time, inferred user intent, and graphical feedback mediate the interaction. This minimises the need for using additional interaction devices to advance the process (e.g. having to use the mouse to click on a "Next Step" button) and maintains the focus strictly on the VoodooIO controls and keyboard keys that are being mapped. The intention is to provide a simple-to-use interface that "talks" the user through the process of creating a mapping, in a way that the process itself presents minimum interruptions to the normal workflow of interaction.

6.3.4 Example: GoogleEarth Interface

We illustrate the dynamics of using the mapping tools by reporting on an experience where the InputMapper was used to extend the interface of a standard desktop application [89]. The Google Earth software allows a user to browse a collection of satellite imagery spanning the whole surface of the globe, making it possible to point and zoom to view any place on the planet (c.f. Figure 6.10). The resolution of some of the imagery is astounding, and after initial exploration on a desktop computer we sketched ideas for a larger-scale interface using physical controls in conjunction with a wall-mounted projection surface.

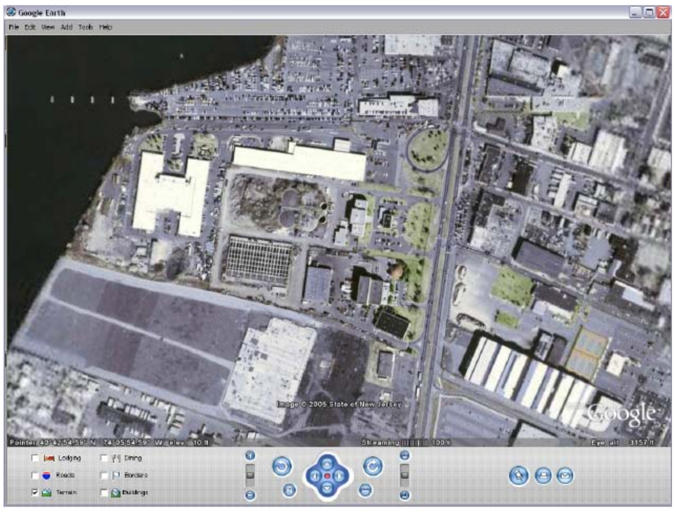


Figure 6.10: The Google Earth graphical user interface.

After playing with Google Earth for a while, we found the most interesting actions to be zooming and panning (others include rotating the image, jumping to a specific place, and adding place-markers). Below is a sketch of the interface idea (Figure 6.11). The Google Earth screen (without graphical controls) is projected onto a wall-mounted substrate area. Four buttons allow the user to pan over the image, and are intended to work like the arrow keys on the keyboard. The fourth control is a rotary knob that controls the zoom level of the image.

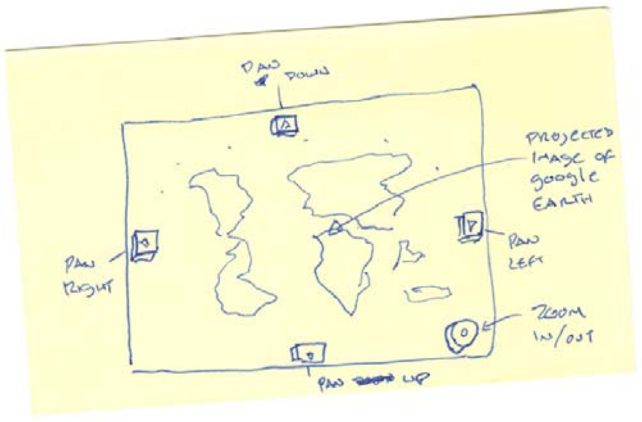


Figure 6.11: Our sketch for a physical interface (right).

The InputMapper tool was used to associate VoodooIO controls with GoogleEarth keyboard shortcuts. For this exercise we picked four button controls, and, for each one, mapped their Pressed / Not Pressed states to simulate Key Up / Key Down keyboard events.

By default, the keyboard arrow keys can be used to pan across the Google Earth visualization, so the application needed no modification. Mapping a knob-type control to the Zoom In/Out function was only slightly more involved. Rotating the knob to the right causes a virtual "+" key to be pressed, which causes Google Earth to zoom in. Rotating it to the left generates a virtual "-" key down event, and rotating the knob to a middle position generates a "+" and "-" key-up events (equivalent of releasing the physical keys) and bringing the zooming to a halt.

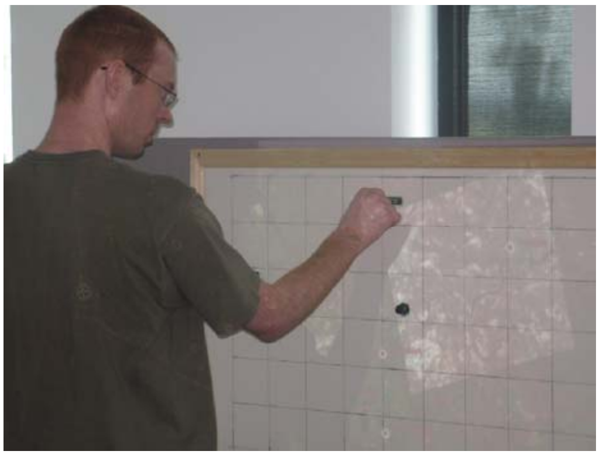


Figure 6.12: Testing the system after deployment in the lab.

We had originally arranged the four "arrow" buttons on the board as in the original concept sketch. As Dave, a colleague in the lab, tried it, he very quickly mentioned that the mapping felt wrong, and we could understand what he meant: pressing the button on the right caused the image to scroll to the left, as it had done sensibly on the desktop computer screen – but now, in the context of a larger and more immersive display it felt wrong (Figure 6.12). We physically switched the left and right buttons around, so that pressing on the right-hand one caused the image to scroll to the right. Note that no software remapping of button-to-action took place, we simply rearranged the physical arrangement of the controls to achieve this in a few seconds.

During this time Martyn, another colleague was carrying out some data collection in the lab, and while waiting for the process to finish he became an unsuspecting user-tester of the interface. The buttons remained in their place, and the zoom dial was attached to the lower-right hand corner of the board, as in the original sketch (Figure 6.13, left).

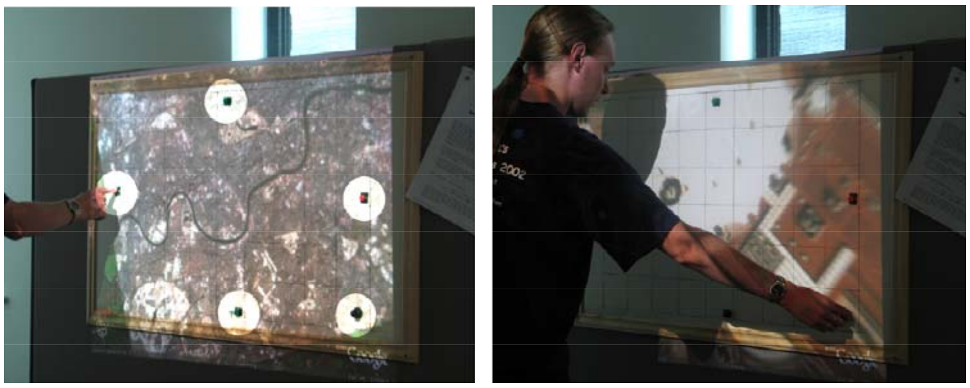


Figure 6.13: The layout of controls as originally sketched (left) and interface occlusion (right)

He was able to pan and zoom in and out using the controls, although the interaction seemed slightly uncomfortable at times. For example he was standing to the left of the board most of the time as to not occlude the projected image, except when reaching for the zoom dial (Figure 6.13, right). Also, he is left-handed so manipulating this control with his right was not ideal. So he unfastened the zoom dial from its original position and attached it instead to the top left corner of the board (Fig 6.14, left). Problems solved.

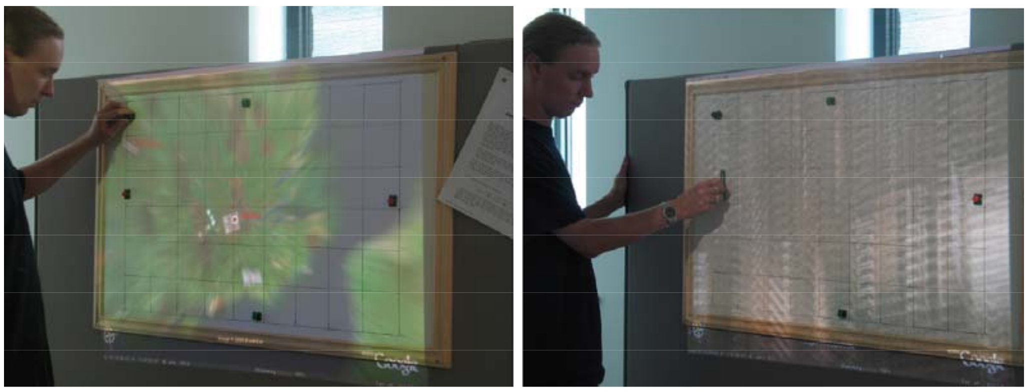


Figure 6.14: The zoom dial quickly repositioned (left) and trying out a slider for panning (right)

Martyn then suggested that instead of using the buttons to pan, we should try using sliders to do the task: we also picked up two slider controls, and using the same three-state mapping as the zoom knob (back-stop-forward) we mapped one to the Left/Right arrow key events and the other to the Up/Down arrrows. Sliding one slider to one side left causes the image to scroll continuously to the left, slide it to the other end and the image begins to pan to the right. He attached the vertical-scroll slider alongside the left side of the board, underneath the zoom control (Fig 6.14, right). He tried scrolling with it, found that the mapping was "reversed" so he simply unfastened the slider, turned it 180 degrees, and attached it again.

An added advantage of using the sliders over the buttons was the fact that you could set the map to drift in a particular direction without having to keep a button constantly depressed, which allowed one to step back and appreciate the effect of flying over the landscape. Martyn then added the second (horizontal) scroll slider, oriented underneath and perpendicular to the first one (Fig 6.15, left). Finally the now redundant buttons were removed to clear up the display area – they weren't being used anyway now that the sliders appeared to be a better way of panning around (Fig 6.15, right).

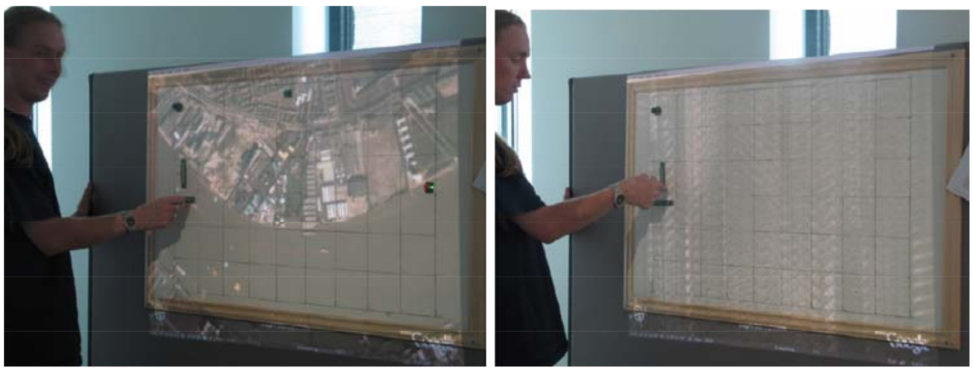


Figure 6.15: Adding a second scroll slider for panning (left) and removing the buttons (right).

This exercise illustrates how the application support tools can contribute to the fast development of ideas for physical interfaces. More importantly, it highlights the flexibility that the technology provides to interaction designers and users: it allows exploration of interface configurations in very fluid manner – functional mappings are quickly defined, then physical controls are simply added, removed, or repositioned in order to modify the user experience without need for reprogramming, recompilation, or stopping and restarting of the application.

6.4 Discussion

The development of the VoodooIO application infrastructure is motivated by the changing relationship between interface devices, computing resources and software applications. As Mark Weisberg predicted in 1991, computing resources are becoming more plentiful, increasingly networked and interoperable [95]. Users commonly interact with remote applications via the web and exchange information between heterogeneous devices. To this end, the VoodooIO application infrastructure allows VoodooIO devices to be flexibly interfaced with applications running on any networked machine that supports the TCP/IP protocol.

The iStuff framework for physical user interfaces uses the iROS middleware to distribute interface events amongst distributed machines, and is based on Java to provide a degree of platform independence [4]. The VoodooIO infrastructure takes a more lightweight approach by obviating the need for a middleware layer between interface services and applications. At the same time, the service/proxy protocol facilitates integration with existing middleware platforms such as iROS to benefit from their advanced features. Another related development is a recent extension to the Phidgets platform, in the form of a software extension that provides a remote application interface via a webservice [72]. Marquadt and Greenberg report on the Shared Phidgets toolkit, which supports the rapid prototyping and debugging of distributed physical interfaces by providing transparent access and control of Phidgets devices across a network [60].

The software tools presented in this chapter are intended to support users with different levels of programming know-how – from everyday users to skilled programmers. As a way to lower development complexity, the Papier-Mache project introduced a high level programming model for building tangible interfaces out of electronic tags, barcodes and computer vision [53]. The authors of the Papier-Mache work discuss the notion of a "toolkit" as software where the "user interface" is an API, and the users are programmers. In this sense, the VoodooIO development libraries are also targeted users who are developers of applications. The libraries for Adobe Flash are intended to provide a development platform for interaction designers and product developers, who are already well versed in the practice of using Flash to prototype user interface concepts (c.f. §7.1, Product Prototyping). A similar approach is taken by the Phidgets platform, which allows developers to access physical controls through software COM objects that can be accessed from a number of programming environments [37]. The Calder toolkit also integrates with existing development platforms, namely the Macromedia Director package [56] in order to extend current practice with new tools, rather than introducing new development concepts and environments that have to be learnt by users. The d.tools project takes a different approach, in that it develops its own software platform specifically designed to support the development and evaluation of novel physical interfaces and interactive products. Unlike Phidgets and Calder, the d.tools design features a flexible and extensible hardware architecture that allows different hardware-oriented platforms (such as Phidgets and VoodooIO) to be used with the d.tools software environment [43].

One of the original aims of developing VoodooIO as a platform for physically flexible interfaces was to enable users to tailor and modify the interface during runtime. When the interface allows users to add, remove and swap control configurations on the fly, it becomes necessary to address the problem of supplying the user with mechanisms to dynamically map changing control configurations to application functionality. This is the problem addressed by the Patch Panel: a tool that provides dynamic reconfiguration of interactive mappings between networked components [3]. The Phidgets WidgetTap offers a way for physical control devices to be bound to graphical widgets through a drag-and-drop mechanism [38]. Like the VoodooIO InputMapper tool, the WidgetTap allows users to map interface controls to conventional applications by simulating mouse and keyboard input events on the GUI. The ICON input configurator provides a graphical tool that allows users to dynamically re-map input from standard control devices to application functions [23]. ICON is similar to the range of VoodooIO application support tools in that it caters both for programmers wanting to quickly test interface ideas, and for end-users wishing to create and tweak their own user interface designs.

A number of different research projects have concentrated in providing support for end-user customisation of ubiquitous computing systems. Rodden et al. report on a design that allows users to configure interactive services in a home setting: by assembling jigsaw pieces that represent services and network objects, users build simple programs such as "if someone rings the door bell, take a picture and send it on my PDA" [75]. The Media Cubes [6] and iCAP [80] both address similar problems of supporting end-user programming of interactive applications that exist beyond the desktop. While the VoodooIO end-user mapping tools are targeted at users of conventional personal computer applications with graphical user interfaces, the underlying VoodooIO application infrastructure supports the development of physically distributed, ubicomp-style interfaces. An area of future work is to explore alternative design for mapping tools for ubiquitous environments, and explore avenues for integration with existing tools.

The motivation for a VoodooIO tool that supports mapping-by-example is to reduce the complexity involved in the process of specifying control mappings, particularly for novice or casual users. The concept of programming-by-demonstration, as a way to simplify end-user development and tailoring, has been explored in a number of research projects, including: the IBM Koala system for web automation [58]; the a CAPella environment for prototyping context-aware applications [20], and, of particular relevance to VoodooIO, the training-by-example system for reconfigurable input controls described in [11].

The VoodooIO application support tools (including an application infrastructure, programming libraries and end-user mapping tools) completes the implementation of a prototypical technology for flexible physical user interfaces. The tools presented in this chapter complement the physical flexibility supported by the hardware, and provide the software glue between control devices and applications. The next chapter looks at how these tools are applied in practice, and how VoodooIO is used to develop flexible physical interfaces for a number of different application scenarios.

Chapter 7

Exploring User Benefit

This chapter looks at the human interaction issues of making the physical interface flexible, and the lessons learnt through the application of the VoodooIO platform to prototypical application scenarios. The goal is to illustrate the application possibilities that are generated by the new technology, and to study the novel interaction techniques that it enables. User benefit is explored via number of different approaches: qualitative user studies, where subjects are engaged in a real-world interaction task; formal experiments where we focus on a particular interaction technique, and in application examples that illustrate novel interaction concepts. The application areas range from rapid product prototyping through to game-playing, music production and live music performance.

Each section in this chapter describes an individual research effort, and discusses its significance to the wider context of the work. The topics covered are structured as follows: Section 7.1 presents an application of VoodooIO as a tool for product prototyping, which allows product designers to quickly craft and evaluate interface ideas. The tool is evaluated through a set of workshops, where expert professionals are observed while engaged in prototyping an interactive product design brief; Section 7.2 reports on the results of an experiment that looks at the different physical interface adaptation strategies adopted by participants in an observational study; Section 7.3 provides a closer exploration of some of the interaction techniques enabled by VoodooIO interfaces, in the specific application scenario of desktop music production; Section 7.4 explores a different application area: playing games – the section uses self-generated interface designs for two different games in order to illustrate the benefits users can derive from the ability to appropriate their gaming environments with VoodooIO; Section 7.5 presents the results of a study that focuses on the use of control presence as an interaction technique, and Section 7.6 reports on the application of this technique to the development of ad hoc musical interfaces, where the act of interface (re)configuration forms part of the musical performance.

7.1 Product Prototyping

Product design processes are fast and fluid. Ideas are rapidly made tangible using paper and foam to create low fidelity prototypes that are iteratively refined. But for interactive products, the design of behaviour and user interaction usually remains decoupled from the 'physical design' process and is developed on desktop screens with 'flat' representations of the product, using content creation tools such as Adobe Flash to develop the interaction. Various tools have been emerging to allow designers to better couple physical design and interaction design, including techniques for hooking up 3D product models with software simulations via keyboard emulation [34], physical interface toolkits [2], and complete design environments that cover hardware and software aspects [43].

The VoodooIO platform is an interesting tool for product prototyping. The substrate material can be carved to shape the interface façade of a product concept, akin to some of the traditional prototyping techniques that use materials such as foam or cardboard for this purpose. Interface ideas can be sketched quickly by arranging control elements on the substrate, which can be quickly laid out, rearranged or exchanged to explore alternative configurations. Using the Adobe Flash libraries (Flash is already predominantly used environment in design practice) interaction designers can quickly prototype functional user interfaces that include graphical and audio elements.

In order to evaluate the application of VoodooIO to the process of product prototyping, we conducted two studies with external users: one in Munich with two expert Flash developers, and one in Delft with a larger group, primarily from an academic industrial design background [82].

7.1.1 The Munich Experience

The study in Munich was organised as a half-day expert evaluation to which we invited two professional Flash developers from a local start-up. Both participants had a background in computing and several years of professional work experience in Flash application development.

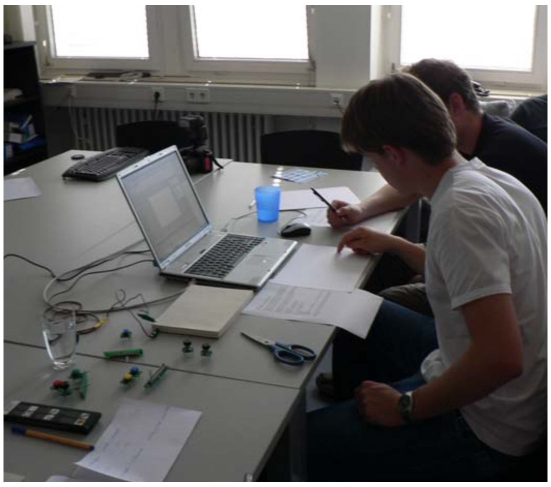


Figure 7.1: The participants for the Munich study were two expert Flash developers.

We started with giving our developer-users a 15-minute introduction to the VoodooIO Flash extensions, using an example design case. They were then given the task to design a user interface for an Internet radio, with a design brief describing the requested functionality (selection of preset stations grouped by categories, etc.), the available resources (a set of buttons, knobs, sliders, etc.) and the design focus (functional interface design, abstracting from issues such as data formats and storage). Our two users were given a VoodooIO kit consisting of substrate sections of several sizes, a full set of Controls, and a laptop running a copy of Adobe Flash, with the VIO libraries and associated documentation installed (Figure 7.1).

The participants were given two hours to jointly work on their task. This was followed by an interactive session, in which they explained their design and were challenged to carry out a change in the interface on-the-fly (simulating change requests customers might have in a design session). Finally, we invited and collected feedback on the design experience.

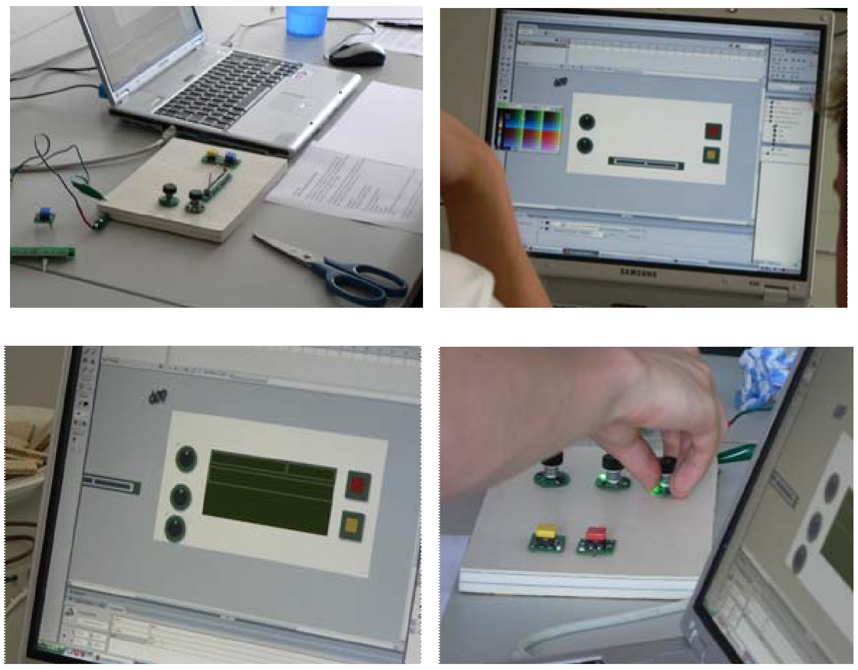


Figure 7.2: The participants initially moved from paper design and physical interface layout to work within Flash (top), and at later points introduced graphical elements first and then reflected and tested them on the physical stage (bottom).

Figure 7.2 shows a series of photos taken during the design session, and indicates how the two developers progressed in general with their task. Initially they sketched a crude design on paper and then laid the interface out physically with VoodooIO controls. They then switched their attention to design on the Flash stage, instantiating, linking and arranging the corresponding virtual controls and associating them with functionality programmed in ActionScript 2.0. At a later stage, changes to the interface configuration were first carried out in the Flash environment, and then reflected on the physical stage.

As a final touch, one of the participants sketched a set of control labels and additional illustrations on a piece of paper, which was then inserted between the controls and the substrate, with the pin connectors piercing through the thin paper layer (Figure 7.3).

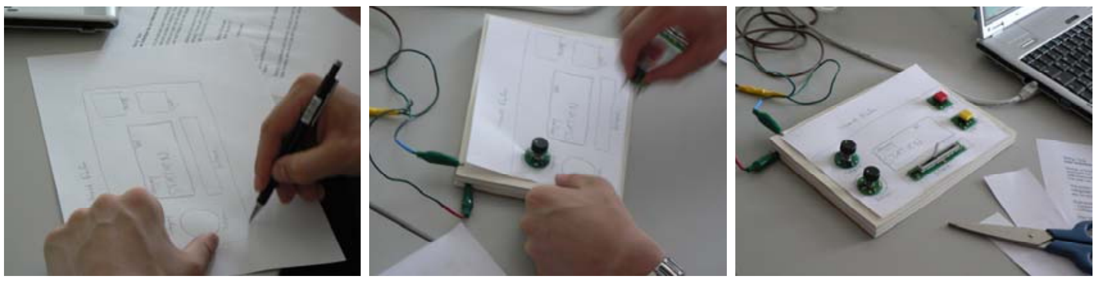


Figure 7.3: Sketching interface labelling using a paper insert.

Throughout the design session, the two developers interacted intensely, in continuous joint reflection over their task (very much exemplifying the reflective prototyping practice that Hartmann et al. discuss in [43]). One of the two developers tended to keep control over mouse and keyboard for work within Flash, and specifically for ActionScript programming. His design partner would simultaneously work with the controls on the physical stage, to generate life input to the script as it evolved, and to continuously try and test the effect of additions and changes in functionality.

The two developers were able to complete their task in the given time, including iterations for refinement (e.g. fine-tuning the response to knob rotation to the specific value range generated by the device). They did not require help other than support we had integrated in the tool environment (documentation of the VoodooIO API), and they were able to completely abstract from VoodooIO technical detail (e.g., they did not have to understand how controls are detected and networked, and at some point during the session one of our users suggested to his partner to "stick the two [knobs] further apart", suspecting they might interfere with each other when to close together). After completion of the task, the two developers were challenged to replace a slider they had selected for volume control with a rotary knob. This only required them to physically replace the devices, to instantiate, name and bind a virtual knob, and finally in ActionScript to replace the name they had used for the slider with the knob's name, all of which was achieved in less than a minute.

In the final feedback session our developer-users reported that working with VoodooIO and the Flash libraries was "*identical in terms of programming*" to routine Flash development, "*all you had to know in addition was the VoodooIO events but there are only three any-*

way". They also speculated that development with separate physical input devices was "*probably faster because you don't have to go through menus [to trigger actions]*", and also because mouse and keyboard focus always remained on the programming task, while the VoodooIO extension served for testing. The users also noted the fun factor of the system, and of being able to immediately see the effect of what you do.

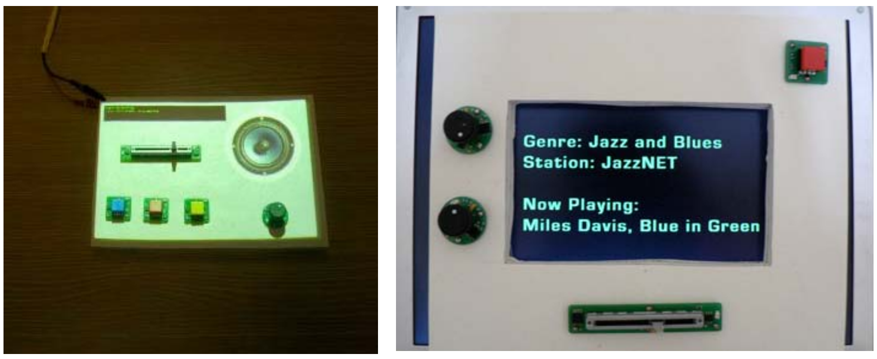


Figure 7.4: Integrating graphics by overhead projection (left) and a window cut-out (right).

Finally we prompted our developer-users to suggest improvements to our tool. Among others this resulted in consideration of how displays could be integrated with the physical stage. This led to the idea that an overhead projection could be used (Figure 7.4, left), or, alternatively, making cut-outs in the physical stage to insert a display or to look through to a display underneath. As the VoodooIO material can be cut too any shape without comprising its functionality, it was easy to try this out (Figure 7.4, right).

7.1.2 The Delft Workshop

The second study was held in the Industrial Design Department of the Technical University of Delft. It was organised as a workshop with an interdisciplinary project team of about 15, primarily composed of academic staff and students from industrial design departments, with some but not expert knowledge of Flash. The team was given a 30 minute introductory presentation of our system, and then split into a 'red' and a 'blue' group, both given the task to develop a version of the classic Etch-a-Sketch toy for which they were given 2 knobs, 2 sliders and 2 buttons and a substrate sheet as resource (Figure 7.5). The groups were to first develop their own version of Etch-a-Sketch in order to sketch a trace on a projected display with separate controls for X-

and Y-axis of the cursor. Participants where then invited to experiment more freely-with the system, and to try and interfere with each others design (facilitated by exposing VoodooIO events over a shared network).

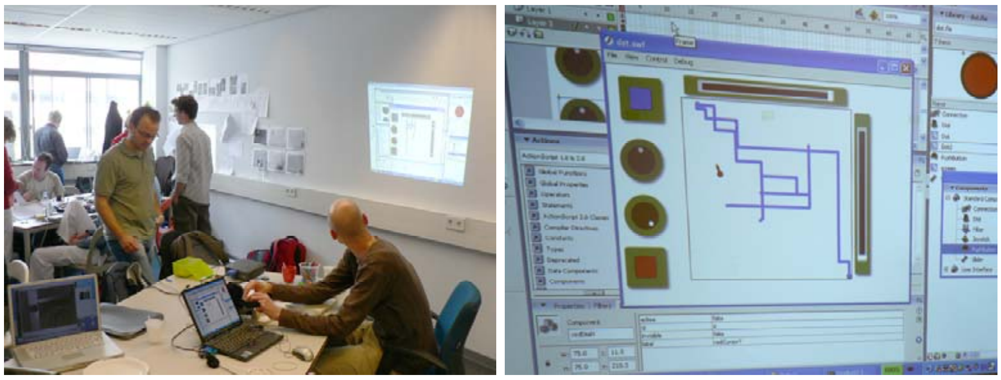


Figure 7.5: Participants were asked to develop two versions of the Etch-a-Sketch toy.

The workshop was concluded with a general feedback session. In contrast to our experience in Munich, the groups struggled more with their initial task, as the participants were not as proficient in using ActionScript. The 'blue' group though quickly got into a more explorative mode, mapping controls in intricate ways to functions such as changing line thickness for etching, so to confuse the 'red' group as to how their Etch-a-Sketch version worked. As programming in ActionScript was taken over by 1-2 individuals in each group, others began to explore how the small set of controls they were given could be physically appropriated. Figure 7.6 shows some examples resulting from this, clockwise from top-left:

- A rotary knob 'dressed up' to modify look and feel.
- Pen and paper used on the physical stage to label and decorate the physical interface.
- A 'voodoo doll' constructed around a strip of VoodooIO substrate, two sliders and a button, for remote controlling (and hi-jacking) the visual display of the other group.
- A slider customized with rubber-band to be self-centering and usable for rate control input (as opposed to absolute control).

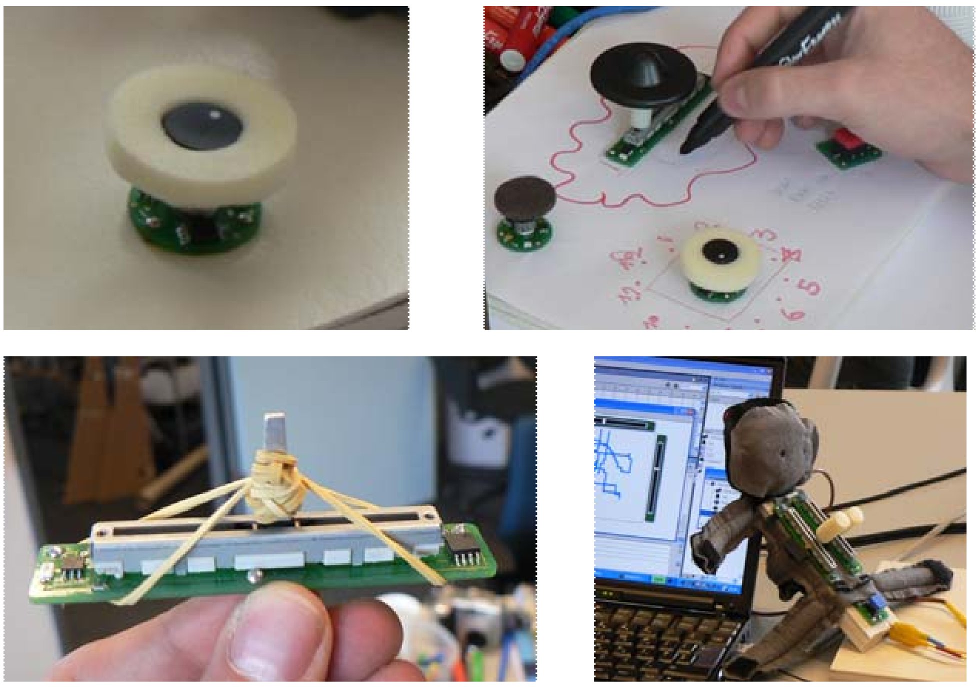


Figure 7.6: Participants in the the Delft workshop explored appropriation of the physical components

In the feedback session, participants welcomed the combination of physical prototyping with programming in Flash, as Flash had been adopted as the first language for design student education in two of the Universities represented in the workshop. Apart from this, feedback was more concerned with the physical sub-system of the VoodooIO tool. Most workshop participants had experience with using Phidgets in product design classes, and in comparison saw in particular the wire-free assembly of VoodooIO devices as a significant advantage. Their other concern was ease of physical appropriation, and for example how more specialised sensors and transducers could be made to work with VoodooIO.

7.1.3 Discussion

Designers have responded to the increased embedding of computing in consumer products with techniques for coupling hard and soft representations of a product in the design process. The Buck method is based on tethering a functional product model to a PC for user testing of physical controls in conjunction with interface software [71], and the IE system uses micro-switches embedded in foam-core models and keyboard emulation to facilitate a physical-interactive experience within hours of an initial design sketch [34] The same principle, in more rudimentary form and focussed on cardboard prototypes, has been adopted in the BOXES system [46]. In common with these approaches, our tool facilitates linkage between physical model and software parts of a design, but on the physical side with a richer set of controls (beyond switches and touch sensors), and on the software side focused on extension of a design environment already in widespread use in design practice.

A variety of toolkits and design environments have emerged for development of physical interactive systems. Phidgets, for instance, provide physical interface building blocks analogous to widgets in graphical user interfaces [37]. The system was initially targeted at making hardware more accessible by GUI programmers but also provides a Flash API. VoodooIO likewise supports Flash development of the behaviour of physical controls but provides a much tighter integration by giving physical controls an explicit representation as Flash components.

The representation of physical devices within the design environment is a feature our tool shares with d.tools [43]. The d.tools system provides an integrated design environment that supports 'plug and draw' integration of physical devices and statechart-based editing of interactive behaviour. However while the system is more open-ended in terms of hardware that can be integrated it does not provide support for existing authoring environments. Specifically the lack of support for Flash developers has been reported as a distinct shortcoming [43].

VoodooIO is similar to Phidgets in providing a range of physical controls but in addition emphasizes malleability of physical interfaces. VoodooIO does this by providing a substrate material on which controls can be dynamically added, arranged, manipulated and removed. This substrate material effectively serves as a network bus to which controls can be connected effortlessly, wirelessly and faster than via a standard USB connection (faster both in terms of user interaction and network discovery).

Overall, the VoodooIO-Flash integration achieves a very seamless extension of an existing and widely used authoring environment for work across physical and interaction aspects in product design and prototyping. The two design exercises with external users reported in this chapter indicate a very good fit of our tool with existing design practices. On the interaction design side, the tool extends the widely used Flash authoring environment in a manner that is intuitive and effective in hiding technical detail of integrating a physical interface system. And on the physical design side, the physical materials prove to be effective in supporting fast and fluid assembly of controls and in facilitating appropriation with other physical design material such as paper, foam, textiles and rubber-bands.

7.2 Make-Your-Own-Interface Experiment

We wanted to get some additional insight into user acceptance of physical control reconfigurability for everyday interaction tasks by non-expert users. To this end, an experiment was designed to explore the supposition that there exist advantages for the user (rather than the interface designer) in being able to design and construct their own interface control solution. The experiment focused on evaluating the hypothesis that individual users benefit from being able to make a choice about the selection of control type and control arrangement with which to carry out a particular interaction task. In addition, the experiment contrasts the use of a standard "rigid" interface device (a keyboard) with that of a VoodooIO setup [91].

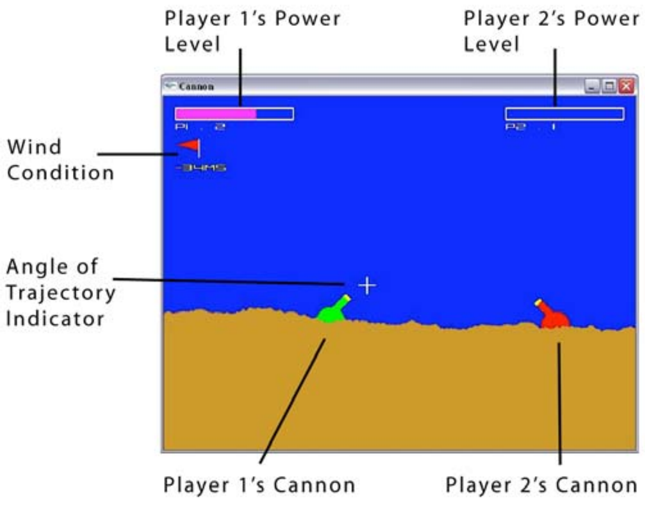


Figure 7.7: Screenshot of the "cannon" game used for the experiment.

A simple game was programmed to take users through the process of composing and using a control configuration. The game was designed as a two-player "cannon" game, where players take turns taking shots at each other's cannon (c.f. Figure 7.7). The cannons are placed on a randomized terrain. The challenge for the players is to judge how to land a direct shot on their opponent's cannon, taking into account variable wind conditions that affect the trajectory of their shot. The player can control three variables relating to their cannon: they can specify the initial angle of trajectory, the power behind the shot, and the moment when to fire.

7.2.1 Procedure

There were 18 participants in the study, 15 male and 3 female. Eleven participants fell under the 21-30 age group, 6 were between 31-40 years old and 1 participant was over 40. All the participants came from an academic environment, mostly researchers, research students and lecturers. 7 of the participants were casual game players, spending roughly 1-4 hours per week on action, role-playing (RPG) and sport type games. 9 of the players were non-starters; they only spent between 0-1 hour per week playing simple card and strategy games on their PC and mobile phones. The remaining 2 players were expert game players who spent up to 6 hours per week playing action, adventure and RPG games.

The experiment involved three rounds of play with the cannon game. Figure 7.8 (left) shows the experiment setup - the game is projected onto a large display with the interaction device laid out in front for both players to manipulate. Before game play commences each player was handed a one-page guide providing a brief overview of the cannon game and what each round would involve.

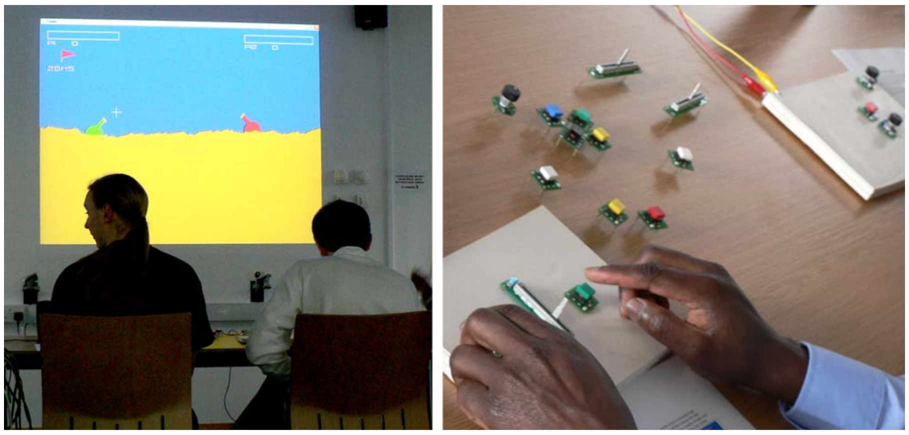


Figure 7.8: Experiment setup: test subjects sit in front of a large projection of the game screen. Each user is provided with a "game pad" made of substrate material, on which to arrange a selection of controls to play the game.

In the first round, a keyboard, shared between the two players, was used as the game controller. The controls for both players (cannon angle, power and fire) were mapped onto a set of predefined keys on the same keyboard. In the second round, players were presented with individual gamepads, measuring about 20x15 cm, and made of VoodooIO substrate. They were also provided with a collection of assorted input controls (dials, buttons and sliders).

At the beginning of the round, each player was guided by the game through the process of constructing their gaming controller from the set of available controls. First, the player was asked to select a control for their cannon's angle setting. At this stage, they were free to insert a control, which was then automatically bound to that function and confirmed graphically on the screen. The process was repeated for the power and fire controls. The cannon angle and power controls could be mapped onto either a dial, a slider or two button controls (to increase and decrease the parameter). The "fire" action had to always be mapped to a button control, but players were free to select a button-colour of their choosing. At each step, players were informed of the possible choices of control they could use by the on-screen display.

After each step the choice of control was confirmed by the system, and the player could test its operation before the game began. Although the association between function and controls remained persistent throughout the duration of the round, the spatial arrangement of the controls was fully configurable during game-play; hence if the physical arrangement was found to be unsuitable, the player could detach it from the substrate and place it again in a new location. In this manner, the control interface reflected each player's preference for the control types used, as well as for their layout on the gaming pad (c.f. Figure 7.8, right).

Before the third round began, players were asked to remove all the controls from the substrate and rebuild their physical interface by repeating the setup process. The reason behind this was to encourage players to re-think their choices of control types and layouts so they could explore other possibilities in this final round.

It should be noted that the controls were not labeled, so, for instance, there was no way of telling which end of a slider control is the 'maximum' or 'minimum.' This was done deliberately in order to allow mistakes in the way that controls are initially arranged, and gain an insight into the way users re-arranged the controls to correct the "mistakes" and have the operation of the controls match their expectations. What follows is an analysis of the results from the experiment, which were mainly based on observation and asking the players a few directed questions at the end of the final round of the game.

7.2.2 Experiment Results

Both expert and casual players started playing the game with no great difficulty. However, non-starters using the keyboard took much longer to remember the keys and had to keep referring back to the introduction guide on which they were outlined. The players took turns in using the keyboard, but they tended to move the keyboard closer to their end when it was their turn to play. One of the first things that was observed was that, as soon as Round 2 started and players were presented with their individual gaming pads, they immediately pulled their pad away from their opponent's and placed it in front of them.

Choice of controls

For the angle control (cf. Figure 7.9), 12 participants opted for the dial, 6 chose the slider (4 arranged it vertically, 1 at a 45 angle, 1 aligned it vertically) and none chose the buttons in the second round. In the third round, 10 participants chose the dial, 4 chose the slider (3 arranged it vertically and 1 at a 45 angle) 4 chose the buttons.

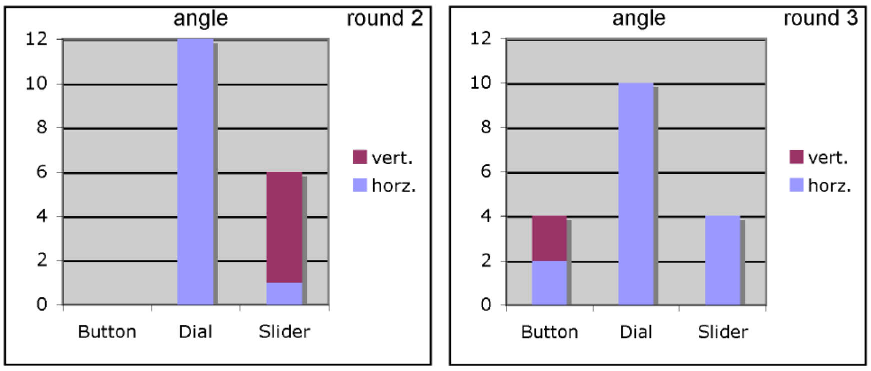


Figure 7.9: Choice of "angle" control in rounds 2 and 3.

The preference for using the dial as the angle control in the second round is interesting, as it does show that the majority of participants chose the hardware control that most closely matches the associated graphical representation (the act of turning the dial matches the cannon's angular movement). Although no player chose to use buttons for the angle in the second round, some did experiment with using them in the third round.

For the power control (cf. Figure 7.10), in the second round 10 participants chose the slider (4 aligned it vertically, 5 chose the horizontally, which 1 changed later to a vertical placement, 1 participant aligned the slider at a 45 angle), 5 chose the dial and only 3 participants opted for using two buttons. In the third round, 11 participants chose the slider for the power control (4 arranged it vertically and 7 aligned it horizontally), 3 chose the dial and 4 chose the buttons.

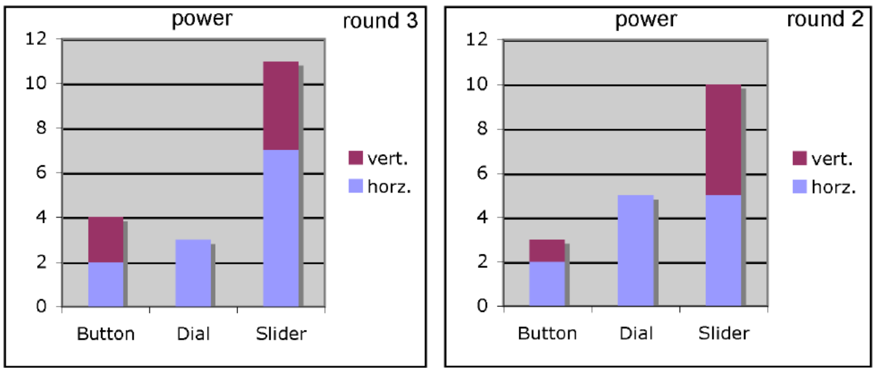


Figure 7.10: Choice of "power" control in rounds 2 and 3.

As with the choice of angle control, the high popularity for the slider as the power control in both rounds does show that participants opted for a control that resembles the graphical representation of the function, as depicted by the power bar on the projected display.

Figure 7.11 shows the choice of button for the fire control in rounds 2 and 3. Although we did not initially set out to assess the impact of the different coloured buttons, the high preference for the red button for the fire control was remarkable, but not totally unexpected – red being a colour often associated with critical or dangerous control functions.

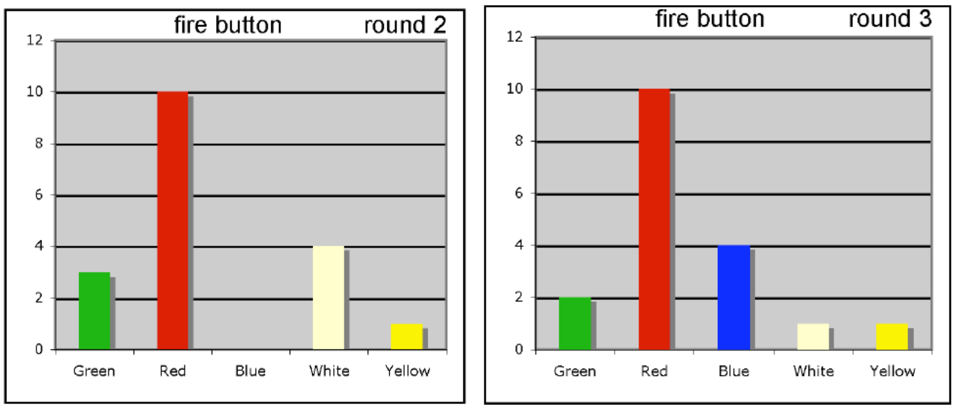


Figure 7.11: Choice of "fire" button colour.

Half of the participants did actually re-arrange their slider controls during the second round. This happened when participants felt unhappy after testing out the control to discover that it reacted in the opposite manner than expected, for example, when the top end of the vertical slider mapped to minimum power or maximum angle. The players would thereafter, and without prompting, turn the controls round 180 degrees to fix the mapping. Other types of reorientation included changing the alignment of the power slider from horizontal to vertical, or physically moving the control to a different location on the pad. In the third round however, fewer participants actually re-oriented their controls during game play, indicating that they had settled with a suitable control arrangement.

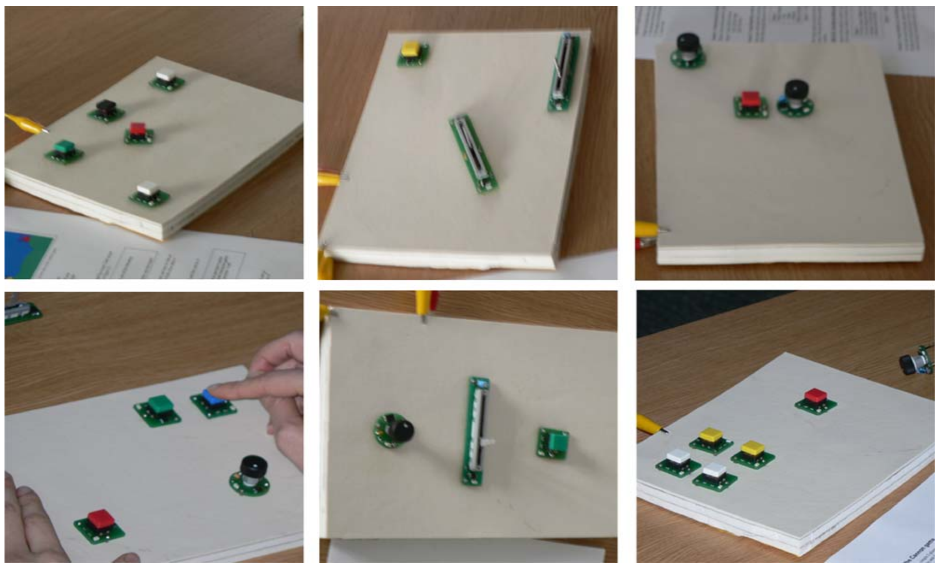


Figure 7.12: Some resulting layouts of controls on substrate pads.

There was a high variance in how the participants spatially laid out their controls on the substrate pads (c.f. Figure 7.12). Some participants lined up their angle, power and fire controls in sequence either horizontally or diagonally on the pad (c.f. Figure 7.13), thus matching the order they were taken through during set up (i.e. first selected the angle control followed by the power and finally the fire control).

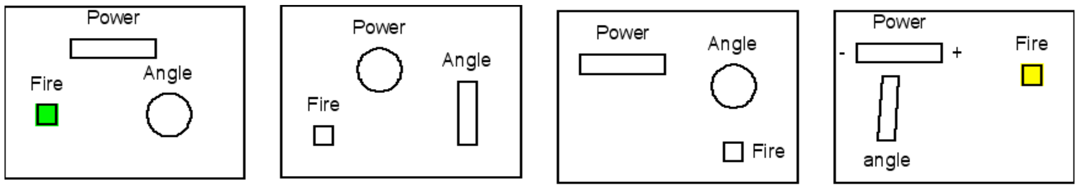


Figure 7.13: Sequential layout of controls.

Some participants laid out the controls so they matched the layout on the projected display (cf. Figure 7.14). One participant (far right in figure 7.14) put the slider at a 45 angle to match the on-screen orientation of the cannon gun barrel.

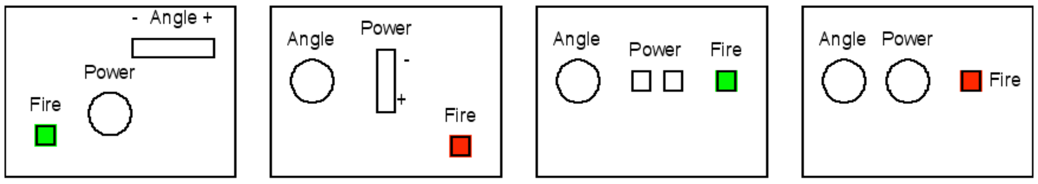


Figure 7.14: Layout of controls to match projected display.

Some participants liked having their angle and fire controls close together (bottom right in Figure 7.14), so once they decided on the amount of power after taking the wind speed into consideration, they then carefully adjusted the angle and hit the fire control.

However most participants preferred to have their fire control placed further away from the other controls, usually at the top or bottom right hand corner but a few did opt for the bottom left hand corner. This placement of the fire control was especially visible in Round 3 and any players who had their fire control placed in the centre of the pad in the second round did actually change its location in the third round.

Finally, in terms of the spatial layout between Round 2 and Round 3, 7 participants kept exactly the same functional layout, some using the same controls while others changing some of the controls. Eleven participants changed the layout of their controls in the third round.

7.2.3 Participant Feedback

The majority of the players, especially the non-gamers, preferred using the VoodooIO controls over the keyboard. Some of the features that the players liked include:

1. The ability to chose a control that is somehow evocative of its respective application function, i.e. the slider for the power which someone likened to "the tank gear box" and the dial to adjust the angle "as a knob". A player mentioned that this feature was very useful as "one did not have to think which keys to press for which function". Although one player mentioned that the slider worked equally well for setting angle, and the dial for setting the power, most preferred using them the other way round.

2. The ability to arrange the controls, which gave the players the opportunity to organise the controls sequentially, or in a manner that corresponded with the interface, or even arrange them in a way that suits one's preference, for instance, "how one wants to feel the control under one's fingers" or "so one can play with both hands"

3. The ability to spatially layout controls, which allowed players to separate out the different functions, place some controls closer together and divide the interface into individual control devices.

4. The ability and ease of moving the controls around or swapping the control direction during game play.

5. The choice of colours for the fire control, particularly the red fire button, which "had its own special place so one can get to it easily."

A few participants amongst the expert and casual players did prefer using the keyboard to manipulate the controls, mainly because they were more familiar with the keyboard and they felt that given the cannon game was based on turn taking, it was not really an inconvenience to share the keyboard. Also, they did not have to remember many keys to press as the cannon game only had three controls. However, they all agreed that VoodooIO “gave a nice set-up” and would be very useful in a game where players had to manipulate several controls in parallel.

7.2.4 Discussion

An observation of the use of the VoodooIO-setup over the keyboard-setup was the ability to split individual player control-clusters to form personal "game-pads". Even though it was a turn-based game, most participants were noticeably more comfortable having their dedicated points of control, resulting in less interference than was caused by sharing the same device. Thus the ability to spatially disperse controls across the surface of the desk was a beneficial property in this case.

In many cases participants did change their control composition between Rounds 2 and 3, and in some instances even modified the control layout during gameplay. This suggests that they were either not happy with their first choice/arrangement of controls, or were simply curious to try alternate configurations. It could be argued that, if the former reason was the cause for the change, this would argue against allowing users design their own control structure: an interface designer would not have made some of the mistakes in control design that users had to correct in the second round. This might be true, but the point can easily be countered by the fact that a bad choice was easily corrected in an effortless way, and users were quickly able to arrive at a working solution that they were personally satisfied with. It is not the one-time configurability that makes VoodooIO interesting, but the ability to continuously change and adapt the control structure over time, allowing a personally-ideal solution to be reached through trial and error. The second plausible reason for changing the control configuration – curiosity over alternate possibilities – could also be taken as a positive result. Encouraging exploration and corroboration of personal design ideas is clearly arguable as a beneficial property of any interactive system.

A further claim can be extrapolated from the results: in general, users make valid, conscious choices about the way their interface controls are presented. This is corroborated by the amount of overlap that occurred between different user's designs. The most clear example was the repeated use of "red" to signify the critical "fire" function (once pressed, there is no going back) and the separation of that control away from the other, less critical controls so that it would not be pressed accidentally. The fact that these seemingly sensible choices were observed repeatedly may have been biased by the fact that most participants came from a technical background, and would have been familiar with a variety of interface equipment (although not necessarily for playing games). Nonetheless, in this case the results suggest that users are pretty good interaction designers, and can be "trusted" with deciding what control configuration works well for them, and particularly if mistakes and bad choices are easy to correct.

There were some design constraints on what users could do with the controls – for example, they were not allowed to choose a "dial" to "fire" the cannon – and that potentially prevented participants from making really bad choices. The reason behind this was a practical one: there is no unambiguous way an analogue control can be mapped to a binary function. If users had been able to do this, some of them would have undoubtedly done so, and the result would have been unpredictable and likely confusing.

In conclusion, the insights gained from this experiment could be summed up as: users appreciate and benefit from being able to personally define the control structures that they are to use, but they need to be allowed to explore alternatives, make mistakes, and correct bad choices in order to arrive at whichever solution is best for them. In this sense, there is a strong case for the development flexible physical interfaces, which would bring exactly these sorts of capabilities to the end-user of technology. However, complete freedom is potentially confusing: practical and common-sense limitations should be applied to shield the user from making difficult decisions by balancing interface flexibility against possible interaction complexity. The following two sections explore these issues in more detail, through the application of VoodooIO to create flexible interfaces for music production and game-playing software.

7.3 Music Production

Users of music production software will be familiar with the frustration that often results from having to use a keyboard and mouse to control such applications. The problem stems from the fact that it is often necessary to perform several actions in quick succession, or even in parallel, in a way which is impossible to carry out on a time-multiplexed graphical interface (e.g. playing a chord across several notes, or moving a bank of volume faders simultaneously) [28].

As a result, some music software packages include MIDI protocol compatibility, which allows electronic musical equipment – ranging from piano keyboards to banks of generic dials, sliders and pedals – to be linked with application parameters. By using the MIDILink software (c.f. §6.3 "End-User Mapping Tools"), it is possible to use VoodooIO as a controller to any music production software or equipment that supports the MIDI protocol [92]. The results provides an interesting platform to explore interaction possibilities of VoodooIO in use with a conventional application, and with users who are expert at using that application with conventional interface equipment.

7.3.1 Reference Application

Reason, by Propellerhead software, is a mature music production suite that emulates a variety of different real-world audio devices. These devices may be connected with one another in a variety of different ways, and controlled with a realistic graphical interface that mimics the function of the physical devices. Hundreds of parameters may be controlled on-screen via the GUI (cf. Figure 7.15).

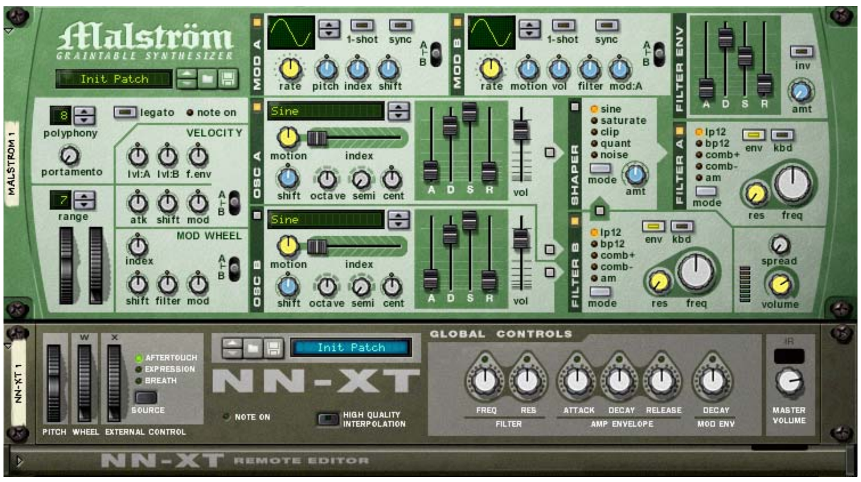
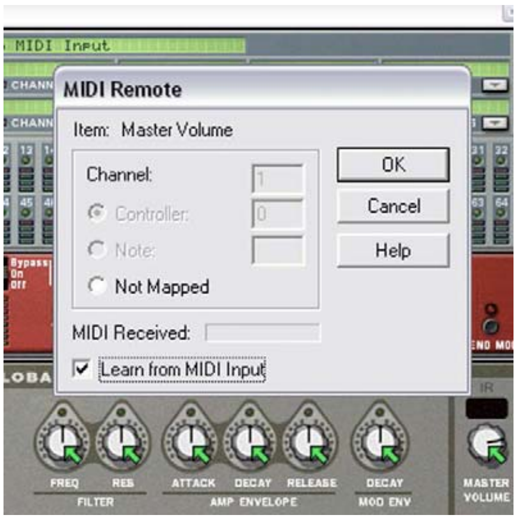


Figure 7.15: The Reason GUI emulates the look and feel of audio equipment hardware.

The program has a very simple way of linking a MIDI controller with an on-screen control: selecting "Edit MIDI Remote Mapping" from the "Options" menu causes all controls to which a MIDI controller can be assigned to be highlighted with green arrow. Clicking on any of these controls causes the window in Figure 7.16 to appear. At this time, the user can manipulate any control from MIDI controller device to associate it with the relevant on-screen control. Selecting "OK" completes the linking process and the physical control is ready for use.



MIDI Remote

Item: Master Volume

Channel: 1

Controller: 0

Note:

Not Mapped

MIDI Received:

[✓] Learn from MIDI Input

Figure 7.16: Reason's "MIDI Remote" mapping window.

The VoodooIO extension allows a user to generate spatially-distributed, custom control configurations to interact with the Reason software, which can then be controlled by manipulating physical buttons, sliders and dials. Controls can be freely arranged, labelled or removed from the physical interface area, and re-mapped via the graphical interface (cf. Figure 7.17). The user can effectively pick functions of the screen, and place direct physical shortcuts to them on the desk.

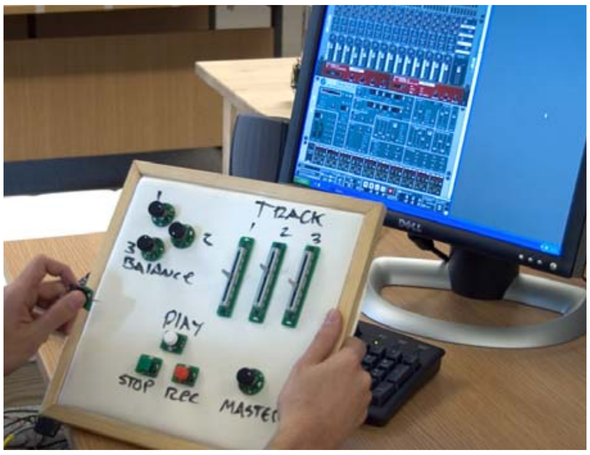


Figure 7.17: Reason extended with a VoodooIO interface.

The solution extends the current working practice, rather than replacing it. By positioning and orienting controls on the substrate, a user is able to effortlessly create an operational mixing desk that is specifically tailored to a session or piece of music: the complex graphical interface is reduced to no more and no less than those parameters that are necessary or desirable.

7.3.2 Evaluation

The extended interface was explored in collaboration with three expert users of the Reason package, who were recruited for the purpose. It was arranged that each user would try out the interface in an extended evening session in their home, to observe how they work with Reason in their accustomed environment. First, users were asked to explain and demonstrate their use of Reason in general terms, and in a detailed walkthrough of a session reproducing some of their recent work. This was followed by a brief introduction to the physical extension of Reason on equipment brought to the meeting. The remainder of the sessions were spend with users exploring the use of the interface, trying out their own ideas, and engaging in joint discussion of interaction possibilities.

The observation and discussion of Reason use showed that only a small fraction of the available controls is used in any particular session. All three users were genuinely excited about the possibility of "taking exactly what you need off the screen" and to "have the rest of the interface fade into the background". They explained that other control devices are available for use with music software (e.g. fader banks), but that these were rigid in structure and would not allow adaptation of the interface to the task as easily as demonstrated with our prototype.

The users were also very creative in exploring possibilities enabled by the soft-wired interface beyond the creation mere physical shortcuts. One user found that the could very elegantly support cross-fading between tracks with slider controls, by simply turning one of the sliders upside down, as shown in Figure 7.18. Another user played with the idea of connecting two controls, a button as MIDI Note and a slider as MIDI Controller, to the same function in the Reason interface. This had the effect that an otherwise scalar control could alternatively be turned from “off” to full volume by a simple button press. Another idea produced was to utilise the device pins to hold paper tags for the purpose of ad hoc labelling of their function.

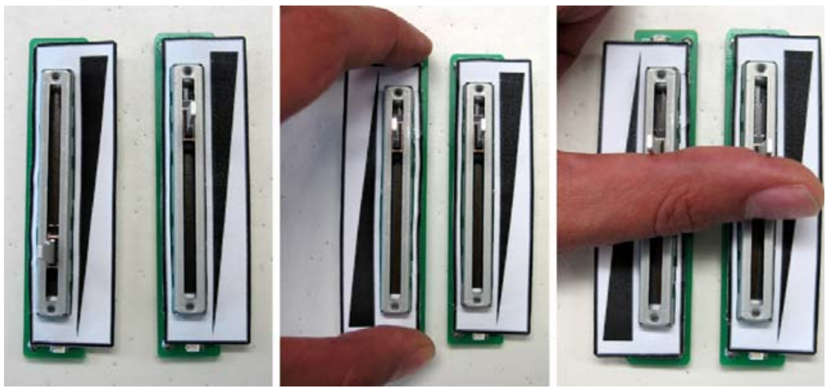


Figure 7.18: Creating an ad hoc crossfader.

An immediate advantage of the physical extension to the graphical interface is that cumbersome controls such as on-screen dials can be made physical, and that interaction can be streamlined through physical shortcuts. Moreover, task conformance can be supported through ad hoc assembly and convenient arrangement of the appropriate subset of controls, with the potential of reducing the cognitive load by having the original and more feature-rich interface retreating into the background. Also noteworthy is that the physicality of interface components facilitates techniques such as the crossfading sliders. This technique exploits meaningful spatial arrangement of devices despite the underlying system having no awareness of control location and orientation.

7.3.3 Discussion

Many compelling examples of novel physical interface designs can be found in the domain of music control, creation and performance as in the Audiopad [70], reacTable [48], Block Jam [64] and the Squeezables projects [94]. This is not surprising, since the alternative interaction modalities introduced by novel user interface designs can often be more useful, intuitive or enjoyable to a musician than the traditional keyboard and mouse. Other related work includes projects that provide generic mechanisms for associating physical input devices with graphical user interface elements on the fly, such as the Phidgets WidgetTap [38] and the ICON input configuration system [23] .

The experimental musician and researcher John Bowers has used the term *performance ecology* to describe the arena for activity created by a musician in his immidiate surroundings. In his work regarding the ethnographically-informed design of improvising machines [8], he discusses the importance of the spacial arrangment of control devices and instruments - not only in allowing the musician to be more effective in his performance, but also in communicating his intentions to co-performers and audience. The ability to effectively organize the layout of these ecologies is a recurring theme throughout his work.

In the theoretical paper "Towards a Musician's Cockpit" Vertegaal et al. [86] make the case for the importance of customisation in musical systems. Instruments should be physically fexible to adapt to the changing needs of a performer, but this functionality should also be "freezable" to allow the system to be properly internalized and learnt. The theory has been applied in the development of the SensOrg system [84], which consists of an arrangement of physical modular devices and music software that provide a high degree of customization and adaptation, in order to increase the level of usability in an electronic musical system by making it sensitive to an individual's cognitive and ergonomic requirements. A particularly interesting feature is the the Flexipad component of the SensOrg, which allows a number of buttons and faders to be positioned and oriented on a metal pad to comfortably fit the position and size of the performer's hand.

Using the VoodooIO platform for flexible physical interfaces, we set out to design and build an interface to improve the usability and experience of a musician working with existing music software. In particular, we have given the user the ability to tailor their physical performance and composition environment. The VoodooIO interface sits amongst existing interface devices, and provides a complementary channel of control alongside the traditional graphical interface of the application. We imagine that VoodooIO could be easily incorporated into the existing working environment of an electronic musician, and it will be interesting to see how the interface is used in extended practice: the inherent flexibility will almost certainly allow it to be appropriated and used in ways we have not thought of.

7.4 Playing Games

Computer games make use of some of the most sophisticated user interfaces and interface devices of any application. Certain games, such as flight or driving simulators, are designed to be played with highly bespoke interfaces. Off the shelf hardware components – such as pedals, steering wheels and throttle controls – are available for assembling cockpits and other custom game-playing environments.

The need to support individual game-player preference is evident in degree of interface configurability that can be found in the configuration panel of many games, such as the ability to re-map game actions to user-assigned keys. There is also a growing trend towards physically configurable to adapt functionally and ergonomically to a users' preference, such as the Ergodex DX1 input system [24] that allows users to arrange keys on a tablet, or the Cyborg gamepad that includes a swappable joystick/keypad module [76].

The variety of desirable interface configurations available for different game-playing situations, coupled with an existing trend to make game interfaces user-configurable, makes gaming an interesting area in which to explore the applicability of VoodooIO. To this end, we set ourselves the task of generating and using two separate game interfaces using a VoodooIO kit and the InputMapper tool (c.f. §6.3 End-User Mapping Tools). We report on the results of these two experiences below, and use them as basis for further discussion.

7.4.1 Building a Mech Cockpit

In MechWarrior the player is in control of a large battle robot, known as a 'mech'. The game's interface is similar to that of a flight simulator's. The player is presented with a first-person view of a pilot sitting in a mech cockpit, with the screen replicating a head-up-display on which navigation and status information is overlaid. Most of the gameplay centers on the piloting of the mech, steering it across the 3D environment around enemy units and other obstacles. Other important functions deal with speed control and the use of weapons, and in total there are dozens of separate parameters that the user has access to. By default most of these functions are mapped to the keyboard, but it is also common for this game to be played with an additional joystick for steering control, and with the most essential game functions mapped to any additional programmable buttons on the device, and remaining functions mapped onto keyboard keys.

As starting point for the exercise we tried to replicate an average gaming setup: an office chair and desk, with a 17" flat-screen monitor, keyboard, mouse and a Logitech Wingman force-feedback joystick. This particular joystick includes seven programmable buttons, eight-way hat switch and throttle lever.

In thinking about how this interface could be extended with VoodooIO, an early decision was to do away with the keyboard. It provided a large number of buttons onto which most of the game's many functions could be mapped. However, the fixed layout and static labelling also made it difficult to remember what the function (if any) of each key was. This was mainly due to the lack of any visual prompt or mnemonic to act as a reminder of specified key bindings. The keyboard itself took up a lot of space, competing for desk area with the joystick, which we wanted to use as our primary input device in the middle of the desk.

The first goal in extending the interface was to provide sufficient controls for all the necessary game functions, without using the keyboard and aiming to make it more comfortable and easier to use. The second goal was to try and make the gaming space more immersive by making it feel more like a mech-cockpit than an office desk.

The process of construction began by considering any available area of the gaming space that could be useful as a control surface. The main concern was ergonomic, mainly considering unused surfaces that were easily accessible while sitting in the chair. At the same time, we considered any area that we felt could be an interesting place on which to arrange controls and improve the look and feeling of being in a cockpit.

In the end we settled on four different areas to augment: desk space to the left and right of the joystick, the lower bezel of the monitor frame, and the left armrest of the chair. The two desk areas we chose for their ready accessibility and efficient use of the space previously taken up by the keyboard, allowing controls to be arranged around the joystick, and without displacing it from its central position on the desk. In selecting the monitor bezel we had in mind that through its proximity to the simulated head-up-display (HUD), it could be used as an appropriate place to arrange controls related to the visualization settings of the HUD. The armrest was chosen because we felt it would reinforce the feeling of sitting in a cockpit, with controls to the side as well as in front of the player.

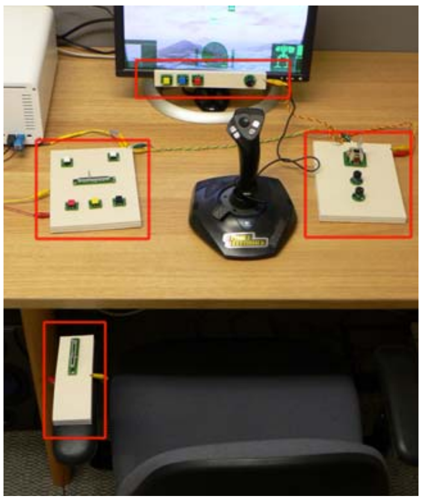


Figure 7.19: Cockpit design - notice how interface areas extend beyond the desktop to the monitor bezel and chair armrest.

From a single sheet of substrate material, four different pieces were cut to measure and then networked together using interconnects of appropriate lengths (c.f. Figure 7.19). The monitor and chair pieces were affixed to their designated surfaces to hold them in place, while the desk pieces were left unfixed and able to be moved freely across the desktop. On the left substrate we arranged controls dealing with the power and weapon systems. Different coloured buttons allowed the mech to be turned on or shutdown, while a horizontally placed slider allows selection between different firing modes. The right desk substrate contains a small joystick to modify the direction of view, allowing the player to look towards the back, front, left and right of the mech. On the monitor substrate we placed buttons to modify the HUD settings, toggling between different levels of information overlay. Finally, the armrest substrate was reserved for a single slider, which was used as a throttle for speed control. In the end, we added a few additional controls, without any predefined functionality, but simply intended to strengthen the effect of sitting in a cockpit and being surrounded by controls.

The InputMapper software was configured to generate key-down events in response to Button controls being pressed. This allowed seamless mapping of buttons to functions through the game's key-bindings configuration screen. In order to incorporate some of the analog controls, it was necessary to specify simple mappings that simulated different key-combinations from the current state of the control. For example, selecting the speed was, by default, set by pressing the numbers 1 through 9 on the keyboard. The continuous output of the analog slider was then re-interpreted by the mapping as nine discrete steps, each triggering the appropriate key-down event.

7.4.2 Example Scenario: Arranging Character Abilities

World of Warcraft (WoW) is a massively multiplayer online role-playing game, where each player is in control of a character in a shared 3D world. As with most games of this genre, the aim of the game is focused on the development of the character, which is advanced in level through the accumulation of experience points. As the character's level increases, it is able to learn new abilities and skills. What abilities are available to a player depends not only on the current level of the character, but also on decisions which a player makes during the process of initially defining and then gradually developing their character. A character may be initially created as being from one of several available 'races', each contributing certain 'innate' skills to the character. A player must further specialize their by selecting a 'class'. For example, a player may have selected a Hunter class, in which case the character will be eligible to develop skills relating to the use of hunting weapons, the practice of setting traps and the ability to tame beasts. If, instead, the character is a Mage, as its level increases it has the opportunity to gradually learn how to use increasingly powerful spells. Even within each class there are opportunities for further specialization, to the degree where it is rare for two characters to have exactly the same abilities and strengths. Furthermore, the way in which a player may chose to actually use those abilities, or apply them in a particular situation, is of course a matter of personal preference and will vary widely from player to player.

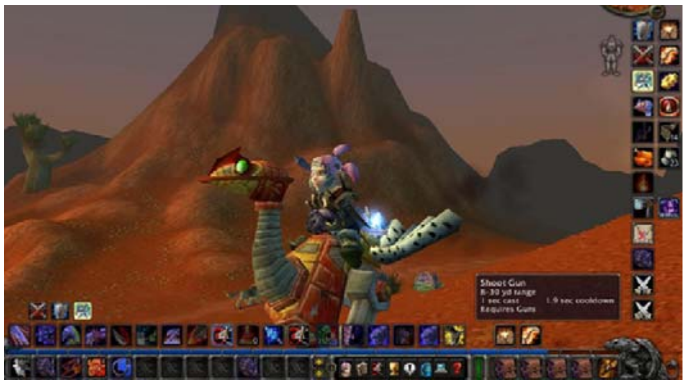


Figure 7.20: An example arrangement of abilities and skills around the edges of the World of Warcraft GUI.

The point that we are trying to convey is that, in this example, the gaming situation is not only unique to every player, but also subject to change over time as their character develops and gains new skills. As such, the interface to control the character must also be player-configurable, and adaptable throughout the course of the game. Particularly in combat situations, it is important to use the abilities effectively – in a timely manner but also in particular orders.

This fact is clearly reflected in the design of WoW graphical user interface: around the edges of the screen are toolbars with a set number of slots where icons, representing the various character abilities, can be dragged into and freely arranged as they become available (c.f. Figure 7.20). These abilities can then be accessed via the toolbar icons by clicking directly on them, or triggered by shortcuts on the keyboard.

The keyboard and mouse are very appropriate gaming controllers for this particular game, as the mouse provides comfortable steering of the character around the world, and the keyboard is perfectly suited for typing text into the game console, which is often used to communicate with other players. So the aim in this exercise was to create an additional control area on which to factor out direct control of individual abilities as they are introduced throughout the game, and allow a space on which to arrange and label those controls in meaningful ways.

The unused desk area between the keyboard and monitor was augmented as a VoodooIO substrate area. When shaping the substrate, we cut it in such a way as to shape it with a slight concave curvature along its lower edge, in order to perfectly accommodate the convex upper edge of the keyboard. This a small detail, but it allowed us to appropriate valuable desk space which would otherwise be wasted, and gave a sense of extending the keyboard rather than introducing an additional input device.

As an additional feature, we placed a plain sheet of paper over the substrate. The intention was to allow a way to label and annotate the arrangement of controls once they were attached in place. The pin-like connectors are easily able to penetrate the paper, and fasten correctly to the substrate underneath.

In our exercise, we used a character of the Hunter class, meaning that the abilities it had accumulated up to that point were mostly related with the use of traps and of long-range weapons. From experience, we developed a particular sequence in which these abilities should be used in the process of hunting (e.g. setting a trap is only permitted before entering into combat). We arranged a number of button controls in such a way as to visually represent our chosen sequence of actions, and labeled them accordingly (c.f. Figure 7.21).

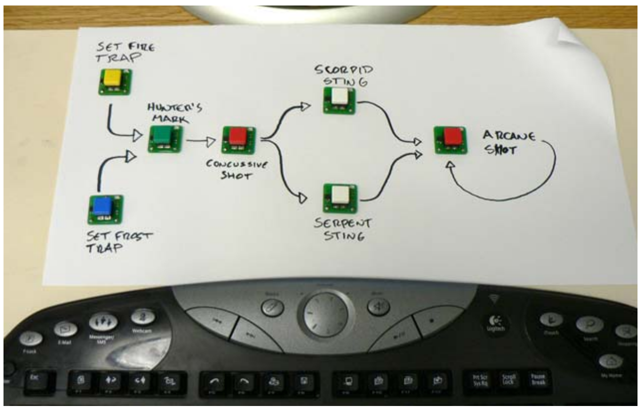


Figure 7.21: Controls are arranged to depict the intended use-sequence. Note the use of a paper sheet between controls and substrate to label their function and annotate the use sequence.

In this arrangement, the player begins the hunt by pressing the button to trigger one of two mutually exclusive tasks: setting down a 'fire' or 'frost' trap. The next two steps always follow each other, the application of a 'hunter's mark' to the target, followed by a concussive shot. These two abilities will only be used once, at the beginning of the combat, and the sequence of arrows from one to the other reflects this. The next step is to select between another two mutually exclusive abilities – applying a 'scorpid sting' or 'serpent sting'. Which ability is used depends on the particular type of prey being hunted, and the respective buttons are laid out and labeled to reflect this choice. The final step in the process is the use of the 'arcane shot.' This ability, in difference to the others, will repeatedly be used for the remaining duration of the hunt. An arrow from the control and doubling back onto itself has been drawn on the paper to illustrate this.

7.4.3 Discussion

The ability to spatially lay out different types of devices in a meaningful way contributed to the legibility of the interface's functionality, making it easier to remember the use of different controls by their different types, colours and locations. Additionally, the way in which different sections of the furniture and equipment were incorporated into the design made for a more immersive use of the space. In the first exercise – and even though the joystick had a perfectly suitable throttle control built onto its base for the purpose – informal user-testers particularly enjoyed controlling the speed of the mech via the armrest-mounted slider. In the second example, the being able to freely annotate the surface using the paper insert contributed greatly towards using the substrate a legible element of the interface, which exactly reflected the particular character's abilities and player preferences in using them.

The World of Warcraft example illustrates the advantages of possible continuos adaptation to reflect changing game conditions. In this case, we imagine that throughout the course of the game a player would, from time to time, make gradual changes to the setup. They may, for example, find that original assumptions about a comfortable arrangement might prove uncomfortable after extended use. The ability to adjust the layout of controls on the fly would prove useful in this situation. If the interface is shared, individual 'cockpit' users and may want to make changes to the interface to suit their preferences whenever it is their turn to play.

For both games, the combination of VoodooIO and the in-game keyboard mapping system supported the process of adding new buttons as new abilities become available. Specifying an additional control consisted simply of inserting a new control onto the substrate, and associating it with the new ability through the key-bindings menu. In this point, the physical interface highly resembled the GUI's support for tweaking interface elements during the course of the game, with the added benefit that these icon-based 'shortcuts' could be factored out of the graphical interface, liberating valuable screen real-estate, by being made accessible through dedicated physical control. In both cases, we believe that the use of VoodooIO contributed to the creation of game-playing environments that were uniquely suitable to the game's interface requirements, as well as being tailorable to a high degree to an individual player's preference.

7.5 Study of Control Presence as an Interaction Technique

The underlying VoodooIO implementation has the distinguishing characteristic of supporting fast control presence detection and unique identification. The physical design makes adding and removing control from the substrate material almost as easy as manipulating them. From an interaction perspective, the question is: can the ability to quickly detect control presence be useful as an interaction technique in its own right? Attachment/removal of controls is a very explicit action already charged with a certain amount of expressiveness, and suggests that detection of control presence could be used as an interaction technique akin to control manipulation. For example, a particular control could be used to physically encapsulate application concepts (e.g. as in [48] and [70]): when a particular control is attached, its associated concept is activated; manipulating the control modifies parameters associated with the concept, and removing the control deactivates it.

In order to test this concept, an experiment was designed which compared different ways to control a software drum machine. The drum machine involves a small set of percussion instruments: a snare, base drum, cymbal and shaker. Each of these can be turned on and off separately, and adjusted in their speed. Figure 4.9 captures the experimental setup with a VoodooIO substrate overlaid a desk surface, a set of controls, and speakers for output.

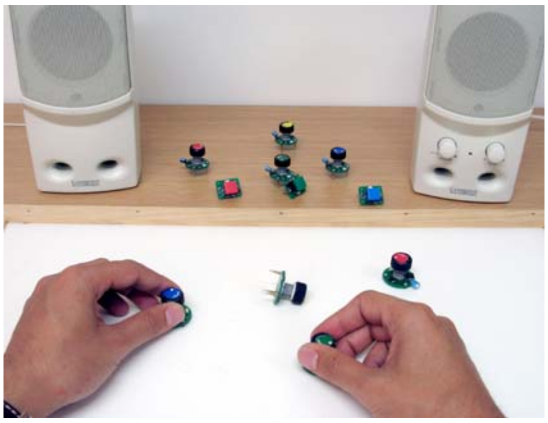


Figure 7.22: Experimental setup: participants interact with a software percussion machine.

7.5.1 Experimental Setup

The application was used for a comparative study of three interface conditions, shown in Figure 7.16. In all conditions, the percussion instruments were associated with a dial to control their playback speed; each dial was clearly labelled with its corresponding instrument. Across the conditions, the LED beacon on each of the dial controls was used to provide localised feedback on whether the instrument was "on" or "off."

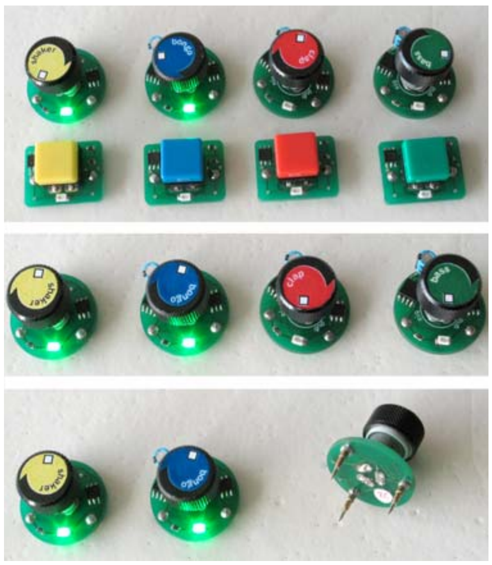


Figure 7.23: Experimental conditions: "Button," "Dial," and "Presence."

Each condition differed in mapping of "on/off" to physical controls: In the first condition, this was mapped to an additional button control, in the second it was overlaid on the rotating dial (i.e. "off" mapped to "zero speed"), and in the third it was mapped to control presence (i.e. instruments turned on by insertion of the associated control on the substrate). These conditions will be referred to as Button, Dial and Presence, respectively.

The study was conducted with 20 participants, including 10 men and 10 women, aged 16 to 48, mostly very familiar with computers, but with varying background in music. All users engaged in test of all three conditions but in varying order. For each condition they received a brief introduction and were then asked to perform a series of small tasks, such as creating a rhythm with two instruments at the same speed, followed by a rating of the interface on a Likert scale. After completion of all three conditions they were asked to rank the conditions for various criteria, and to comment more generally on what they liked or disliked with particular conditions. The test sessions were also video-taped and transcribed to collect anecdotal remarks.

7.5.2 Experiment Results

Figures 4.11 and 4.12 summarise the quantitative results. The Button condition, which is also the most common in comparable to regular audio hardware, and the Presence condition consistently rated higher than the Dial condition which can be attributed to the Dial prohibiting separate control of on/off and speed.

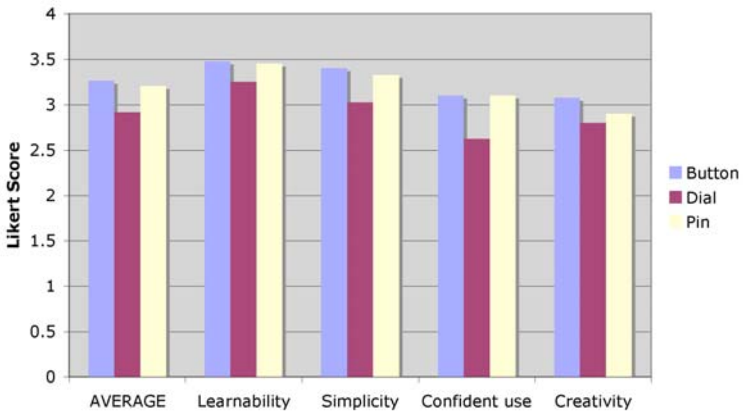


Figure 4.11. Likert scores for usability of the Button, Dial and Presence conditions.

It is noteworthy that device insertion and removal are evidently seen as equally usable for direct control of an application parameter as more conventional means. The cumulative ranking scores provide more detail on relative preferences. Overall, the Button condition is seen as best suited for the task and in particular providing best control. The Presence condition is rated relatively low on task suitability, but valued as easiest to understand, most confident to use and causing least mistakes. This is also backed up by user comments hinting at the simplicity and transparency as particular advantages of this condition. One user drew a parallel to WYSIWYG, pointing out "what you see is what you hear".

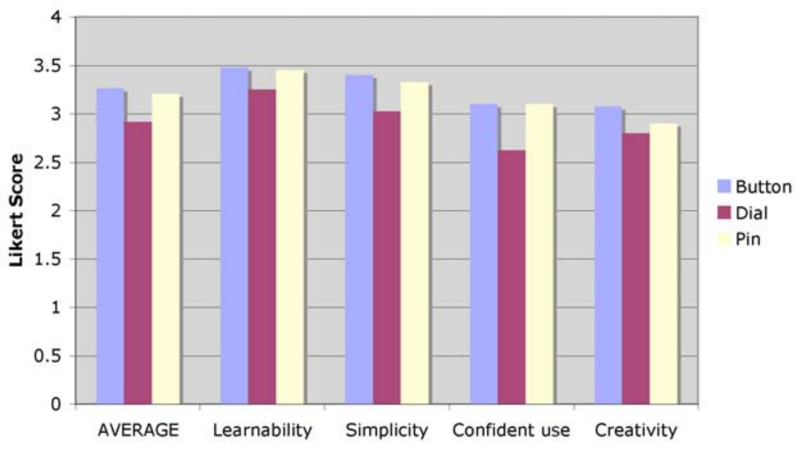


Figure 4.12. Ranking scores for the tested conditions.

7.5.3 Discussion

The results of the experiment are interesting: the fact that, in terms of usability, adding/removing controls to turn on/off application parameters was widely accepted as an interaction technique is a powerful result. This concurs with some of the findings from tangible user interface research, which establishes the practice of having tangible objects represent abstract functionality or data [26]. In the case of VoodooIO, each "object" also comes with a dedicated control for the concept it represents, and which can intuitively be used to manipulate a parameter associated with that concept. The results suggest that users can equate the act of removing a control to disables its associated concept from the application.

The VoodooIO design provides a clear sign of when a control is "in play" or "out of play" – there is little ambiguity about this, as the two states of being attached or detached from the surface are explicit and mutually exclusive. This is validated by the user feedback, where "Presence" condition scored so highly in the "Most confident to use," "Caused least mistakes," and "Easiest to understand" categories. One user even commented on the pleasure she gained from attaching controls from the substrate, making the analogy to "walking on snow." However, the effort of taking controls in and out of the structure does require more effort than is necessary to push a button, and for this reason the "Button" condition probably won over, seemingly better suited for an action which users were asked to perform repeatedly over the course of the study.

To sum up the most interesting results, and their applicability to the design of flexible user interfaces: the VoodooIO interface allows users to attach and detach physical controls to and from substrate areas; the system is able to quickly detect changes in control presence, and the result of our study show that users can equate adding/removing controls as an interaction technique akin to control manipulation. The possibilities for user interface design include the use of physical device attachment and detachment as expressions in the dialogue between user and application, dynamic mapping of virtual interface elements to physical ones, and ad hoc creation of composed devices. The following section explores these interaction techniques in more detail, by using control presence as a central concept in the design of interfaces for musical performance.

7.6 Ad Hoc Musical Instruments

Ad hoc musical instruments are, in some significant way, constructed during the course of interacting with them: by definition, an ad hoc instrument is made and played at the same time [10]. By interleaving performance with the fabrication of the instrument itself, one can explore extended possibilities for music performance. In Section 7.5, a simple ad hoc instrument was used to evaluate user acceptance of using control presence as an interaction modality, which allowed percussion elements to be brought in and out of play by adding and removing associated controls. In this section, we take the idea further and present two designs for more sophisticated ad hoc instruments to demonstrate the possibilities of this interaction technique.

Our concept of building an instrument while playing it has much in common with the philosophy of live coding. Indeed, Wang et al. [93] discuss how, through the process writing music-synthesis code in live performances, one can dynamically reconfigure controller mappings. The work reports on successful experiments with commercially available control surfaces. However, one can go further and reverse this picture: the creation of an ad hoc control surface could become the means by which live coding takes place.

Our work explores the utility of a particular approach to building dynamically configurable interfaces for musical purposes, thereby making a particular kind of hybrid (software/hardware) ad hoc instrument. Research on physical interfaces that involve ad hoc composition and customization includes the Behavior Construction Kits [74], Electronic Blocks [97] and Triangles [36], which support a variety of functionalities (behaviour animation, information access, simple programming) through exploration of variable interface configurations sometimes in a playful fashion. Commercial products enabling the ad hoc construction of interfaces and control surfaces are beginning to emerge. For example, the Ergodex input system [24] provides a way to easily arrange a number of buttons on a tablet to suit the user's ergonomic preferences.

From time to time, authors in the New Interfaces for Musical Expression (NIME) conference and allied research communities have explored instruments and other artefacts which manifest a degree of ad hocery. For example, the Flexipad element in Vertegaal and Ungvary's SensOrg [84] allows controls to be arranged on a metallic plate. However, Sensorg supports a somewhat limited number of controls which are all hardwired, constraining the ease with which controls can be added to the ensemble and freely moved. As with Ergodex, the motivation seems mainly ergonomic (e.g. allowing varied arrangements for ease of manipulation) rather than to explore a more extended utility of ad hoc interfaces for musical interaction.

BlockJam enables users to arrange blocks to construct musical structures [64] while Audiopad [70] also affords a level of ad hoc interaction, allowing the performer to compose by manipulating the arrangement of tokens on a surface. ReacTable [48] and the Round Table [44] involve the manipulation of physical artifacts on a table surface and have both found application in supporting collaborative performances from multiple musicians. Bowers and Archer [9] entertain the possibility that the juxtaposition of incomplete or half-made 'infra-instruments' could be a viable form of performance. Like that paper, we are concerned with ways of reformulating the instrument design 'life-cycle', in our case making means for dynamically configuring interaction surfaces available in/as performance.

7.6.1 An Ad Hoc Mixer

We used the MIDILink tool (c.f §6.3, End-User Mapping Tools) to couple VoodooIO with the MAX/MSP audio synthesis software to build a pair of ad hoc instruments: an ad hoc mixer and a synthesiser (Figure 7.24).

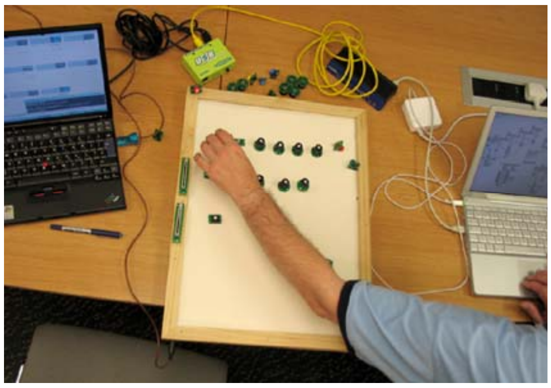


Figure 7.24: The MIDILink tool was used to forward VoodooIO events via MIDI interface to MAX/MSP, which ran on a separate machine.

Both of these allow one to incrementally build a performance interface without interrupting the music. In this way, we intend that the construction of an interface becomes part of the gestural repertoire of a performer. Furthermore, as we shall see, many of the background enabling actions that performers commonly have to do in advance of interaction (e.g. load patches or choose presets) can be elegantly folded in to their interface building and performance activities.

The mixer allows the performer to build a mixer interface interleaved with the performance of a mix. Up to four stereo soundfiles can be played back with amplitude control by means of slider controls. In addition, each mixer 'channel' has associated with it a resonant filter, the centre frequency of which can be set with a dial control. A typical interaction with the mixer might proceed as follows.

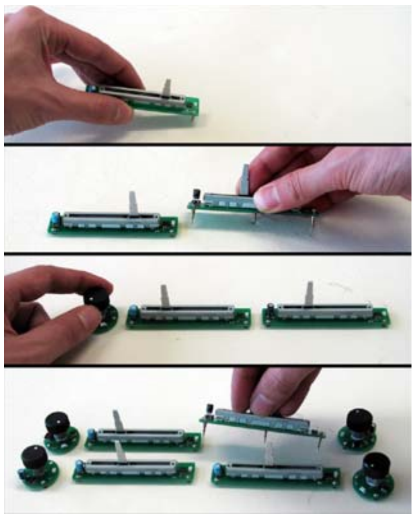


Figure 7.25: An example performance sequence using the mixer.

Placing a slider on the substrate enables mixer Channel 1 and identifies a soundfile to be played back. The initial default amplitude is zero but this is supplanted when the slider is manipulated (Figure 7.25, top). The performer might, for example, raise the amplitude of the soundfile to moderate levels before inserting a second slider into the substrate. This would enable mixer Channel 2 and identify a second soundfile for playback (Figure 7.25, middle-top). Having adjusted the amplitude level to taste, the performer may then wish to filter Channel 1 (Figure 7.25, middle-bottom). A third mixer channel could then be created and so forth. Imagine now that the performer has created four mix channels, each with its own resonant filtering, and the music has reached its most dense moments. The performer may then wish to thin out the mix. Removing one of the sliders will stop the playback of the associated file (Figure 7.25, bottom). Removing a dial associated with a channel that is still playing will remove the effects of the resonant filter. A performance might be completed by taking the interface down to just one slider with its associated soundfile playing out. The performer could at any moment cut to silence by removing the last slider.

7.6.2 An Ad Hoc Synthesiser

While we became aware that our mixer exhibited certain ordering constraints in the behaviours it was capable of, we wondered whether these could also be exploited in interesting ways. Our second instrument, the synthesiser, demonstrates how a synthesis patch could be interacted with by means of incrementally building the interface to it. In contrast to the mixer, though, the order in which interface widgets are added is 'parsed' so as to further inform how the synthesizer should be configured. Placing a slider on an empty substrate makes available a sine wave oscillator with default amplitude and frequency. Manipulating the slider gives amplitude control. The next dial to be placed on the substrate will give frequency control for that oscillator (sweepable through the MIDI note number range). The next three dials will control phase modulation of the oscillator with the first controlling modulation depth, the next the frequency of a first modulator, and the last controlling the frequency of a second modulator which phase modulates the first (Figure 7.26, top). In this way, a synthesizer can be configured which has a single audible sine wave oscillator with two modulators cascading phase modulation. Placing a second slider on the substrate makes available a second 'voice' which can be incrementally added to in the same fashion as the first to create another cascade of phase modulation (Figure 7.26, middle).

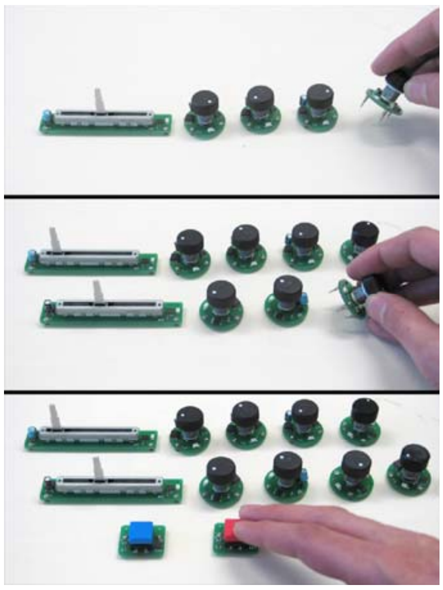


Figure 7.26: Performing with the synthesiser.

In our synthesiser, the exact significance of a control depends (at least for the dials) on where in order they appear on the substrate. The first dial is a frequency control. Subsequent ones control various aspects of phase modulation. This contrasts with our mixer where a dial was always associated with the centre frequency of a resonant filter.

Our synthesizer can be completed by adding up to two buttons (Figure 7.26, bottom). These have fixed frequency oscillators associated with them. Pressing the button alternately turns the oscillator on and off. The time intervals between presses of the button are measured and used to automatically pulse the oscillator. This rhythm can be interrupted at any time and reset though three (or more) successive manual button presses, or the pulsing can be stopped altogether by removing the button from the substrate.

Altogether then, the synthesiser has (up to) two frequency-variable oscillators which can be complexly phase modulated and two fixed frequency oscillators whose pulsing behaviour can be manually shaped. While this is a simple synthesizer, it is nevertheless capable of a variety of pulsing, rhythmic effects in a retro sort of fashion. The important point, however, is that it demonstrates how we can use VoodooIO technology to interface to synthesis, building interfaces as we configure the topology of synthesis units and do all that without interrupting the music. Furthermore, our synthesizer shows how we can, in rudimentary ways, 'parse' the interface building activity of the performer to make more varied assignments between interface elements and their underlying function.

7.6.3 Performance Experience

We have demonstrated our mixer and synthesiser on a number of occasions. In particular, the author performed as part of Circuits of Malpractice, a concert of new performance and installation work at the School of Music, University of East Anglia, Norwich, UK, 3rd October 2005 (c.f. Figure 7.27).

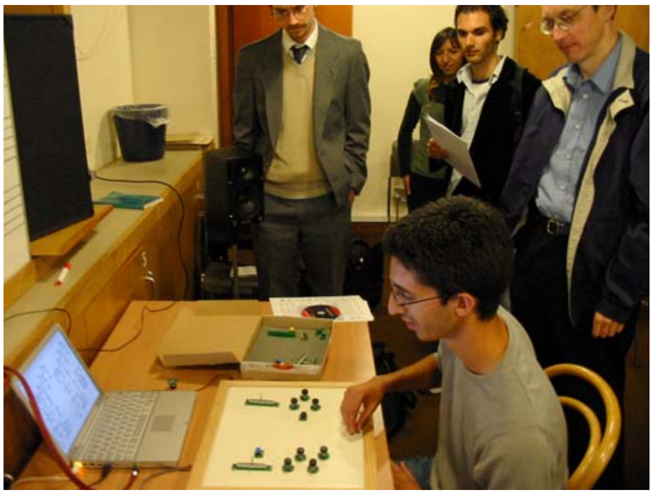


Figure 7.27: Ad hoc instrument performance at Circuits of Malpractice (2005).

In discussions with people that attended the performance we gathered that they enjoyed watching the process of "construction". All of the actions involving placement of controls on the substrate were functional even if their significance was not immediately revealed (e.g. a slider would only be heard to control amplitude when it was moved). This gave the performance a kind of subtle legibility. Even if the exact mappings between action and effect were not immediately transparent, it was clear that the instrument was gradually, as one audience member put it, "written down on the blank sheet" as the piece grew and developed. It was interesting to find that, even though the underlying implementation was imagined to be complex or "very clever", the actual operation of the interface hardly needed explaining. Our audience members recognised the basic controls in our set and what can be done with them: knobs are for twisting, sliders are for sliding, buttons are for pressing. Sharp pins on the bottom of the controls and the soft, membrane-like rubberised substrate provide a strong sign of how the two can be used together. Amongst some musicians in our audience, the fact that the actual programming of the instrument is done in Max/MSP gave a sense that this was a technology that they could use and appropriate. It also encouraged speculation about how our applications could be tweaked and modified in their behaviour. The aesthetics of the components was appreciated—the mixture of electronics and mechanics involved.

7.6.4 Discussion

Much of the point of building new interfaces (or instruments) for musical expression is to enable a musician's performance activity to made legible for an audience in new ways. This is sometimes seen as especially important for forms of music which might otherwise be difficult to follow. Our work with VoodooIO adds to this concern in an interesting way. As a mixer or synthesiser is being built in performance, in front of the performer and those audience members within perceptual range are just those interface elements which are needed for interacting with the music. In this sense, we have made use of control presence to enable a rich and meaningful interaction technique: in contrast to using a subset of the sliders or dials on a conventional controller and leaving some idle, the emerging and changing complexity of the interface, over the course of a performance, parallels and helps illustrate the complexity of the ongoing music. There are no surplus controls to distract the performer or to enigmatically be left untouched. A performer's 'latitude for action' (variation, choice) is clearly displayed in terms of the available controls: not just what you are doing but what you can immediately turn to. Bringing out a new controls presages an imminent change in the music, helping the audience anticipate and listen for transitions. The coming and going of dials and sliders can give a literal impression of the overall arc of the music and the return of the substrate to emptiness (and silence) ending a performance has been found aesthetically pleasing by many of our audience members.

It is a common complaint of contemporary digital music that watching a performer hunched over a laptop and a MIDI control box is a boring affair. In many ways, our task has been the re-enchantment of dials, sliders and buttons, to return a degree of fascination and intrigue to their use. By regarding these as elements to be worked with constructing an ad hoc instrument in the time of performance itself, we feel we have gone some way towards achieving this. The act of building the interface draws attention to the organization of the music and its relation to performer-activity, matters which are hidden in much conventional technology-rich performance. While we have explored just one way of making ad hoc instruments, we hope we have shown how this concept might help engender some innovative approaches to music making.

7.7 Analysis

The individual discussion sections found throughout this chapter provide snapshots into the novel interaction techniques that are enabled by VoodooIO. This section provides a final discussion that identifies common themes across the various experiences, highlights user benefits, and generalises insights into the role of physical flexibility in user interfaces. The usability principles set out by Dix et al. serve as a framework from which to structure the analysis; these principles are identified in [22] as three broad concerns of good user interface design practice:

Learnability: How easy is it to attain effective use of the system?

Flexibility: How much scope is there for interacting in multiple and different ways?

Robustness: How well is the user supported in achieving their goals?

The focus in the design of VoodooIO is to provide users with increased interface flexibility for interaction with interactive systems, and as a result the discussion is centred on flexibility-related factors. Nonetheless, we also consider issues of learnability and robustness in order to provide a complete picture of the usability properties associated with VoodooIO, and to avoid its presentation as a ragged or incomplete design.

7.7.1 Learnability

Dix identifies several factors that influence the learnability of an interface – one of them is *familiarity*: the extent to which a user, on encountering the system for the first time, can recognise its purpose and/or intended form of operation based on past experience and previous knowledge. Familiarity is related to the discussion in Chapter 2 about the "Language of Controls", which allows users to recognise what a system is for and how to use it based on the structure and composition of its interface. Rather than introducing novel or abstract interface elements, VoodooIO controls are reminiscent of machine-like interfaces and recognisable interface transducers such as buttons, sliders and knobs. In our experience, this design ethos has proved particularly valuable in music-related applications, where experienced musicians appreciate the use of familiar controls that allows them to operate software in the same way that they operate hardware musical equipment (c.f Sections 7.3.3 and 7.6.4). New users of VoodooIO are introduced to the notion that controls can be added and removed from the interface, which is not normally possible with traditional user equipment, and hence not an action that users are likely to be familiar with. However, this technique is easy to grasp by novice users, quite likely aided by the fact that the control/substrate attachment mechanism is in itself a familiar action – closely resembling the ubiquitous practice of affixing pushpins to notice boards. Our experience demonstrates that users are able to comprehend and adopt this concept as a novel interaction technique very easily (c.f. Section 7.5.3).

A second factor that influences learnability is *synthesizability*, which is the ability of a user to judge the effect of their actions on the interface (or, conversely, the "honesty" of the interface in accurately reflecting the users actions). As a result of their physicality, VoodooIO controls inherently provide visual and tactile feedback regarding their state. For example, when a physical slider control is moved to its maximum position, the user can both feel and touch the fact that its associated interface parameter has reached its limit. Another example is the clear distinction between controls being "in" or "out of play," which reflects a user's latitude for interaction at any point in time: when a control is detached from the substrate, it is, unambiguously, both physically-disassociated and functionally-decoupled from the interface configuration (c.f. Sections 7.5.3 and 7.6.4).

Another two closely related properties are *generalizability* and *consistency*, which encourage users to learn particular interaction techniques in one context, and then apply them to different situations and across multiple applications. One example of generalizability in the design of VoodooIO in evident the way controls attach/detach from the substrate: once a user has learnt how to correctly pin one control into the substrate, they can then apply this technique across all other control instances and control types. The design is also consistent in that it encourages reuse of a particular set of interface controls, which are applicable across different applications – note, for example, how the same basic control types (buttons, sliders, joysticks and knobs) are reused across the various applications found throughout this chapter.

7.7.2 Flexibility

Dix defines *flexibility* in interaction as the multiplicity of ways the end-user and the system exchange information. Increasing interaction flexibility is the main concern behind the design of VoodooIO, and most of its user-related benefits stem from its ability to support multiple ways for the user to interact with a system.

There are a number of properties that underpin flexibility, and the most relevant to this discussion is the degree to which the interface supports user *customizability* – the ability of the user to modify the input/output capabilities of the interface. VoodooIO is highly flexible, particularly in its physical instantiation that allows users to *embed* the interface into their environment. This property is afforded by the shapeable substrate, which can be cut to shape to create custom control areas that can be flexibly deployed on vertical and horizontal surfaces, and integrated with objects and furniture. Examples of this include the way the substrate was used to build a cockpit around a standard game-playing setup (c.f. Section 7.4.1) or shaped to accommodate a keyboard on a desk (c.f. Section 7.4.2).

A second interaction property related to interface customizability is the way that VoodooIO supports fluid adaptation of control layouts. As illustrated by the experiment results reported in Section 7.2.2, users are able to make use of this feature to optimise the control arrangement, and tailor it based on personal preference to make it more effective and comfortable to use. The ability to easily reconfigure the control setup also encourages exploration of interaction alternatives, and supports unexpected forms of improvisation (e.g. the creation of an impromptu cross-fader by re-orienting volume faders, mentioned in Section 7.3.2). This same property can be harnessed by product designers as a way to quickly sketch out interface concepts that can be interactively modified during the early stages of evaluation (c.f. Section 7.1.3).

There one more aspect of the flexibility of VoodooIO that is related to customisability but is not explicitly captured by Dix's usability principles, and that is the way in which users can *appropriate* the interface by coupling it with other interface devices, and adapting it to circumstances for use in unexpected ways. Several instances of interface appropriation have been observed in the application examples, including: projecting onto the substrate, and carving out windows to reveal graphical displays (c.f. Section 7.1.1); using pen and paper to label control arrangements (c.f. Section 7.4.2), and modifying controls with additional materials to customise their look-and-feel (c.f. Section 7.1.2).

There is a particular form of interface flexibility that Dix refers to as *substitutivity*, which relates to the ability of a user to to arbitrarily substitute equivalent values of input and output with others that achieve the same purpose, but in a different way. The design of VoodooIO removes the necessity to hard-wire a specific control instance with a fixed function, and can allow the user to swap a control with another (or multiple other) controls with compatible input/output characteristics (c.f. Section 5.6). The degree to which substitutivity is made available to the user is an issue that needs to be carefully addressed in the design of individual applications; it may be applicable to some applications and not to others, but in principle it is supported by VoodooIO. A practical example of different levels substitutivity in VoodooIO can be found in the tank-game that was developed to carry out the "make-your-own-interface" experiment (c.f. Section 7.2). The game allowed users to select between a dial *or* a slider for the "set angle" and "set power" parameters, but limited the choice of control for "fire" to a button (of any colour). The application provided flexibility through substitutivity in the cases where it was sensible to do so (both sliders and dials are 'interchangeable' in that they both provide one-dimensional, continuous input), and limited it in cases where it was less appropriate (there was no ambiguity of how a momentary button-press mapped to the binary action of firing the cannon, which would have been the case if a slider/dial had been used for this purpose).

The ability for a user to perform multiple and independent interactions on the system is called *multithreading*, and is defined by Dix as being either a *concurrent* (supporting parallel input/output through multiple channels) or *interleaved* process (using time to mediate multiple interactions over a single channel). The principle is closely related to the discussion of space/time-multiplexing, and benefits of using specialised interaction devices (c.f. Section 2.2.1). VoodooIO supports concurrent, space-multiplexed input and output, which allows users the desirable flexibility to interact with multiple controls in parallel, benefiting multi-handed and multi-user interaction (c.f. Section 7.4.3).

There is one final issue to discuss with regards to the flexibility of VoodooIO, and it relates to an earlier discussion regarding adaptive-versus-adaptable user interfaces (c.f. Section 2.4.1). The physical elements of VoodooIO afford manual user adaptation, but cannot support proactive, system-initiated adaptability. This is a technological limitation that is not likely to be overcome in the near future, as, in order to support full interface adaptability (to the same degree that manual adaptation is currently possible), the physical interface would need to be able to dynamically morph its shape, instantiate and dispose of physical controls on demand, and be able to autonomously exchange one physical control type for another. This limitation, however, is in line with the original philosophy for the development of physical flexible interfaces and does not hinder its realisation: the goal is to place flexibility in the hands of the user, rather than the system.

7.7.3 Robustness

The *robustness* of a user interface is determined by how well the interface supports the user in achieving their interaction goals – a robust user interface should provide an efficient and enjoyable interaction experience to its users. One of the principles influencing robustness is *observability*, which allows users to comprehend the internal state of the system by perceiving its interface. VoodooIO contributes to the observability of user interfaces by making the interaction legible and explicit through the use of distributed physical controls, which allows the user (and anyone observing the interaction) to infer the state of the system by the presence, arrangement state of controls on substrate areas. This is validated by the feedback we received from audience members observing the performance of VoodooIO ad hoc musical instruments (c.f. Section 7.6.4), where people appreciated the ability to see, infer and anticipate changes in the music as a result of the performer's actions on the interface. Another example is the comment that we received from one of the participants in the experiment reported in Section 7.5, who drew a parallel to WYSIWYG, pointing out that they appreciated the fact that “what you see is what you hear.”

One factor that enables observability is the *persistence* of interface controls. As VoodooIO controls are tangible physical objects, they exist permanently in the real world and do not disappear when the system is turned off. This feature can be particularly important for ubiquitous computing applications, where the state of the system may be hard to infer from anything other than its interface, as the system itself is distributed and 'invisible' in the environment. Although the control configuration of VoodooIO interfaces is fluid and changeable, the pin connection mechanism affords *freezing* of the interface configuration in a particular arrangement, in a manner that is unlikely to change accidentally, and until controls intentionally detached from the substrate. This property is important, and contributes to interface robustness by allowing users to internalise and become proficient with a particular control configurations over extended periods of time [84].

Recoverability refers to the ability of a user to correct mistakes by going back to a previous stage in the interaction. This is an important concern of the design of VoodooIO, as it is essential that users feel free to explore interaction alternatives and are encouraged to try different options without being penalised by causing mistakes or making bad decisions. Most mistakes with VoodooIO result from users incorrectly orienting controls on the substrate, or finding, after placing a control, that a different location or control type might have been preferable. From our experience, even novice users present no hesitation in correcting these mistakes, and need little or no prompting to realise that they can simply detach (and reattach) a control to reverse their actions (c.f. Section 7.2). One factor that might influence recoverability and contribute to user confidence is the self-healing property of the substrate material. If every time a control was removed its pins left permanent holes, or otherwise damaged the surface of the material, this could be perceived as an irreparable cost associated with making a mistake. As it is, the holes close over and leave no permanent trace or visible degradation: as far as the user is concerned, there is no penalty associated with changing their mind or exploring alternatives.

Responsiveness is a measure of how quickly the system can detect and react to user interaction. Short response times, and stable (consistent) response times are usually desirable in an interactive system. The way VoodooIO affects the overall responsiveness of a system was analysed in detail in Chapter 4, which concludes that VoodooIO supports reasonably low and stable response times that can appropriately support user expectations for a large number of applications. However, further improvements to the overall responsiveness of the system (especially for manipulation of multiple controls in parallel) would be beneficial and form the basis for future work.

The final principle to be discussed is *task conformance*: does the interface allow the user to do everything they want, in a way that is understandable by a user? In principle, VoodooIO is very well suited to support taks conformance, as it allows the user to grudally adapt, extend and modifiy their interface to suit an evolving set of needs and task requirements. For example, it allows the user to *reduce* the interface controls to only those that are necessary to produce a particular piece of music (c.f. Section 7.3.3), and *extend* the interface as new actions become available throughout the lifetime of a game (c.f. Section 7.4.3). The ability to support task conformance across evolving task requirements and individual user preference is perhaps the most exciting property of VoodooIO. It is also the most difficult to collect evidence for, particularly given the relatively short-term experiences that form the basis of our investigations throughout this chapter. In order to gain further insights into the possibilities enabled by flexible physical interfaces, as enabled by technologies such as VoodooIO, will require future work based on longitudinal studies, where users are allowed to work for extended periods to develop, shape and maintain their personal interaction environments over time.

Chapter 8

Conclusion

This thesis has covered the various facets of motivating, developing, evaluating and studying a novel user interface technology for flexible user interfaces. As a final note, this chapter provides a summary of the contributions that have been made to the field of human-computer interaction, discusses the initial impact of the work amongst the research community, and points to further work in this area.

8.1 Summary of Contributions

The introduction to this thesis motivated the development of physical flexible interfaces: user interface designs that exhibit a high degree of user adaptability and reconfigurability in their physical control composition. The work was inspired by a number of interface design philosophies from the last three decades of human-computer interaction research, which were discussed in Chapter 2.

Chapter 3 developed a vision where the human-computer interface is deconstructed into a set of modular controls that can be individually added, removed and physically manipulated by users; the aim was to provide greater flexibility in the physical control structure. This goal was facilitated through the development of an architecture for user interfaces as networks of controls, where each control is implemented as a network node with physical input and output capabilities. The architecture overcomes the inflexibility that is usually imposed by hard-wired circuitry in traditional interface devices, by enabling individual control elements that can be connected and disconnected ad hoc from a network bus. The key features of the architecture are: a lightweight design, requiring minimal cabling and simple networking components; support for a wide and extensible range of control types; fast control identification and presence detection, and an application-level interface that abstracts from low level implementation details and network management processes.

The performance of the architecture was characterised by its system response time – the period that elapses between the user interacting with the interface and the system detecting the interaction event. Chapter 4 presented the results of a series of tests that measured how quickly the architecture can detect a control device being added, removed or manipulated. The results showed how the system can be optimised to detect either control addition/removal, or control manipulation. A further set of tests results showed how the system performs with a variable number of controls on the network. The results demonstrated that, while the system response time is an issue and can result in perceptible interface delays, the performance of the system is nonetheless adequate to support user interaction.

The architecture for networked controls is given concrete physical form in the VoodooIO design, which was documented in Chapter 5. The design is based on the idea of using rubberised conductive sheets as a network medium, to which physical control nodes can attach via a pin-connector mechanism. The VoodooIO design introduces an innovative composition for the substrate material, the use of coaxial-pins as digital/physical connectors for control devices, and a range of control instances with varied input and output capabilities. The physical design of VoodooIO lends itself to be shaped, configured and manipulated in novel ways. The chapter concluded by illustrating the experience of working with the substrate material and physical control devices, and highlighting the novel interaction possibilities enabled by a flexible form factor.

Chapter 6 described the software tools that have been developed to support application development and integration for VoodooIO. Here, the goal was to maximise the deployment opportunities of flexible interfaces, and increase the range of possible applications that can be explored. The tools that have been developed include: a software proxy/client application interface model, which supports remote and distributed applications running on heterogeneous computer platforms; programming libraries, aimed at application developers and interaction designers who wish to include physical interface flexibility in their designs, and mapping tools that provide end-users with ways to integrate VoodooIO into the user interface of existing computer applications.

Evidence of the new interface design possibilities and novel interaction techniques that are enabled by VoodooIO were collected through a number of different research exercises, which were the subject of Chapter 7. The practice of product prototyping is the first application area that was explored: through hands-on workshops, expert participants were exposed to the VoodooIO platform as a tool to quickly sketch out working models of interface ideas and product concepts. The observed results demonstrated that this target user group is able to harness the flexibility of the physical interface as a means to develop a design brief, in a manner that integrates well with current practice , while at the same time affording room for extension and appropriation of the materials. A second theme that was explored is end-user interface adaptation, where regular computer users were given the ability to adapt and personalise their physical interface configuration. An insight into end-user interface configuration strategies was gained from an observational study, where participants are encouraged to explore different control configurations and layouts for playing a game. The case for end-user physical interface flexibility was further strengthened by a series of reported application exercises, where VoodooIO was applied to music production game-playing tasks. The later sections of Chapter 7 dealt in more detail with a particular interaction technique, which is made possible with VoodooIO: association of control presence with application functionality. User acceptance of this technique was established by the results of a user study, and was then applied in practice in the design of interfaces for ad hoc musical instruments, which interleave performance with the fabrication of the instrument itself. Finally, in order to synthesise the user benefit across all the different application experiences, Chapter 7 concludes with a detailed usability analysis of VoodooIO.

8.2 Initial Impact and Future Work

The development of VoodooIO has received positive feedback and notable interest from the research community. In addition to forming the basis for a number of peer-reviewed publications ([91], [82], [90]) the work has been been presented in a number of high profile events, including a five-day exhibit as part of the ACM SIGGRAPH 2006 Emerging Technologies track [88], and an interactive demonstration at the 2006 ACM Symposium on User Interface Software and Technology [87]. The application of VoodooIO as a gaming interface won both the best paper and best demonstration awards at the 2006 ACM Conference on Advances in Computer Entertainment.

A VoodooIO kit has been in regular use by colleagues at the School of Product Design, University of Wales Institute, Cardiff, where it has been used as tool for quickly sketching functional product designs. Researchers from that institution have plans to develop the technology further, and integrate it into their own ongoing research efforts into creating tools to support the process of rapid prototyping of product concepts. As part of a collaboration with the ID-StudioLab of Delft University of Technology in the Netherlands, VoodooIO is being used as part of an explorative research project into the applicability of localised information appliances as a means to educate a rural population in the Gujarat region of India. VoodooIO will provide on-site and end-user physical interface reconfigurability for one of the information appliances, allowing appointed members of the community to tailor and adapt the interface layout as the appliance is updated with information over its period of deployment.

Future work in this area will benefit from more longitudinal studies, exploring how users adapt and personalise their interaction environments over extended periods of time, in real-world applications and outside of the research laboratory. A joint grant from the Equator Interdisciplinary Research Collaboration, and the Microsoft Research centre in Cambridge has made possible the manufacture of a number of VoodooIO kits, which are in production at the time of writing. Each kit will contain a large sheet of the substrate material, a comprehensive set of controls and all the cables and software necessary for development. The kits will be made available to the wider research community as an open research platform, with the goal to inspire and facilitate the further study of flexible physical interfaces and further the possibilities of human-computer interaction.

8.3 Closing Remarks

The various research efforts presented in this thesis can be deemed successful in validating the original motivation for flexible physical interfaces. The development of VoodooIO provides a precedent for a user interface equipment that is end-user adaptable in its physical configuration and logical functionality; its underlying implementation as a network of controls demonstrates a way to achieve ad hoc control reconfigurability in a manner which is both practical and functional, and its physical design introduces a novel form factor that provides users with powerful mechanisms to adapt and appropriate their interaction environment. As a research platform, VoodooIO has proven effective as a vehicle to gather evidence of user benefit from its application, and, in the future, it will provide valuable tools for further exploration into the role of physical interface flexibility in human-computer interaction.

Bibliography

1. *Arduino.* http://www.arduino.cc/, 2007.

2. Avrahami, D., Hudson, S. E. Forming interactivity: a tool for rapid prototyping of physical interactive products. In Proceedings of *Designing Interactive Systems (DIS).* pp. 141-146, 2002.

3. Ballagas, R., Szybalski, A., Fox, A. Patch panel: enabling control-flow interoperability in ubicomp environments. In Proceedings of *Conference on Pervasive Computing and Communications (PerCom).* pp. 241-252, 2004.

4. Ballagas, R., Ringel, M., Stone, M., Borchers, J. iStuff: A physical user interface toolkit for ubiquitous computing environments. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 537-544, 2003.

5. Schneiderman, B. Response Time and Display Rate in Human Performance with Computers, *Computing Surveys,* vol. 16: pp. 265-285, 1984.

6. Blackwell, A. F., Hague, R. AutoHAN: An Architecture for Programming the Home. In Proceedings of *IEEE Human-Centric Computing Languages and Environments.* pp. 150-157, 2001.

7. Block, F., Villar, N., Gellersen, H., Hazas, M., Molyneaux, D. Locating physical interface objects on interactive surfaces. In Proceedings of *Workshop on Mobile and Embedded Interactive Systems.* pp. 374-381, 2006.

8. Bowers, J. *Improvising Machines: Ethnographically informed design for improvised electro-acoustic music.* http://www.ariada.uea.ac.uk/ariadatexts/ariada4/index4.html, 2003.

9. Bowers, J., Archer, P. Not hyper, not meta, not cyber but infra-instruments, *New Interfaces for Musical Expression (NIME),* vol. pp. 5-10, 2005.

10. Bowers, J., Villar, N. Creating ad hoc instruments with Pin&Play&Perform. In Proceedings of *New Interfaces for Musical Expression (NIME).* pp. 234-239, 2006.

11. Brown, E., Buxton, W., Murtagh, K. Windows on tablets as a means of achieving virtual input devices. In Proceedings of *Human Computer Interaction (INTERACT).* pp. 675-681, 1990.

12. Butler, T. W. Computer response time and user performance. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 58-62, 1983.

13. Buxton, W. Lexical and Pragmatic Considerations of Input Structures, *Computer Graphics,* vol. 17: pp. 31-37, 1983.

14. Buxton, W., There's more to interaction than meets the eye: some issues in manual input in *Human-Computer Interaction*. Prentice Hall Press: NewJersey, USA, 1990.

15. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M., Vanderdonckt, J. Plasticity of User Interfaces: A Revised Reference Framework. In Proceedings of Workshop on Task Models and Diagrams for User Interface Design. pp. 127-134, 2002.

16. Card, S. K., Mackinlay, J. D., Robertson, G. G. A morphological analysis of the design space of input devices, ACM Transactions on Information Systems (TOIS), vol. 19: pp. 99-122, 1991.

17. Card, S. K., Moran, T. P., Newell, A. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates: New Jersey, 1983.

18. Churchill, E. F., Nelson, L. Tangibly simple, architecturally complex: evaluating a tangible presentation aid. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 750-751, 2002.

19. Cypher, A. *Watch What I Do.* MIT Press: Cambridge, Massachusetts, 1993.

20. Dey, A., Hamid, R., Beckmann, C., Li, I., Hsu, D. a CAPpella: Programming by demonstration of context-aware applications. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 33-40, 2004.

21. Dieterich, H., Malinowski, U., Kühme, T., Schneider-Hufschmidt, M., State of the art in adaptive user interfaces in Adaptive User Interfaces: Principles and Practice. North-Holland: Amsterdam, 1993.

22. Dix, A., Finlay, J., Abowd, G., Beale, R. *Human-Computer Interaction.* Prentice Hall Europe: Hemmel Hampstead, 1998.

23. Dragicevic, P., Fekete, J.-D. The input configurator toolkit: towards high input adaptability in interactive applications. In Proceedings of *Conference on Advanced Visual Interfaces (AVI).* pp. 244-247, 2004.

24. *Ergodex DX1 Input System.* http://www.ergodex.com, 2006.

25. Findlater, L., McGrenere, J. A comparison of static, adaptive, and adaptable Menus. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 89-96, 2004.

26. Fishkin, K. P. A taxonomy for and analysis of tangible interfaces, *Personal and Ubiquitous Computing,* vol. 8: pp. 347-358, 2004.

27. Fishkin, K. P., Gujar, A., Harrison, B. L., Moran, T. P., Want, R. Embodied User Interfaces for Really Direct Manipulation, *Communications of the ACM,* vol. 43: pp. 74-80, 2000.

28. Fitzmaurice, G. W., Buxton, W. An Empirical Evaluation of Graspable User Interfaces. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 43-50, 1997.

29. Fitzmaurice, G. W., Ishii, H., Buxton, W. Bricks: Laying the Foundations for Graspable User Interfaces. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 442-449, 1995.

30. Foley, J. D., Wallace, J. D. The Art of natural Graphic Man-Machine Interaction, *Proceedings of the IEEE,* vol. 62: pp. 462-471, 1974.

31. *Gainer.* http://www.gainer.cc/, 2007.

32. Gaver, W. Technology Affordances. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 79 - 84, 1991.

33. Gibson, J. J. *The Ecological Approach to Visual Perception.* Houghton Mifflin: Boston, 1979.

34. Gill, S. Developing Information Appliance Design Tools for Designers, *Personal and Ubiquitous Computing,* vol. 7: pp. 159 - 162, 2003.

35. Goodman, T., Spence, R. The Effect of System Response Time on Interactive Computer Aided Problem Solving. In Proceedings of *ACM SIGGRAPH Computer Graphics.* pp. 100-104 , 1978.

36. Gorbet, M. G., Orth, M., Ishii, H. Triangles: tangible interface for manipulation and exploration of digital information topography. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 49–56, 1998.

37. Greenberg, S.,
, C. F. Phidgets: easy development of physical interfaces through physical widgets. In Proceedings of *ACM Symposium on User Interface Software and Technology (UIST).* pp. 209-218, 2001.

38. Greenberg, S., Boyle, M. Customizable physical interface for interacting with conventional applications. In Proceedings of *ACM Symposium on User Interface Software and Technology (UIST).* pp. 31-40, 2002.

39. Guynes, J. L. Impact of System Response Time on State Anxiety, *Communications of the ACM,* vol. 31: pp. 342-347, 1988.

40. Hakansson, M., Ljungbad, S., Holmquist, L. E. Like solving a giant puzzle: Supporting collaborative scheduling at a film festival. In Proceedings of *Human Computer Interaction (INTERACT).* pp. 209-218, 2001.

41. Hardenberg, C. v., Brard, F. Bare-Hand Human-Computer Interaction. In Proceedings of *Perceptual User Interfaces.* pp. 113–120, 2001.

42. Harrison, B. L., Fishkin, K. P., Gujar, A., Mochon, C., Want, R. Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 17-24, 1998.

43. Hartmann, B., Klemmer, S. R., Bernstein, M., Mehta, N. Reflective physical prototyping through integrated design, test, and analysis. In Proceedings of *ACM Symposium on User Interface Software and Technology (UIST).* pp. 299-308, 2006.

44. Hoch, M., Bowers, J., Jää-Aro, K.-M., Hellström, S.-O. *The Round Table.* http://www.nada.kth.se/erena/table.html,

45. Holmquist, L. E., Redstrom, J., Ljungstrand, P. Token-Based Access to Digital Information. In Proceedings of First International Symposium on Handheld and Ubiquitous Computing (HUC). pp. 234-??, 1999.

46. Hudson, S., Mankoff, J. Rapid Construction of Functioning Physical Interfaces from Cardboard, Thumbtacks, Tin Foil and Masking Tape, *ACM Symposium on User Interface Software and Technology (UIST),* vol. pp. 289-298, 2006.

47. Ishii, H., Ullmer, B. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 234-241, 1997.

48. Kaltenbrunner, M., Jorda, S., Geiger, G., Alonso, M. The reacTable*: A Collaborative Musical Instrument. In Proceedings of IEEE International Workshop on Enabling Technologies. pp. 406-411, 2006.

49. Kantorowitz, E., Sudarsky, O. The adaptable user interface, *Communications of the ACM,* vol. 32**:** pp. 1352-1358, 1989.

50. Kawai, S., Aida, H., Saito, T. Designing interface toolkit with dynamic selectable modality. In Proceedings of ACM SIGACCESS Conference on Assistive Technologies. pp. 72-79, 1996.

51. Kirsh, D. The intelligent use of space, *Artificial Intelligence,* vol. 73**:** pp. 31-68, 1995.

52. Kishino, Y., Terada, T., Villar, N., Nishio, S. A Position Detection Mechanism using Camera Images for Pin-Shaped Input/Output Devices. In Adjunct Proceedings of *International Conference on Ubiquitous Computing (UbiComp).* pp. 308 - 317, 2006.

53. Klemmer, S. R., Li, J., Lin, J., Landay, J. A. Papier-Mâché: toolkit support for tangible input. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 399-406, 2004.

54. Klemmer, S. R., Newman, M. W., Farrell, R., Bilezikjian, M., Landay, J. A. The designers' outpost: a tangible interface for collaborative web site. In Proceedings of *ACM Symposium on User Interface Software and Technology (UIST).* pp. 1-10, 2001.

55. Kurakake, R., Nishizawa, Y., Sakakura, K., Ouchi, H., Minami, M., Morikawa, H. Magic Surfaces: A Smart Building Material for Indoor Sensing Infrastructures. In Proceedings of *Fourth International Conference on Networked Sensing Systems.* pp. 213-220, 2007.

56. Lee, J. C., Avrahami, D., Hudson, S. E., Forlizzi, J., Dietz, P. H., Leigh, D. The Calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In Proceedings of *Designing Interactive Systems (DIS).* pp. 167-175, 2004.

57. Lifton, J., Seetharam, D., Broxton, M., Paradiso, J. A. Pushpin computing system overview: A platform for distributed, embedded, ubiquitous sensor networks. In Proceedings of *Pervasive Computing.* pp. 139-151, 2002.

58. Little, G., Lau, T. A., Cypher, A., Lin, J. Koala: capture, share, automate, personalize business processes on the web. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 943-946, 2007.

59. MacLean, A., Carter, K., Lovstrand, L., Moran, T. P. User-tailorable systems: pressing the issues with buttons. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 175-182, 1990.

60. Marquadt, N., Greenberg, S. Shared Phidgets: A Toolkit for Rapidly Prototyping Distributed Physical User Interfaces. In Proceedings of *Tangible and Embedded Interaction (TEI).* pp. 13-20, 2007.

61. Maxim/Dallas. *1-Wire.* www.maxim-ic.com/1-Wire.cfm, 2007.

62. Merril, D., Kalanithi, J., Maes, P. Siftables: towards sensor network user interfaces. In Proceedings of *Conference on Tangible and Embedded Interaction.* pp. 75-78, 2007.

63. Nakatani, L. H., Rohrlich, J. A. Soft machines: A philosophy of user-computer interface design. In Proceedings of Conference on Human Factors in Computing Systems (CHI). pp. 9 - 23, 1983.

64. Newton-Dunn, H., Nakano, H., Gibson, J. Block jam: A tangible interface for interactive music. In Proceedings of *New Interfaces for Musical Expression (NIME).* pp. 170-177, 2003.

65. Ng, K. H., Benford, S., Koleva, B. PINS push in and POUTS pop out: creating a tangible pin-board that ejects physical documents. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 1981-1984, 2005.

66. Norman, D. *The Design of Everyday Things.* Basic Books: New York, 1988.

67. Norman, D. Affordance, convention and design, *Interactions,* vol. 6: pp. 38-42, 1999.

68. Patten, J., Ishii, H. A comparison of spatial organization strategies in graphical and tangible user interfaces. In Proceedings of *Designing Augmented Reality Environments (DARE).* pp. 41-50, 2000.

69. Patten, J., Ishii, H., Hines, J., Pangaro, G. Sensetable: a wireless object tracking platform for tangible user interfaces. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 253-260, 2001.

70. Patten, J., Recht, B., Ishii, H. Audiopad: a tag-based interface for musical performance. In Proceedings of *New Interfaces for Musical Expression (NIME).* pp. 1-6, 2002.

71. Pering, C. Interaction design prototyping of communicator devices: towards meeting the hardware-software challenge, *Interactions,* vol. 9: pp. 36-46, 2002.

72. *Phidgets.* http://www.phidgets.com, 2007.

73. Repenning, A., Ioannidou, A. Agent-based end-user development, *Communications of the ACM,* vol. 47: pp. 43-46, 2004.

74. Resnik, M. Behavior construction kits, *Communications of the ACM,* vol. 36: pp. 64-71, 1993.

75. Rodden, T., Crabtree, A., Hemmings, T., Koleva, B., Humble, J., Akesson, K. P., Hansson, P. Configuring the Ubiquitous Home. In Proceedings of *ACM Symposium on Designing Interactive Systems.* pp. 191-220, 2004.

76. *Saitek Cyborg Gamepad.* http://www.saitek.com/uk/prod/p3600v2.htm, 2007.

77. Schneiderman, B. Direct manipulation: a step beyond progamming languges, *IEEE Computer,* vol. 16: pp. 57-69, 1983.

78. Scott, J., Hoffmann, F., Addlesee, M., Mapp, G., Hopper, A. Networked surfaes: a new concept in mobile networking, *Mobile Networked Appliances,* vol. 7: pp. 353-364, 2002.

79. Smith, D. C. *A Computer Program to Model and Stimulate Creative Thought.* Birkhauser Verlag: Basel, 1977.

80. Sohn, T. Y., K., D. A. iCAP: An Informal tool for Interactive Prototyping on Context-Aware Applications. In Proceedings of *International Conference on Pervasive Computing.* pp. 974-975, 2006.

81. Souza, C. S. d. *The Semiotic Engineering of Human-Computer Interaction.* MIT Press: London, England, 2005.

82. Spiessl, W., Villar, N., Gellersen, H., Schmidt, A. VoodooFlash: authoring across physical and digital form. In Proceedings of *Tangible and Embedded Interaction (TEI).* pp. 97-100, 2007.

83. Ullmer, B., Ishii, H. Emerging Frameworks for Tangible User Interfaces, *IBM Systems Journal,* vol. 39: pp. 915-931, 2001.

84. Ungvary, T., Veertegaal, R. The SensOrg: a musical cyberinstrument with a cognitive ergonomic touch. In Proceedings of IEEE International Conference on Systems, Man, and Cybernetics. pp. 1066-1071, 1998.

85. Van Laerhoven, K., Villar, N., Schmidt, A., Gellersen, H., Håkansson, M., Holmquist, L. E. Pin&Play: The Surface as Network Medium, *IEEE Communications Magazine,* vol. 41: pp. 90-96, 2003.

86. Vertegaal, R., Ungvary, T., Kieslinger, M. Towards a Musician's Cockpit: Transducers, Feedback and Musical Function. In Proceedings of *International Conference on Computer Music.* pp. 308-311, 1996.

87. Villar, N., Block, F., Gellersen, H. Distributed and Adaptable Home Control. In Adjunct Proceedings of *ACM Symposium on User Interface Software and Technology (UIST).* 2006.

88. Villar, N., Block, F., Gellersen, H., Molyneaux, D. *VoodooIO SIGGRAPH E-Tech Exhibit.* http://www.siggraph.org/s2006/main.php?f=conference&p=etech&s=voodoo,

89. Villar, N., Gellersen, H. The Friday Afternoon Project: A Two-Hour VoodooIO Prototyping Exercise. In Proceedings of *Workshop on Mobile and Embedded Interactive Systems.* pp. 411-420, 2006.

90. Villar, N., Gellersen, H. A malleable control structure for softwired user interfaces. In Proceedings of *Tangible and Embedded Interaction (TEI).* pp. 46-56, 2007.

91. Villar, N., Gilleade, K. M., Ramduny-Ellis, D., Gellersen, H. The VoodooIO Gaming Kit: A Real-Time Adaptable Gaming Controller. In Proceedings of *Tangible and Embedded Interaction (TEI).* pp. 40-46, 2006.

92. Villar, N., Lindsay, A. T., Gellersen, H. Pin&Play&Perform: a rearrangeable interface for musical composition and performance. In Proceedings of *New Interfaces for Musical Expression (NIME).* pp. 188-191, 2005.

93. Wang, G., Cook, P. On-the-fly programming: Using Code as an Expressive Musical Instrument. In Proceedings of *New Interfaces for Musical Expression (NIME).* pp. 138-143, 2004.

94. Weinberg, G., Gan, S.-L. The Squeezables: Toward an Expressive and Interdependent Multi-player Musical Instrument, *Computer Music Journal,* vol. 25: pp. 37-47, 2001.

95. Weiser, M. The Computer for the Twenty-First Century, *Scientific American,* vol. 265: pp. 94-104, 1991.

96. Wong, J., Hong, J. Making mashups with marmite: towards end-user programming for the web. In Proceedings of *Conference on Human Factors in Computing Systems (CHI).* pp. 1435-1444, 2007.

97. Wyeth, P., Wyeth, G. Electronic blocks: Tangible programming elements for preschoolers. In Proceedings of *IFIP TC13 Conference on Human-Computer Interaction.* 2001.

Appendix

VoodooIO Protocol

VoodooIO Services make use of a bi-directional protocol to communicate with client application Proxies. The protocol encodes event and request information as formatted ASCII strings, which are transmitted over a TCP/IP connection. The design of the protocol is intentionally simplistic: the ASCII text is human readable and easy to debug, while the TCP/IP protocol provides a highly generic transport mechanism.

Connection and Handshake Messages

Upon connection of a client proxy to a service there is a three-packet handshake sequence, which is outlined below. To begin with, the service sends to the client the following string:

```
$:SERVICE_INFO:<SERVICE_ID>:V_<VERSION>
```

Where <SERVICE_ID> is a verbose identifier of the service, usually referring to its physical location or other user-identifiable characteristic. <VERSION> is the current version of the service, and allows the client to check that it supports a compatible protocol version.

Upon receipt of the above message, the client responds to the service with:

```
$:APPLICATION:<APPLICATION_NAME>:V_<VERSION>
```

Where <APPLICATION_NAME> is a verbose description of the connecting application, and allows the service to keep track of which applications it is serving at any point in time. <VERSION> is the current client-supported protocol version. If the protocol evolves or changes, this allows the service to detect a legacy application and modify future communication exchanges to accommodate it. If a compatible protocol version is found, the service finalises the exchange by sending:

```
$:OK:<SERVICE_ID>:PROTOCOL_VERSION(V_<VERSION>)
```

Where the <SERVICE_ID> is a unique identifier for this service, and allows clients to keep track of future exchanges with this service in amongst any number of services to which they can connect simultaneously.

Service Event Messages

The following series of messages are sent by the service in the event of any detected user interactions with the VoodooIO interface: adding, removing and manipulating controls. When a new control is added, the server issues the following message to all connected clients to describe the newly added control:

```
$:ADDED:<SERVICE_ID>:<ID>:<FUNCTION_LABEL>:<FAMILYNAME>:<OUTPUTS
>:
  <INPUTS>
```

Where <SERVICE_ID> is the unique identifier of the issuing service; <ID> is the unique identifier of the newly added control; <FUNCTION_LABEL> is a user- or application-assignable description of what the control's functionality is (e.g. "Volume Knob"); <FAMILYNAME> describes the control-type (e.g. "DIAL"); and <OUTPUTS> and <INPUTS> specify the input/output characteristics of the control:

```
<[OUTPUT|INPUT]_NAME>,<TYPE>,<VALUE>;<OUTPUT_NAME>,<TYPE>,<VALUE
>...
```

Where <TYPE> can be "digital" or "analogue." The client application does not need any previous knowledge of the particular input/output capabilities of any control type, as the "ADDED" message encapsulates all this information. When a control is removed from the network, a much simpler message suffices to inform client applications of this fact:

```
$:REMOVED:<SERVICE_ID>:<ID>:<FUNCTION_LABEL>
```

Every time a user manipulates a control (e.g. turning the "Volume Knob" / "DIAL"), and a change is detected on that control's state, the service sends an event to all connected clients to inform them of the change and the control's new values:

```
$:MANIPULATED:<SERVICE_ID>:<ID>:<FUNCTION_LABEL>:<INPUT_NAME>,
<VALUE>; <INPUT_NAME>,<VALUE>...
```

Client Request Messages

Client applications can modify the output of controls by requesting a service to alter their state with the following message:

```
$:SET:<TARGET_SERVICE_ID>:<TARGET_TYPE>:<TARGET_OBJECT>:<OUTPUT_
NAME>,<VALUE>;<OUTPUT_NAME>,<VALUE>...
```

The result is acknowledged by the service with a respective message:

```
$:SET:<TARGET_SERVICE_ID>:<TARGET_TYPE>:<TARGET_OBJECT>:<OUTPUT_
NAME>,<VALUE>;<OUTPUT_NAME>,<VALUE>...
```

The client can easily check the success or failure of its requests in an asynchronous way by performing a direct comparison between sent messages and received replies. Finally, there is one more request that a client application can make on a service:

```
 $:REQUEST_UPDATE
```

This message is usually sent by the client application immediately after the handshake sequence in order to request an updated view of the service state (present controls, and control states). In response, the service will issue a series of "ADDED" events: one for each control which was already added before the application connected.