

Environment control and computer interface for
people with severe physical disabilities

Steven Daniel Webb

Supervisors: Brett McLoughlin and Prof., Dr Geoff West

April 15, 2005

Contents

1	Introduction	2
1.1	Stephen Hawking	2
1.2	Computer Usage	3
1.3	Cost on the health care system	3
1.4	Socio-economic disadvantage	4
1.5	Goals	4
2	Literature Review	6
2.1	Special Access Technology	6
2.2	Switch driven computer interfaces	6
2.2.1	Gnome Onscreen Keyboard (<i>GOK</i>)	7
2.2.2	Microsoft Windows	7
2.2.3	Apple Macintosh software	7
2.3	Gnome Human Interface Guidelines (<i>HIG</i>)	7
2.4	Health Technology Consultancy Infra Red Device	9
2.5	Linux Infrared Remote Control (<i>LIRC</i>)	9
2.6	Objectives	9
3	Approach and Methodology	11
3.1	Open Source Software	11
3.2	Internationalisation	12
3.3	Design Goals	12
3.3.1	Old/Slow Systems	12
3.3.2	Fast and adjustable	12
3.4	Reducing the number of features	12
3.5	Splitting the system into parts.	13
3.5.1	Uno applet	13
3.5.2	ppuirc	13
3.5.3	Icon Commands	14
4	Uno	15
4.1	Warning	15
4.2	Gnome Applet	15
4.3	Design	15

4.3.1	Actions	15
4.3.2	Pointer	17
4.3.3	Keyboard	22
4.3.4	Configuration Options	25
5	Icon Commands (IC)	27
5.1	Concept	27
5.2	How it works	27
5.3	Configuration	29
5.4	Gnome HIG	29
5.5	Extensibility	29
6	Configuring the operating system and desktop	30
6.1	Desktop	30
6.1.1	Background	30
6.1.2	Icons	30
6.2	Panel	31
6.3	Auto Login	32
7	Testing	33
7.1	The human trial	33
7.2	Setup	33
7.3	Testing and Results	34
8	Conclusion	36
8.1	Future Work	37
A	Tools	42
A.1	Linux	42
A.2	Language	42
A.2.1	C	42
A.2.2	C++	43
A.3	Gnome	43
A.4	Gtkmm	43
A.5	Libpanelapplet	43
A.6	XML	44
A.7	Doxygen	44
B	Parallel Port Universal Infra-red Controller (PPUIRC)	45
B.1	Design	45
B.2	libppuirc	45
B.3	ppuirc-switch	46
B.4	ppuirc-receive	46
B.5	ppuirc-send	47

List of Figures

4.1	The icons for the actions that the user can perform.	16
4.2	The uno applet cycling through the possible actions at four different times (t_1 to t_4).	18
4.3	Finding the y coordinate to perform a mouse click at.	19
4.4	Finding the x coordinate to perform a mouse click at.	19
4.5	The icons to send scroll up and scroll down events to the X server.	21
4.6	The icon used to activate the on-screen keyboard.	22
4.7	The on-screen keyboard.	22
4.8	Screen shot of the on-screen keyboard.	23
4.9	Screen shot of the on-screen keyboard with a frequency of use layout.	24
4.10	Selecting a key from the bottom row with scroll lock turned on.	25
4.11	Keyboard layout with the shift key pressed.	25
5.1	Screen shot of <i>IC</i>	28
6.1	The <i>Gnome</i> desktop configured for use with <i>Expand</i>	31
6.2	The <i>Show Desktop</i> applet.	32
7.1	The initial layout of the desktop used by Ken.	34

List of Tables

1.1	Onscreen keyboards for use with Apple Macintosh.	3
2.1	Onscreen keyboards for use with Microsoft Windows.	8
2.2	Onscreen keyboards for use with Apple Macintosh.	8

Chapter 1

Introduction

This project aims to develop a human/computer interface to allow a person with severe physical disabilities - but no mental disability - to have complete control of a computer. This interface has been incorporated into a system that allows the user to control their environment. The types of environmental control include: changing the temperature of the air-conditioning, turning lights on and off, switching channels on TV, controlling a VCR/DVD player, opening and closing window blinds, and much more. This system is called *Expand*, as its aims to expand a patient's control of their environment, access to information, and communication with others via the Internet. It is designed to be low cost or free.

Expand is part of an ongoing community project by Health Technology Consultancy (Health Technology Consultancy, 2005). The software developed for this project was financed by a Studentship Award (Office of Science and Innovation, 2005b) from the Office of Science and Innovation (OSI) (Office of Science and Innovation, 2005a), and the Western Australian Government (Western Australian Government, 1998). This software has been developed in collaboration with Curtin University of Technology (Curtin, 2003).

1.1 Stephen Hawking

Stephen Hawking (Hawking, 2001) is an excellent example of a person who may find this project helpful. Stephen Hawking is a world famous theoretical physicist, the author of "A Brief History of Time" (Hawking, 1988), and arguably one of the most intelligent people alive today. Unfortunately, Stephen Hawking suffers from Amyotrophic Lateral Sclerosis (ALS). He is confined to a wheel chair, is unable to speak, and is only capable of tiny movements at the time of writing this report. He communicates using software written by Words+ valued at US \$1395 (Words+, 2004), running on a computer donated by Intel. Unlike Stephen Hawking many people with severe physical disabilities are unable to generate an income for themselves, and cannot afford to purchase expensive

computers and software to improve their quality of life.

1.2 Computer Usage

Typically a computer is controlled using a keyboard and a mouse. A mouse is a type of pointer device, so this can be abstracted to: a keyboard and a pointer device. However, there is a growing number of new input devices for controlling a computer. For example, many graphical artists use a graphics tablet instead of a mouse (Wacom, 2004), and most operating systems provide an on-screen keyboard for instances when a real keyboard is not attached (Microsoft, 2000).

Company	Product	Price (EU \$)
Madentec	Discover:Screen	Not listed
AssistiveWare	KeyStrokes	\$250
AssistiveWare	SwitchXS	\$199

Table 1.1: Onscreen keyboards for use with Apple Macintosh.

A switch is the most basic input device for a computer. A switch can only send two signals to the computer: when it is activated, and when it is deactivated. A switch may take physical forms such as:

- A button that the user may press (similar to the button on a mouse),
- A lever that may be activated by a finger, eyebrow, or any muscle the user controls,
- A pressure switch that may be activated by a blow tube,
- Many more.

Regardless of the type of switch used the input is identical; hence, this system may be controlled by any form of switch that the user finds most convenient. The switch must be adjusted to be easy for the user to operate repeatedly without strain (Nisbet and Poon, 1998).

Expand includes a software driven keyboard and a pointer device, that are entirely controlled using a single switch. *Expand* will be as low cost as possible to the end user for both hardware and software.

1.3 Cost on the health care system

The Western Australian Government spends \$224 million annually supporting people with disabilities (McHale, 2004), a portion of which is spent on expensive equipment and carer support for people with severe physical disabilities. This project aims to make patients less dependent on their carers by giving them

control over their environment. Communication between patients requiring assistance and the care givers will also be improved, resulting in an increase of service for patients, at a reduced cost to the government. Requests for help will include a message stating their needs, allowing the care givers to triage their efforts and solve the most critical problems immediately. Note that as this system is still experimental it should only be used as an extension of current technology, not a replacement.

1.4 Socio-economic disadvantage

Many patients with physical disabilities are not employed, even though they may be active in seeking work. Severe physical disabilities drastically reduce the number of suitable jobs available, and patients confined to health care facilities have even fewer opportunities for employment. Unemployment forces many patients to be completely financially dependent on the government, family, and friends. The cost of purchasing expensive computer interfaces or environment control systems is beyond many patients and their support network.

The cost of this system will be as low as reasonably possible. The software will be distributed free of charge over the Internet, and has been designed to run efficiently on old computers, as it is unreasonable to assume that these patients can afford a new expensive computer. Convincing government and industry organisations to donate their old computers for use by patients is a long term goal of this project.

1.5 Goals

This project was developed with several goals in mind:

- Usable by everyone - This is extremely difficult to achieve; however, when design decisions have occurred that could possibly limit the range of possible users, we have attempted to choose the option that allows the greatest number of users. Several problems that fall into this category include: previous knowledge and experience with computers, poor vision, poor hearing, colour blindness, and learning disabilities.
- Flexible - the requirements for each user will vary greatly and the system must be flexible to cope with this.
- Configurable - As users gain more experience they will want to modify parts of the system to suit their usage. Users must be able to configure the software to run as fast as possible, as constant usage will quickly turn them into experts at using this software.
- Extensible - it should be easy to provide additional functionality for those who want or require it. As companies release new products they should be easily integrated into the system.

- Low cost or free - The target users will have little or no disposable income; hence, the system must be as low cost as is reasonably possible. The software will be a free download from the Internet.
- Small learning curve - This system should not be difficult for new users. Many people become frustrated quickly by systems that are difficult to use, and cease using them.

Each of these goals adds increased complexity to the project. Although not all of goals have been met, significant progress has been made in achieving them.

Chapter 2

Literature Review

This chapter begins with a review of the literature about computer interfaces for people with disabilities. This includes an overview of the current technologies available and their costs. Following this, material directly related to the systems development, but not related to people with disabilities, is discussed.

2.1 Special Access Technology

Special Access Technology (SAT) (Nisbet and Poon, 1998) is an excellent introduction to computer interfaces for people with physical disabilities. This book is aimed at both developers of technology and care givers. It covers the different technologies available and provides guidelines on how to choose the appropriate system to match a patient's abilities and needs. SAT also covers the aspects that make technology easy or difficult to use. Care givers are encouraged to download a free copy for reference.

SAT also covers many of the products currently available; however, many of these are obsolete as SAT was published in 1998 and are now out of date. Most of the companies have released new products, and this book can be used effectively to find them. This book is an excellent starting point for new researchers.

2.2 Switch driven computer interfaces

There are several software packages that simulate mouse and keyboard input, that are used to control a computer. The majority of this software uses a commercial licence - you need to pay for a license - hence, many potential users cannot use it as they cannot afford the software (See Section 1.4). This section gives an overview of the software available.

2.2.1 Gnome Onscreen Keyboard (*GOK*)

The Gnome Onscreen Keyboard (*GOK*) is an open source on-screen keyboard, licensed under the GNU Lesser General Public Licence (LGPL). It can be downloaded free of charge from the Internet (GOK, 2001). *GOK* uses Gnome 2's built-in accessibility framework to dynamically create an on-screen keyboard. As well as using the *X* server configuration file to generate the keyboard layout, any application which supports Gnome's AT SPI - part of the accessibility framework - will have its menus dynamically added to the instance of *GOK*. Applications that are controlled using the keyboard and menu items can be used easily and efficiently with *GOK* - such as a word processor. Many Gnome applications support the AT SPI; however, many applications are dependent on the mouse and cannot be used efficiently by *GOK* - such as web browsers. *GOK* does provide some facilities to control the mouse; however, it is slow and inefficient. Using *GOK* to operate a web browser would be a slow and frustrating experience.

2.2.2 Microsoft Windows

Assuming the user can afford the several hundred dollars for a Microsoft Windows Licence, this operating system includes an onscreen keyboard (Microsoft, 2000). However, from the website:

“The accessibility tools in the Windows operating system are intended to provide a minimum level of functionality for users with special needs. Most users with disabilities will need utility programs with more advanced functionality for daily use.” (Microsoft, 2000).

There is a variety of software packages for daily use that are compatible with Microsoft Windows, as listed in Table 2.1 along with their price. The more expensive packages tend to have a greater range of features than the lower priced ones. The two free on screen keyboards are included for completeness but lack features required by the target users of this project; therefore, they are not sufficient.

2.2.3 Apple Macintosh software

There is also a variety of packages for Apple systems as listed in table 2.2.

Note that the cost of Table 2.1 and Table 2.2 does not include the cost of the operating system, the computer, and the input switch. The user may also want additional software such as word processing or voice synthesis. The cost of a complete system is financially infeasible for many patients.

2.3 Gnome Human Interface Guidelines (HIG)

The Gnome Human Interface Guidelines (HIG) (Benson *et al.*, 2002) is a book detailing the design and layout of Graphical User Interfaces (GUIs) for Gnome

Company	Product	Price (US \$)
Mount Focus	Onscreen Virtual Keyboard	\$97
Origin Instruments	SofType	\$295
Innovation Management Group, Inc	OnScreen	\$99.95
Bloorview MacMillan Children's Centre	WiVik	\$350
Lake Software	Click-n-Type	Free
MiloSoft	Virtual On-Screen Keyboard	Free
Reach	Interface Author	\$329

Table 2.1: Onscreen keyboards for use with Microsoft Windows.

Company	Product	Price (EU \$)
Madentec	Discover:Screen	Not listed
AssistiveWare	KeyStrokes	\$250
AssistiveWare	SwitchXS	\$199

Table 2.2: Onscreen keyboards for use with Apple Macintosh.

applications. The HIG specifies the look, feel, layout, and behaviour for Gnome applications; and details the dos and don'ts of interface design. Two of the HIG goals directly relate to this project:

- Give all applications a consistent look and feel. Users will learn to use the application faster as its layout and design will be consistent with other applications.
- Allow all users to use the application, including those with disabilities and special needs.

This document was an important source of information for the design of *Expand* and the software developed for it. The HIG provided insight into how to make the software easy to use, consistent, and usable by people with disabilities. This software was designed for daily usage over a long period of time; therefore, when a decision about consistency *vs* usability occurred the software was developed for usability. Due to the slow speed of input when using a single switch, making the software usable equated to making the software fast and efficient. This decision has increased the initial learning curve for new users but will also increase long term user satisfaction.

2.4 Health Technology Consultancy Infra Red Device

Before the commencement of the studentship Health Technology Consultancy (HTC) had developed an Infra-Red Device (IRD) for use by patients. The IRD connects through a parallel port, and allows the computer to capture and send infra-red signals. The device was designed to be easy to construct, and as low cost as possible. All the components can be purchased from common electronics stores such as *Dick Smith* or *Altronics*. The circuit diagram for the IRD is going to be released on the Internet free of charge, and any person with some experience with electronics and soldering will be capable of constructing one.

Using the IRD receiver a computer can capture infra-red signals from a remote control and store them - this is called training the computer. Once trained, these signals can then be re-broadcast an infinite number of times. Training the computer to a remote control will be performed by a carer, relative, or friend; and then the users will re-transmit the signals as required.

In addition to infra-red control, the device has several computer controlled switches. These can be connected to a nurse call system, and then the patient can activate them using the computer. The IRD may contain up to 6 switches, each connected to a compatible device. The user could potentially control many more devices in their environment; however, this is beyond the scope of this project.

To test the IRD a demo application was written for Microsoft Windows. This application demonstrated the principle; however, it was not suitable for its intended purpose. As part of this project that application was re-written to be more flexible, configurable, and user friendly.

2.5 Linux Infrared Remote Control (LIRC)

The Linux Infrared Remote Control project (LIRC) (Scheibler and Bartelmus, 1999) is an application that allows a computer to capture and transmit infra-red signals using an IRD. Using LIRC the computer can be trained to a remote and then used to transmit the signals. LIRC is a large project with far more functionality than required by our system. The decision was made that creating a driver for LIRC to control the hardware developed by HTC (Health Technology Consultancy, 2005) would be more difficult and less effective than developing a simple tool from scratch. However, LIRC has been an excellent source of information throughout this project.

2.6 Objectives

The following objectives clearly define what the developer wanted to achieve as part of the studentship:

- Develop software to allow the simulation of a keyboard and pointer device using a single switch. This was achieved by the *Uno* applet.
- Develop software to control the IRD developed by HTC. The development of *ppuirc* met this objective.
- Develop a graphical utility to make controlling the IRD easy and efficient using the *Uno* applet. The *IC* application achieved this.
- Configuration of a computer to combine all the elements together. This was achieved with a *Linux* distribution.

Chapter 3

Approach and Methodology

This chapter discusses the approach and methodology used to develop *Expand*. This includes our goals for the finished system, design decisions, constraints of the project, and the method used to develop a system with multiple different complex components. This chapter begins with discussions about the licence for the software, and translating the software to languages other than English.

3.1 Open Source Software

Expand is potentially a very large and complex project. There is a tremendous number of features that could be incorporated for the user; therefore, it is not possible for a single person to develop this entire system to its full potential. *Expand* and all of its software and documentation must be made publicly available, so that others may continue the work that has been started.

Software that is distributed with its complete source code is called Open Source Software (OSS). A common licence for OSS is the GNU General Public Licence (GPL) (Free Software Foundation, 1991). The most significant features of the GPL are:

- All source code must be distributed along with the software.
- Users are free to modify and reverse engineer the software, with the requirement that they make their changes available to others licensed under the GPL.
- Users are free to copy and redistribute the software and source code.
- The source code does not need to be free of charge; however, it may be distributed free of charge by anyone who purchases the software.

The software developed for our system will be distributed under the GPL, as this licence is well suited to our goals. The GPL will allow other people to

continue its development, while protecting the copyright of the authors. Enthusiastic users will be able to improve the software by fixing bugs and adding new features.

3.2 Internationalisation

Western Australia is a diverse, multicultural state. It is likely that non-English speakers who live in WA — and many more who don't — will want to use *Expand*. Throughout the studentship the developer has attempted to make *Expand* as easy to translate as possible, within the time constraints. *Expand* does not follow *Gtk's* explicit tips and practices for internationalization; however, the developer believes that translating *Expand* would not be a difficult task. Due to time constraints and lack of resources no attempt at translating *Expand* has been made. This has been left for future work.

3.3 Design Goals

There are several design goals for *Expand* that have influenced its development. This section details those goals.

3.3.1 Old/Slow Systems

As already stated in Chapter 1, *Expand* is targeted at people with little or no income. It will be difficult or impossible for the target users to get access to new, fast computers. *Expand* is designed to run on old, second hand computers. Our hope is that government and private organisations will donate old computers for people to run this system.

3.3.2 Fast and adjustable

Initially the system will need to run slowly; however, with practise people can become extremely efficient with this type of system. If the users are not able to modify the system they will become frustrated very quickly. To prevent frustration all speeds and time-outs must be adjustable. As well as this, on older computers it is important to avoid long delays in the system. If there are unexpected delays in timing - for example, due to paging memory - the system will be difficult to use and frustrating for the users.

3.4 Reducing the number of features

It would be beneficial for the system to handle multiple switches for input. This would allow the user to input data twice as quickly or faster if the software were correctly written and configured. Due to time constraints it was not possible to include this feature. Our decision was to develop the interface to only handle

one switch; allowing as many people as possible to use the system. Designing *Expand* to use two switches would not increase the number of possible users.

3.5 Splitting the system into parts.

Due to the size and complexity of the system, *Expand* was broken up into modules. This was easily achieved as the objectives have clear levels of separation, each solved by a different module (see Section 2.6). This approach allows modules from *Expand* to be used in other projects, and new modules to be added to *Expand* in the future.

3.5.1 Uno applet

The goal of the uno applet was to create a software system to allow a user to control a computer using a single switch. A pointer device and a keyboard will be simulated using software, which can be controlled entirely using a single switch.

3.5.2 ppuirc

ppuirc is an acronym for Parallel Port Universal Infra Red Controller. This module is responsible for controlling the hardware developed by Health Technology Consultancy. This module was broken up into four small related modules: libppuirc, ppuirc-switch, ppuirc-receive, and ppuirc-send.

libppuirc

libppuirc provides a high level C API for the IRD which can be used to easily develop front end applications to control the IRD. As there are many uses for infra-red it is expected that others will develop applications using the IRC, that are not intended to be used with *Expand*. This C API will make it easy and efficient for other developers to do so.

For example, if a hospital has patients that have no disabilities but are required to stay in bed for medical reasons, this device can be used to control their environment. As these patients have complete use of their facilities (arms, hands, fingers, etc.) they will be using a traditional mouse and keyboard. Users using a mouse and a keyboard will want to use a different software interface than people using a single switch to control the computer. Software to accommodate hospital patients will be easy to write using *libppuirc*.

ppuirc-switch

ppuirc-switch is a command line application to activate a switch on an IRD. This application will allow the user to specify which device to use (if the user has multiple parallel ports and multiple IRDs), which switch to activate, and the number of seconds to turn the switch on. A command line application is useful

as it can be activated by a script, allowing multiple actions to be performed by a single command, or allow the switch operation to be automated.

ppuirc-receive

ppuirc-receive captures infra red signals and displays them to the console. The signal displayed is a series of times in milliseconds corresponding to switching the infra-red LED on or off. The list of times alternates between on and off, with the first time representing the LED switched on. All infra red signals start with the on position, else the receiver would not know when a signal had started. The end of the signal is marked with a 0. Multiple signals can be captured sequentially - but not simultaneously - with each signal separated by a blank line. *ppuirc-receive* is used to train the computer.

ppuirc-send

ppuirc-send is a command line application to send an infra red signal that has previously been trained. One advantage of a command line application is that several signals can be sent with one use action by placing the commands in a script. A command line application also allows the process to be automated.

3.5.3 Icon Commands

Icon Commands (IC) is a GUI application that allows the user to execute a console command by clicking on an icon. This application is designed as a graphical front end for *ppuirc-send* and *ppuirc-switch* (but not *ppuirc-receive*). This application will provide a simple and clear method for users to send IR signals and activate the switches on the IRD, as well as executing any other commands the user requires.

Chapter 4

Uno

This chapter details the *uno* applet, which is a core module of this system and consists of approximately 5500 lines of C++ code. *Uno* allows a person to completely control a computer using a single switch for input. This is achieved by simulating a keyboard and pointer device using software.

4.1 Warning

The *uno* applet is still under heavy development and is prone to crashing unexpectedly. **THE UNO APPLET SHOULD NOT BE USED TO CONTROL ANY CRITICAL SYSTEM.** For example, *uno* should not be used to drive the controls of a wheel chair. The *uno* applet comes with no warranty of any kind either expressed or implied.

4.2 Gnome Applet

Uno was developed as a *Gnome* applet. A *Gnome* applet is a small application that resides in a *Gnome* Panel (McLoughlin, 2005). A *Gnome* applet was used to minimise the amount of screen real estate used by *uno* and to prevent the application from covering other windows. *Gnome* applets require the developer to use the *Gtk* or *Gtkmm* library. *Gtkmm* was used for the reasons discussed in A.4.

4.3 Design

4.3.1 Actions

For the purposes of this document an action is an event triggered by the user, such as: a mouse click, a mouse move, a key click, etc. Button and key clicks can be broken down into: a down action (button/key press), and an up action



Figure 4.1: The icons for the actions that the user can perform.

(button/key release); however, clicks are usually far more useful than press and release actions on their own. To make this applet as easy to use and as fast as possible for the user, simple actions have been combined together to form useful complex actions. This has reduced the number of possible actions the user can perform, while greatly increasing the speed and efficiency of the actions that are available to the user. Each action the user is capable of is depicted as an icon in Figure 4.1, and a description of the icons from left to right is given below.

- No Action - does not perform any action if activated. Is used to reduce the required reaction time to operate *uno*.
- Left Click - performs a single left click at a user specified coordinate on the screen.
- Double Left Click - performs two left clicks in rapid succession at a user specified coordinate on the screen.
- Middle Click - performs a single middle click at a user specified coordinate on the screen.
- Right Click - performs a single right click at a user specified coordinate on the screen.
- Move - moves the mouse to a user specified coordinate on the screen.
- Drag - performs a left mouse button down action at a user specified coordinate, moves the mouse to a second user specified coordinate, and performs a left mouse button up action.
- Scroll Up - sends a mouse button 4 click to the active window. Button 4 is equivalent to scrolling the scroll wheel up on a mouse.
- Scroll Down - sends a mouse button 5 click to the active window. Button 5 is equivalent to scrolling the scroll wheel down on a mouse.
- Keyboard - activates the on-screen keyboard.

Icons

The different actions are represented using icons. They are stored using the PNG format (Roelofs, 2001) for the following reasons.

- PNG results in high levels of compression for images with large areas of one colour.
- PNG supports transparency.
- PNG is an open format (Roelofs, 2005).

Representing the actions as icons provides the greatest flexibility to the user. Examples of why the user may want this level of flexibility are given below:

- A user may want to customize the icons to symbols that they find more meaningful.
- The icons can be converted into any language (internationalisation).
- People who suffer from learning or recognition disabilities can convert these letters into symbols they can understand.
- A person with poor vision may modify the colour to make them easier to see.

The size of the icons can be adjusted to suit the user. This is useful for people with poor vision, allowing them to increase the size; while users with good vision can make the icons small to prevent valuable screen real estate from being wasted. The icons are drawn only in black and white so users who are colour blind can still see them. Users who are not colour blind may modify the colour to make the icons easily distinguishable.

Start / Stop cycle

The base state for the uno application is a sleep state where no operations are performed. This is to prevent anything distracting the user. When the user wants to perform an action they activate the switch to start the action selection sequence. The action selection sequence highlights each icon in turn, and when the user activates the switch the highlighted action is performed. The action selection sequence can be seen in Figure 4.2. If the user does not select any action the user input is assumed to be a mistake and the applet returns to the sleep state.

4.3.2 Pointer

Left Click, Double Left Click, Middle Click, and Right Click

These actions are very similar. They involve specifying two pieces of information to the X server:

1. An (x, y) coordinate of the screen.
2. The mouse button that is clicked (two clicks for a double click).

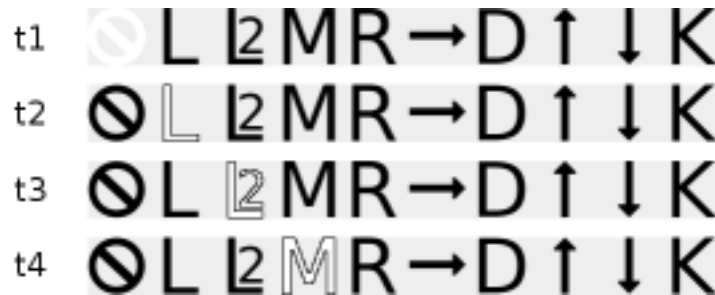


Figure 4.2: The uno applet cycling through the possible actions at four different times (t_1 to t_4).

The pointer button is specified using an integer - left (1), middle (2), right (3), scroll down (4), scroll up (5). The applet only needs to determine the (x, y) co-ordinate of the pointer as the button is specified by the action. To get the y coordinate a horizontal line is slowly translated down the screen as shown in Figure 4.3. When the user activates the switch the line is stopped and the y coordinate recorded. Then a vertical line is translated from left to right across the screen as shown in Figure 4.4. The vertical line determines the x coordinate and is recorded when the user activates the switch. The intersection of the two lines indicates the (x, y) coordinate and the mouse click action is sent to the X server. The process of performing a left click successfully is highlighted using the following use case:

Use Case: Left click.

Scenario: Successful.

Pre conditions: The applet must be running and in its default sleep state.

Use Case begins when the user activates the switch.

The applet highlights each icon in order from left to right.

The user activates the switch when the desired action is highlighted.

The applet starts the horizontal line scrolling slowly down the screen.

The user activates the switch when the horizontal line is at the correct location.

The applet starts the vertical line scrolling slowly across the screen from left to right.

The user activates the switch when the vertical line is at the correct location.

The applet moves the mouse to the appropriate (x, y) coordinate.

The applet sends the left mouse button down event to the X server.

The applet sends the left mouse button up event to the X server.

The applet returns to its default sleep state.

The use case ends when the applet has returned to its default sleep state.

Post conditions: the mouse has been moved to the specified (x, y) co-ordinate, a left click at that co-ordinate has been performed, and the applet has returned

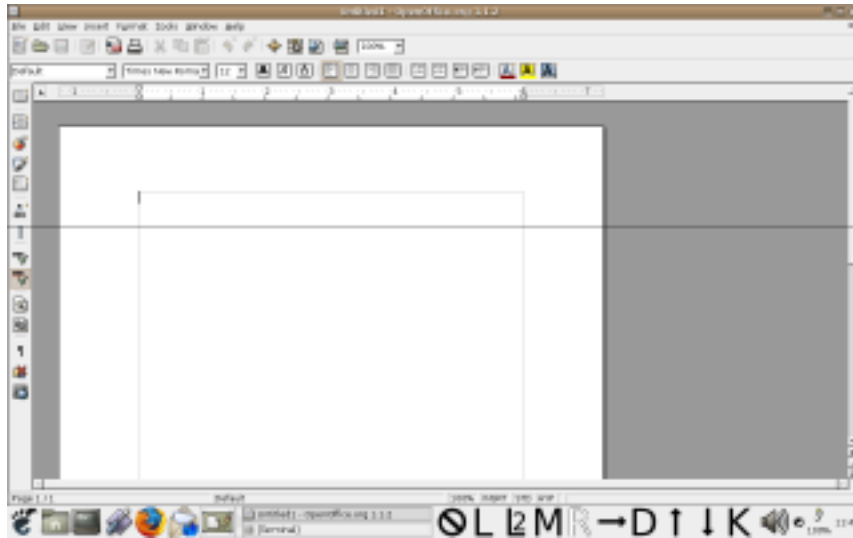


Figure 4.3: Finding the y coordinate to perform a mouse click at.

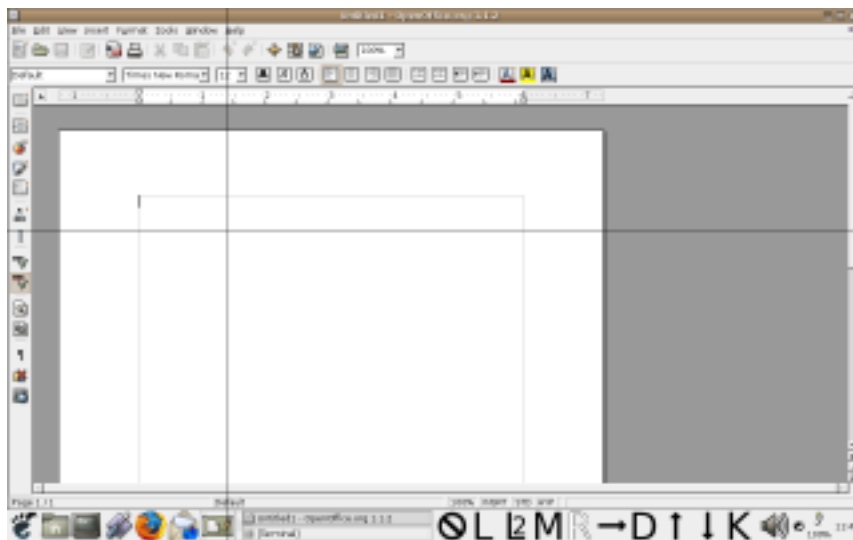


Figure 4.4: Finding the x coordinate to perform a mouse click at.

to its default sleep state.

This Use Case is slightly simplified. The applet will increase the speed of translation of both lines over a user specified distance and at a user specified acceleration. This allows the user to click areas of the top left hand side of the screen easily - with the lines moving at a slow speed - but without having very long delays if the user wants to click the bottom right of the screen. The speed and timing of every part of the applet is user configurable.

Underscanning is a feature of *uno* that can be enabled by the user. Underscanning allows the horizontal and vertical lines to translate faster without effecting the accuracy of the user. The maximum translation speed is increased significantly. When the line is approaching the target the user activates the switch, which reduces the speed of the line to a slow translation. When the line passes over the target the user activates the switch again and the line stops. The following use case highlights the process:

Use Case: Left click with underscanning activated.

Scenario: Successful.

Pre conditions: The applet must be running, in its default sleep state, and underscanning must be activated.

Use Case begins when the user activates the switch.

The applet highlights each icon in order from left to right.

The user activates the switch when the desired action is highlighted.

The applet starts the horizontal line scrolling quickly down the screen.

The user activates the switch when the horizontal line is approaching the target region.

The applet sets the translation speed to the slow pre-defined speed.

The user activates the switch when the horizontal line is at the correct location.

The applet starts the vertical line scrolling quickly across the screen from left to right.

The user activates the switch when the vertical line is approaching the target region.

The applet sets the translation speed to the slow pre-defined speed.

The user activates the switch when the vertical line is at the correct location.

The applet moves the mouse to the appropriate (x, y) coordinate.

The applet sends the left mouse button down event to the X server.

The applet sends the left mouse button up event to the X server.

The applet returns to its default sleep state.

The use case ends when the applet has returned to its default sleep state.

Post conditions: the mouse has been moved to the specified (x, y) co-ordinate, a left click at that co-ordinate has been performed, and the applet has returned to its default sleep state.

As with the version without underscanning, the speed of translation is accelerated over time. When the user activates the switch the new translation speed is the minimum of the pre-defined slow translation speed and the current translation speed.



Figure 4.5: The icons to send scroll up and scroll down events to the X server.

Move and Drag

Move and drag are not covered in-depth as the process is similar to performing a left click. The move action is used to move the mouse to a new (x, y) coordinate without performing a click action after the move. The drag action is used to move the mouse between two user specified points with the left mouse button down. Underscanning may be activated for both move and drag.

Scroll Up and Scroll Down

The scroll up and scroll down actions are used to simulate the scroll wheel on a mouse. The scroll-up and scroll-down icons are shown in Figure 4.5. Every time the scroll wheel on the mouse is moved a button click is sent to the X server; therefore, from a technical perspective simulating a scroll up or down event is similar to a specifying a left click event.

From the user's perspective there is considerable difference between a left click and the scroll wheel. A left click involves specifying the (x, y) co-ordinate with the mouse and then clicking the button. When generating scroll up and scroll down events the user is rarely concerned with the (x, y) co-ordinate of the mouse. The user simply wants the window containing the mouse to scroll up or down. Uno is designed to follow the users perspective of using a mouse; hence, uno does not get the (x, y) co-ordinate from the user when generating a scroll event. If the user needs to specify a different location for the event they can use the move mouse action before the scroll action. The scroll up and scroll down icons are shown in Figure 4.5. The following use case demonstrates the use of scroll up.

Use Case: Sending scroll up events.

Scenario: Successful.

Pre conditions: The applet must be running, in its default sleep state, and underscanning must be activated.

Use Case begins when the user activates the switch.

The applet highlights each icon in order from left to right.

The user activates the switch when the scroll up action is highlighted.

The user activates the switch repeatedly.

Every time the switch is activated a scroll up event is sent to the X server.

The user stops activating the event.

After a user defined number of seconds the applet times out.

The applet returns to its default sleep state.

The user case ends when the applet has reached the default sleep state..



Figure 4.6: The icon used to activate the on-screen keyboard.



Figure 4.7: The on-screen keyboard.

Post conditions: zero or more scroll up events have been sent to the window containing the mouse.

4.3.3 Keyboard

The on-screen keyboard is used to simulate input events from a physical keyboard. The keyboard icon is shown in Figure 4.6. An image of the on-screen keyboard is shown in Figure 4.7. The keyboard is displayed when the icon is activated by the switch. Once the keyboard is activated, the following use case is used to type one key.

Use Case: typing a single key in the keyboard.

Scenario: Successful.

Pre conditions: The keyboard must be activated and in its sleep state.

Use Case begins when the user activates the switch.

Each row of the keyboard is highlighted in order, from top to bottom.

When the row containing the letter the user wishes to type is highlighted the user activates the switch.

Each key on the selected row is highlighted in turn, from left to right.

When the key the user wishes to type is highlighted the user activates the switch.

A key down and key up event with the corresponding key code is sent to the X server.

The keyboard returns to its default sleep state.

The use case ends when the keyboard returns to its sleep state.

Post conditions: A key click event has been sent to the X server and the keyboard is in its default sleep state.

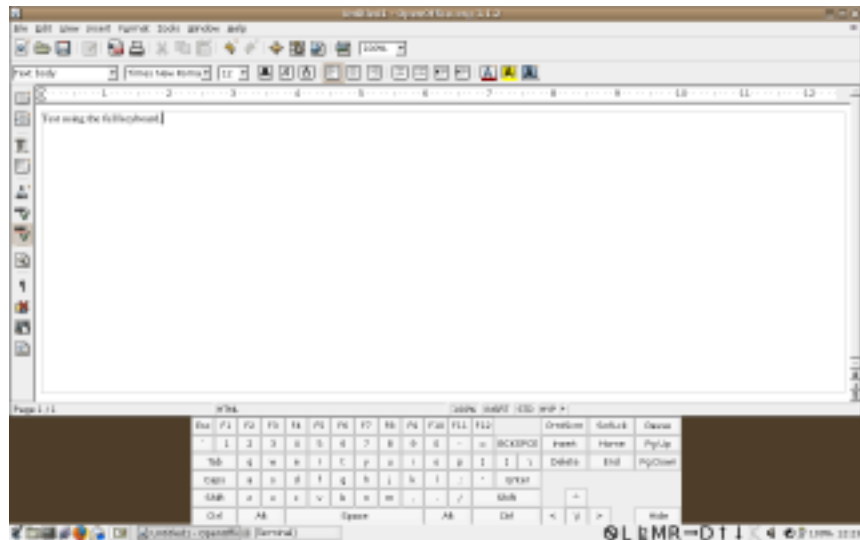


Figure 4.8: Screen shot of the on-screen keyboard.

Features

The keyboard is built using simple Gtk buttons - black text on a light background. Using colours would make the keyboard more visible for people with normal vision; however, it would be difficult or impossible for people who are colour blind to use. Using black text on a light background is usable by everyone.

To prevent the keyboard from covering important information on the screen, the keyboard notifies the window manager to reserve space for it. This re-sizes all maximised windows so that the keyboard is not covering them. This is shown in Figure 4.8.

When the user has finished with the keyboard there is a hide button in the bottom right corner that hides the keyboard and returns *uno* to its default sleep state - ready to perform pointer operations or bring the keyboard back again.

Layout

The layout of the keyboard is completely user configurable, and *uno* is designed to contain two keyboard layouts. The full keyboard layout shown in Figure 4.8 was designed with the following two goals in mind.

1. Display correctly on a screen with a resolution of 800×600 . This is to accommodate old computers that are only capable of low resolutions.
2. Contain as many keys as possible that are found on a traditional keyboard.

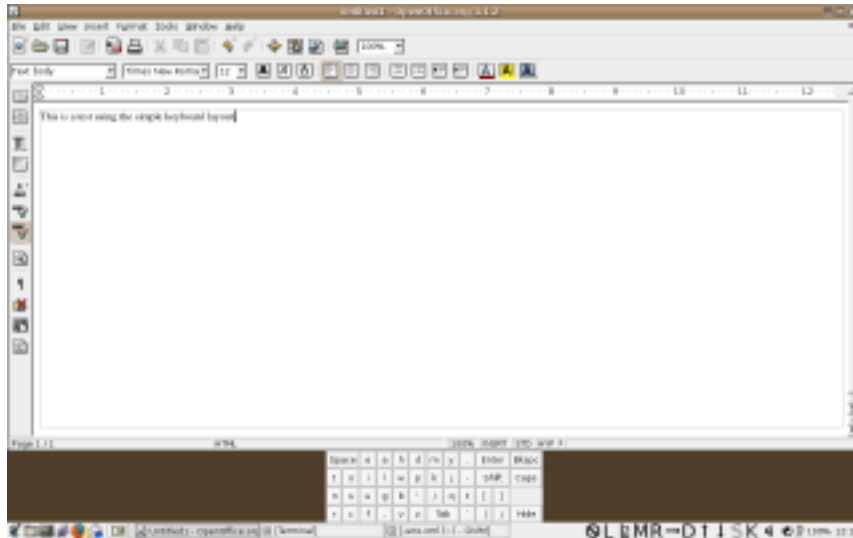


Figure 4.9: Screen shot of the on-screen keyboard with a frequency of use layout.

This keyboard would be used when the user needs to use rarely used keys such as the *F* keys. The layout shown in figure 4.9 was designed with the following two goals in mind.

1. Order the keys by frequency of use to increase the typing speed of the user.
2. Remove rarely used keys to reduce the amount of screen real estate used by the keyboard.

This keyboard will be used for the majority of keyboard input.

Features

Modifier keys

Modifier keys are keys that change the function of other keys on the keyboard, such as *Caps Lock*. For the uno applet, modifier keys were broken up into two types: one time keys and lock keys.

Lock keys are keys that are toggled on and off when the user clicks a key on the keyboard. The most common lock keys are Num Lock, Scroll Lock, and Caps Lock. When the user activates a lock key, the on-screen keyboard keeps the Gtk button pressed to highlight that the lock key is active, as shown in Figure 4.10. The Gtk button is released when the user activates the key again.

One time keys are keys that are held down to modify the function of other keys. The most common one time keys are Ctrl, Alt, and shift. When the user

Esc	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12		PrintScrn	ScrLck	Pause
`	1	2	3	4	5	6	7	8	9	0	-	=	BACKSPACE	Insert	Home	Pg Up
Tab	q	w	e	r	t	y	u	i	o	p	[]	\	Delete	End	Pg Down
Caps	a	s	d	f	g	h	j	k	l	;	'	Enter				
Shift	z	x	c	v	b	n	m	,	.	/	Shift		^			
Ctrl	Alt					Space			Alt	Ctrl	<	V	>			Hide

Figure 4.10: Selecting a key from the bottom row with scroll lock turned on.

Esc	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12		PrintScrn	ScrLck	Pause
-	!	@	#	\$	%	^	&	*	()	_	+	BACKSPACE	Insert	Home	Pg Up
Tab	Q	W	E	R	T	Y	U	I	O	P	[]		Delete	End	Pg Down
Caps	A	S	D	F	G	H	J	K	L	:	"	Enter				
Shift	Z	X	C	V	B	N	M	<	>	?	Shift		^			
Ctrl	Alt					Space			Alt	Ctrl	<	V	>			Hide

Figure 4.11: Keyboard layout with the shift key pressed.

activates a one time key the Gtk button is pressed to highlight that the key is active. When the user activates the next key the corresponding key down and key up events are sent to the server, and then all one time keys that are active are released.

Capital Letters

When the user presses either the *Shift* key or *Caps Lock* key the character generated by the keyboard is changed. The on-screen keyboard changes the text displayed in each key to a user defined alternate text as shown in Figure 4.11, when *Shift* or *Caps Lock* are activated. This is usually the upper case version of that key (*h* changes to *H*, *2* changes to *@*). The actual key code sent to the X server does not change. It is the X servers responsibility to interpret the key press and any modifiers.

4.3.4 Configuration Options

The on-screen keyboard is configured using *xml*, which controls the size and location of the keys using a grid. This allows for the size of the buttons to be altered; however, they can only be rectangular shapes. The grid layout is generated with a user defined number of rows and columns.

The text shown in the key - with or without the shift key pressed - is also configurable, although they may be identical. The number of keys is user specified. There is one special *hide* key that is used to exit from the keyboard back to the applet default sleep state.

The displayed text and the key code generated are user specified; therefore, it is possible that a keyboard could be constructed for a foreign language. However,

this is untested as the author only speaks one language.

Chapter 5

Icon Commands (IC)

5.1 Concept

Most applications on *Unix* operating systems start out as command line applications before a graphical version is developed. One reason for this is because developing command line applications is easier than developing graphical applications. A second reason is that command line applications can be automated using scripts. A final reason is that an experienced user can control the computer faster and more effectively using the command line.

However, typing commands using *uno* is slow and frustrating. The *Icon Commands (IC)* application was developed to allow a person using *uno* to execute common commands by clicking on icons. These commands will need to be typed only once into the configuration file and then can be executed with a minimum number of clicks.

IC is primarily designed to control the PPUIRC (see Section B); however, by using commands *IC* is extremely flexible and can be extended to perform other commands. It is also possible to run scripts using *IC*, automating common tasks for the user.

5.2 How it works

Figure 5.1 shows a typical layout of *IC*. The possible commands that the user can perform are grouped together. Each group of commands is stored in its own tab. Figure 5.1 contains three groups: *Switches* (the switches on the PPUIRC device), *VCR Controls* (sends infra-red signals to control a vcr), *Airconditioning* (Controls the airconditioner via infra-red). Each tab is identified via a label and an icon. This combination was chosen for the following reasons:

- Labels are simple and easy when adding new tabs.
- Labels can be read by people with colour blindness.



Figure 5.1: Screen shot of *IC*.

- Icons are useful for people with poor vision, as the icon size can be adjusted easily.
- Icons are useful for people who cannot read. Possibly due to learning disabilities.
- People can interpret icons faster than they interpret text.

Each button on a tab executes a command. The buttons have both a label and an icon for the same reasons as a tab. The icons shown in Figure 5.1 are used to control a VCR using *ppuirc-send* (see Section B.5).

5.3 Configuration

IC is configured using an xml file stored in the user's home area. The configuration file specifies the number of tabs, their labels, and their icons. The xml file also specifies every button's label, icon, and command. Each tab contains a user defined grid of rows and columns that is used to specify the size and shape of each button (the buttons must be rectangular). The GUI is constructed dynamically at run time from the xml file, making it entirely user controllable.

5.4 Gnome HIG

The *Gnome* HIG is written to make all *Gnome* applications consistent and easy to use (Benson *et al.*, 2002). They are written to cover all computer users. The purpose of *IC* is to assist a very small group of users - people running *uno* - to use the computer more efficiently. As the goals of *IC* and the *Gnome* HIG are vastly different, *IC* has not been written to follow the *Gnome* HIG. Instead every design decision for *IC* has been made with the target users in mind.

5.5 Extensibility

As described in Section 5.1, *IC* was written to be as extensible and flexible as possible. At the time of writing the main use of *IC* is to control the *PPUIRC*; however, one of our long term goals is to develop an API similar to *libppuirc* (see Section B.2) to control X10 compatible devices (Boone, 2004). This would allow the user to control a greater variety of devices such as dimmer lights, surveillance cameras, sirens, locks, and switch *on* and *off* any device that plugs into a power socket (Boone, 2004).

Chapter 6

Configuring the operating system and desktop

To use *Expand* easily and effectively the operating system must be configured correctly. This chapter discusses the modifications to a default *Gnome* installation that will assist the user.

6.1 Desktop

6.1.1 Background

The *uno* applet uses black, horizontal and vertical lines to find the (x, y) coordinate for button clicks from the user (See Section 4.3.2). To make these lines clearly visible to the user the background on the desktop should be a light colour, to contrast the black lines. For people with poor vision the background should be one solid colour, as gradients and images can make the lines harder to see. The colour should be modified to suit the user's aesthetic taste, and accommodate people with colour blindness. An example of using a light blue background is shown in Figure 6.1.

6.1.2 Icons

The *Gnome* menu provides access to many useful *Gnome* and *X Windows* applications. The icons are sorted into categories to make them easier to find and the menu easier to navigate. The menu structure is elegant and simple for the user to understand and use. Unfortunately, the *Gnome* menu requires many mouse operations to activate a single program. The following actions are required to start *OpenOffice Writer* (each item is equivalent to one action of the *uno* applet):

1. Left click on the *Gnome* menu icon.



Show Desktop

Hide application windows and show the desktop

Figure 6.2: The *Show Desktop* applet.

allowing users with good vision to minimise the amount of screen real estate wasted. The bottom panel in Figure 6.1 is 48 pixels high.

6.3 Auto Login

It is possible that more than one user will want to use *Expand* with the same computer. Due to time constraints, *Expand* does not have any functionality to log on a user; therefore, this project assumes that only one user will use the computer. A logon feature may be incorporated into a new version of *Expand*. Currently the computer must automatically login the user at boot time.

The *Gnome Display Daemon* (*GDM*) allows *root* to specify a user who will automatically be logged in (GDM, 2003). As *Gnome* supports sessions to automatically start applications each time the user logs in, *uno* will be automatically started every time the computer is switched on.

Chapter 7

Testing

Testing and validation are vital components of scientific research. *Expand* has only had a very small amount of testing due to the short time frame. The software was tested incrementally as each component was developed, to ensure that the software ran to specification. This phase of the testing was successful. *Expand* has been tested using a human trial. As it is possible for the software to meet its specification - and yet still fail to assist people to use a computer - the only way to make sure this software is helpful is to give it to people to try, and get their feedback.

7.1 The human trial

Ken is a relative of one of the members of this project, and lives at the Quadriplegic Centre in Shenton Park. Ken has no control over his arms or legs; however, he has complete control over his body from the neck upwards. He can look around, and speak, and uses a blow tube to call a nurse. Ken agreed to spend one morning testing the software.

7.2 Setup

The switch used was an air pressure switch with a blow tube attached, as this is the same type of switch Ken currently uses to call a nurse. The switch was connected using two wires to the game port on a sound card (Wikipedia, 2005), and forms a closed circuit when the user blows into the tube. *Uno* counted closing and opening the circuit as the switch being activated (see Section 4). The blow tube was mounted alongside Ken's nurse call, so that he could use both. Although *IC* can be used to signal for a nurse (See section 5), *IC* was not attached as the current nurse call system is more robust than *Expand*. When the *uno* applet has had widespread testing, then it may be suitable to replace the current nurse call system.

- When performing a *click* operation Ken started by looking at the top left hand corner of the screen. This is where the two bars originate - first the horizontal bar and then the vertical bar. However, Ken would often find himself looking at the top left corner when he should have been watching the *uno* applet, to select the action to perform. The solution is to place *uno* in the top-left hand corner of the screen. Then the user will always look to the top left corner when performing input. The motion on the screen - the actions being highlighted or the horizontal bar scrolling down - will attract the users attention and they will be immediately focused on the correct portion of the screen.
- If the user does not make a selection the software returns to its sleep state. When learning *Expand* each selection should repeat a user defined number of times, giving the user an increased number of opportunities to make a selection. However, if the user makes an incorrect selection they will need to wait longer before the applet will return to its default sleep state, so that the user can try again. Repeating the number of times the applet loops should be incorporated into selecting an action, inputting the *x* and *y* co-ordinates, and when using the keyboard.
- From the *X Server's* perspective a mouse click consists of two parts: a mouse button down event and a mouse button up event. When listening for mouse operations most applications respond to the button up event - as this is the end of the mouse click - hence, most applications are driven by the mouse up event. When using a blow tube, the *uno* applet ignores the point at which a user starts blowing (similar to the way most applications ignore mouse down events) and uses the time at which the user stops blowing to be the time when the switch is activated. This is unintuitive to the user. Consider the user inputting the *y* co-ordinate of a click action (the horizontal line is scrolling down the screen). It is intuitive for the user to begin blowing when the horizontal line is at the correct location. It is not intuitive for the user to stop blowing at this point. Although the time difference is usually very small, it can be significant for patients with poor muscle control - such as people with physical disabilities. The time difference becomes more significant as the user improves their skills and increases the speed of translation. *Uno* should be configured to listen for when the user starts blowing.

In addition to the changes just mentioned, Section 8.1 lists further improvements to the *Expand* system.

Chapter 8

Conclusion

As discussed in Section 3.1, the *Expand* system is potentially very large and complex. The studentship has been successful in developing a large amount of this system. The following milestones have been achieved:

- The development of an application to allow a single switch to completely control a computer (*uno*)
- The development of software to control the PPUIRC developed by HTC (*libppuirc*).
- The development of a simple method for the user to operate the PPUIRC (*IC*).
- The configuration of an old computer to operate *Expand*.
- A human trial to test the system.

Re-evaluating the goals from Section 1.5, the following has been achieved:

- Flexible and configurable - Virtually every part of this system is configurable. The user can modify the XML configuration files to change the behaviour of *Expand*.
- Extensible - *Expand* has been created as a series of modules. Adding new modules to the system or replacing existing modules will be an easy task.
- Low cost or free - All of the software developed and used is downloadable free of charge from the Internet. The hardware used by *Expand* can be purchased cheaply from a local electronics store.
- Small learning curve - The human trial demonstrated that the learning curve is still too high; however, the trial also highlighted ways in which to reduce the learning curve. There will need to be further trials to test the effectiveness of the changes.

8.1 Future Work

The following list outlines the work still to be completed on *Expand*. Some items are crucial and must be addressed before *Expand* can begin wide spread testing and usage, while others are beneficial to the user, but are not crucial. The items are not listed in any particular order.

uno

- Develop a settings dialogue to modify the applet behaviour at run time, and without the need to understand XML.
- Implement a reset feature which returns *uno* to a default configuration if the user makes a mistake in the settings.
- Implement dynamic resizing.
- Replace *X Windows* code with *Gtkmm*.
- Allow the keyboard text to be resized.
- Improve the method used to parse the configuration file.
- Allow the applet/keyboard to be at the top of the screen.
- Implement a dictionary for the keyboard.
- Implement a check for the existence of the configuration file.
- Increase the width of the horizontal/vertical lines as their translation speed increases.
- Allow multiple monitors.

IC

- Develop a dialogue to add and remove tabs.
- Develop a dialogue to add, remove, and modify buttons.
- Create an easy process to train a button to an infra-red signal.

Expand

- Improve the installer.
- Create packages for common *Linux* distributions.
- Perform more human trials.
- Create a *sourceforge* project to distribute the software and instructions (OSTG, 2005).

This project is far from complete; however, the studentship has made significant progress towards a complete system to assist people with severe physical disabilities. The human trial demonstrated the short comings of the system; however, it also showed the great potential for *Expand*.

Bibliography

- Benson, C., Elman, A., Nickell, S., and Robertson, C. (2002). Human interface guidelines (hig). Technical Report 2, Gnome.
- Boone, K. (2004). Using x10 for home automation. Web page. <http://www.kevinboone.com/x10.html> Last accessed on the 1st March 2005.
- Curtin (2003). Department of Computing. Web page. <http://www.computing.edu.au/> Last accessed on the 19th March 2005.
- Free Software Foundation (1991). Gnu general public license. Web page. <http://www.gnu.org/copyleft/gpl.html> Last accessed on the 29th March 2005.
- GDM (2003). *Configuring GDM*. Sun Microsystems. <http://docs.sun.com/app/docs/doc/817-3909/6mjfjvs3?a=view> Last accessed on the 1st March 2005.
- Gnome Project (2003). Gnome. Web page. <http://www.gnome.org/> Last accessed on the 1st March 2005.
- GOK (2001). Gnome onscreen keyboard. Web page. <http://www.gok.ca> Last accessed on the 13th March 2005.
- Hawking, S. (1988). *A brief history of time*. Barnes and Noble.
- Hawking, S. (2001). My experience with ALS. Web page. <http://www.hawking.org.uk/disable/dindex.html> Last accessed on the 6th March 2005.
- Health Technology Consultancy (2005). Health Technology Consultancy. Web page. <http://www.htcsa.com.au/> Last accessed on the 5th March 2005.
- Johnson, A., de Vienne, C., and Cumming, M. (2005). libxml++. Web page. <http://libxmlplusplus.sourceforge.net/> Last accessed on the 1st March 2005.
- Koffman, E. and Wolz, U. (1999). *Problem Solving with Java*. Addison Wesley.
- MacKenzie, D. (2003). *chmod manual page (man page)*. Free Software Foundation. Use *man chmod* to access.

- McHale, S. (2004). 2004/5 disability budget. Web page. <http://www.dsc.wa.gov.au/cproot/877/2/BBulletin2004%20.pdf> Last accessed on the 17th September 2004.
- McLoughlin, M. (2005). Panel applet writer's reference manual. Web page. <http://developer.gnome.org/doc/API/2.0/panel-applet/libpanel-applet.html> Last accessed on the 28th March 2005.
- Microsoft (2000). On-Screen Keyboard overview. Web page. <http://www.microsoft.com/enable/training/windowsxp/usingkeyboard.aspx> Last accessed on the 13th March 2005.
- Nice (2001). *nice manual page (man page)*. Free Software Foundation. Use *man nice* to access.
- Nisbet, P. and Poon, P. (1998). Special access technology. Web page. http://callcentre.education.ed.ac.uk/About_CALL/Publications_CAA/Books_CAB/SAT_CAC/sat_cac.html#download Last accessed on the 6th March 2005.
- Office of Science and Innovation (2005a). Office of Science and Innovation (OSI). Web page. <http://www.scienceandinnovation.dpc.wa.gov.au/index.cfm?fuseaction=home.welcome> Last accessed on the 5th March 2005.
- Office of Science and Innovation (2005b). Studentship award. Web page. <http://www.scienceandinnovation.dpc.wa.gov.au/index.cfm?fuseaction=studentships.main> Last accessed on the 5th March 2005.
- OSTG (2005). Source Forge. Web page. <http://sourceforge.net/> Last accessed on the 28th March 2005.
- Roelofs, G. (2001). libpng.org. Web page. <http://www.libpng.org/> Last accessed on the 1st March 2005.
- Roelofs, G. (2005). Portable network graphics. Web page. <http://www.libpng.org/pub/png/#history> Last accessed on the 1st March 2005.
- Scheibler, K. and Bartelmus, C. (1999). LIRC. Web page. <http://www.lirc.org/faq.html> Last accessed on the 1st March 2005.
- Sleep (1993). *sleep manual page (man page)*. Free Software Foundation. Use *man sleep* to access.
- Van Heesch, D. (1997). Doxygen. Web page. <http://www.stack.nl/~dimitri/doxygen/> Last accessed on the 1st March 2005.
- Veillard, D. (2005). libxml - the xml c parser and toolkit of gnome. Web page. <http://www.xmlsoft.org/> Last accessed on the 1st March 2005.

- W3C (1996). Extensible markup language. Web page. <http://www.w3c.org/XML/> Last accessed on the 1st March 2005.
- Wacom (2004). WACOM Asia Pacific. Web page. <http://www.wacom-asia.com/> Last accessed on the 5th March 2005.
- Western Australian Government (1998). The Government of Western Australia. Web page. <http://www.wa.gov.au/> Last accessed on the 5th March 2005.
- Wikipedia (2005). Game port. Web page. http://en.wikipedia.org/wiki/Game_port Last accessed on the 5th March 2005.
- Words+ (2004). Leader in augmentative communication devices. Web page. <http://www.words-plus.com/> Last accessed on the 6th March 2005.
- X.org Foundation (2005). X Windows. Web page. <http://www.x.org/> Last accessed on the 1st March 2005.

Appendix A

Tools

This appendix introduces the different software tools used to build *Expand*, and the reasons for using each tool.

A.1 Linux

Firstly, *Linux* can be downloaded free of charge from the Internet, making *Linux* the cheapest operating system available. This is ideal for people with little or no income. Secondly, *Linux* is licensed under the GPL, which is the same licence used for this project. Users can download the source code for *Linux* and modify the operating system to suit their needs. Finally, there is a version of *Gnome* for *Linux* (See Section A.3).

A.2 Language

This system will need to run on older computers and yet still be fast. This constraint means that scripting languages or partially compiled languages will not be acceptable. The time spent on parsing and interpreting, or running a virtual machine, will use precious resources that could be used more effectively elsewhere; hence, the language must be fully compiled.

A.2.1 C

C has excellent support under *Linux*. The majority of *Linux* APIs are written using C, which can then have bindings to other languages made. These were the reasons for developing *libppuirc* in C. *Ppuirc-send*, *ppuirc-receive*, and *ppuirc-switch* would not benefit from Object Orientation; hence, these three applications were written in C so that they may use *libppuirc* directly.

A.2.2 C++

As demonstrated by Java (Koffman and Wolz, 1999), GUIs can be modelled effectively and cleanly using an Object Orientated approach. C++ is an object orientated language similar to Java; however, it is compiled into native executables making them faster and more efficient than Java applications. C++ has been designed to be a fast and efficient language like C, while including the object oriented paradigm. *IC* and *uno* were developed using C++.

A.3 Gnome

Gnome (Gnome Project, 2003) is a desktop environment for *X Windows* (X.org Foundation, 2005). There are many desktop environments available; however, the two most prevalent are *Gnome* and *KDE*. There are no major reasons for choosing either *KDE* or *Gnome*. As the developer is more familiar with *Gnome*, that was the desktop environment selected.

Both *Gnome* and *KDE* suffer from hogging resources. Large amounts of memory and processor time are used when running either of them. This is a problem as *Expand* is designed to run on older computers. The use of a smaller desktop environment such as *XFCE* would have been preferable. The decision made was to develop for *Gnome* initially as this is what the developer is familiar with, and then port *Expand* to another desktop environment once the project was working. Porting *Expand* has been left as future work.

The decision to use an applet for *uno* was to make the application as unobtrusive as possible. By placing *uno* on a panel the amount of screen real estate consumed is reduced. This is important for users with poor eyesight who need to increase the size of the icons.

A.4 Gtkmm

The Gnome Tool Kit (Gtk) is a library used to develop GUI applications. This is the tool kit on which Gnome is built. As the desktop environment is *Gnome*, it made sense to use *Gtk* to avoid incompatibilities. *Gtkmm* is a set of C++ bindings for *Gtk*. As *uno* and *IC* were developed using C++, it made a cleaner design and implementation to use *Gtkmm* over *Gtk*. *Gtk* was also selected as the developer is familiar with this library, having used it for previous projects.

A.5 Libpanelapplet

Libpanelapplet is the library used to develop a Gnome Panel Applet. *Libpanelappletmm* is a set of C++ bindings for *libpanelapplet*. *Libpanelappletmm* was the logical choice to keep the code object orientated.

A.6 XML

The eXtensible Markup Language or XML, is a mark-up language designed to help define and represent data (W3C, 1996). The following benefits of XML directly relate to this project:

- Human readable - users can directly modify the contents of an XML document.
- Portable - XML files can be copied between platforms.
- Robust - XML can be validated against a DTD (a template).

For these reasons the configuration files for *uno* and *IC* were written in XML. Users may modify these files by hand to change the settings of the applications. The configuration files are parsed and validated at run time.

Libxml (Veillard, 2005) is an XML library written in C to handle the reading, writing, and validation of XML documents. *Libxml* is licensed under a GPL compatible licence allowing it to be modified and distributed free of charge (along with its source code). This fits in with *Expand* which is licensed under the GPL. *Libxml++* (Johnson *et al.*, 2005) is the C++ bindings for libxml.

A.7 Doxygen

Doxygen (Van Heesch, 1997) is a C and C++ documentation library. All source code written for *Expand* was documented using *Doxygen*. This was to make it easier for others to expand and modify *Expand*.

Appendix B

Parallel Port Universal Infra-red Controller (PPUIRC)

The *PPUIRC* device developed by Health Technologies was discussed in Chapter 2. This chapter discusses the design and implementation of the software that drives the *PPUIRC*.

B.1 Design

The software to control the PPUIRC is designed to be flexible and modular for both developers and users. The actual code to control the device is in a shared library *libppuirc*. A library was created to control the hardware so that developers may add functionality to their projects easily. There are three command line applications - *ppuirc-switch*, *ppuirc-receive*, and *ppuirc-send* - which utilise *libppuirc* to control the hardware. These command line utilities were developed because they can be incorporated into other projects - such as *IC* - and allow automation.

B.2 libppuirc

libppuirc is a user space driver to control the PPUIRC. This driver could have been developed in kernel space - and would probably been more efficient - however, this would have considerably increased the complexity and difficulty of the project. Due to the short 10 week time frame to complete the project, the decision was made to develop a user space driver. As multiple applications on the same system will use *libppuirc* it was compiled as a shared library.

Libppuirc allows developers to activate any of the 6 switches (the reference model used for testing only contained two switches; however, the PPUIRC could

contain up to 6 switches). It is anticipated that switches will be activated for several seconds and sub-second timing will not be required. There is a function call that caters specifically for this purpose.

To train the computer about an infra-red remote *libppuirc* contains functionality to capture signals. An infra-red signal is a digital signal with the infra-red light being either *on* or *off*. It is illogical for a signal to start with the light beam off; hence, the start of the signal is when the light is switched on. The captured signal is a series of decimal numbers representing the timing of the signal in milliseconds. The first number is the length of time the light should be switched *on* for. The second number is the length of time the light should be switched *off* for. This *on/off* pattern oscillates until the light is switched *off* at the end of the signal. The captured signal is delimited by a 0.

The send function of *libppuirc* takes a signal captured by the receive function - a zero delimited set of numbers with the first number representing the light on - and emits it through PPUIRC.

As discussed in Section A.7, the documentation for this library is created using *Doxygen* (Van Heesch, 1997).

B.3 ppuirc-switch

Ppuirc-switch is used to activate any of the 6 switches for a specified number of seconds. A user may use a script with this command to activate several switches at once. This is more useful as the number of switches on the PPUIRC increases.

B.4 ppuirc-receive

Ppuirc-receive is used to capture an infra-red signal. As infra-red signals are usually accurate to within a few microseconds it is important that context switches are avoided; hence, using signal driven I/O or select is impractical (Sleep, 1993). To avoid calls that put the process to sleep polling inside a *while loop* is used, resulting in excessive CPU utilisation and poor system performance. As the signal is only captured once this is an acceptable solution. The *LIRC* project uses the same approach for the parallel port (Scheibler and Bartelmus, 1999).

To prevent context switches from occurring the process must have a very high priority. To increase the priority of a process the *nice* system call is used; however, to raise the priority of a process requires super user privileges (Nice, 2001). To allow any user to run *ppuirc-receive* as super user the SUID bit must be set using the *chmod* command (MacKenzie, 2003). This is done as part of the installation. By turning on the SUID bit *ppuirc-receive* will run as the owner of the file (the *root* user).

B.5 ppuirc-send

Ppuirc-send is similar to *ppuirc-recv*. The critical timing required to capture the signal is also required to send the signal. The same approach is used and the same excessive CPU usage results. As it only takes a few milliseconds to send the signal this is acceptable. This is the solution used by LIRC (Scheibler and Bartelmus, 1999).