Lightweight User Interfaces for Watch Based Displays

Peter Hutterer[†], Mark T. Smith[‡], Bruce H. Thomas[†], Wayne Piekarski[†], and John Ankcorn[‡]

[†]Wearable Computer Lab
School of Computer and Information Science
University of South Australia
Mawson Lakes, Adelaide, SA, 5095, Australia

office@who-t.net
bruce.thomas@unisa.edu.au
wayne@cs.unisa.edu.au

[‡]Hewlett Packard Labs
Hewlett Packard Corporation
1501 Page Mill Rd
Palo Alto, Ca 94304, United States of America

msmith@hpl.hp.com
jca@hp.com

Abstract

Ubiquitous mobile computing devices offer the opportunity to provide easy access to a rich set of information sources. Placing the display for this computing device on the user's wrist allows for quick, easy, and pervasive access to this information. In this paper we describe a user interface model and a set of five applications we have developed, with the aim of providing a user interface that supports lightweight interactions. Our goal is to make our pervasive watch as simple to use as a common wrist-watch worn today.

Keywords: Ubiquitous computing, user interfaces, watch based computing, mobile computing.

1 Introduction

Watches have been around us for several centuries. In the 19th century, the watch gradually moved from the pocket to the wrist, thus being visible at all times (Martin 2002). Digital watches appeared with additional functions such as a stop-watch and alarms. There are a number of esoteric watches that provide features such as compasses, phone books, and GPS, but the purpose of the watch is still mainly (and often solely) to show the time.

Several years ago Personal Digital Assistant (PDA) technologies became readily available. They have become quite popular and are widely used in the business domain to improve personal organisation. These PDA's already have more processor performance than a desktop PC had just a few years ago. The applications they support are not limited to the Personal Information Management (PIM) domain. Video playback, office applications, and games are standard nowadays, and even telephony and Augmented Reality (Geiger, Kleinnjohann et al. 2002) is possible with the newest models. PDAs are designed to be carried in a pocket of the user's clothing, and removed from the pocket and held in the hand for operation. Recently, a number of commercial watches have started to implement some PDA functionality on the wrist, but these platforms are quite limited.

In this paper we tried to answer the question – "Now that Bluetooth allows your coffee maker to communicate to your watch - what would they talk about?" The answer we came up with is – "Who is going to tell Bruce his coffee is going to be twenty minutes late this morning?" One solution to this question is to leave the computing in the user's pocket and bring only the display onto the wrist. We have constructed a prototype for a new watch which contains only a display, limited processing capability, and (in the current version) no input devices. Bluetooth is used to communicate with an external device which we call a Personal Server (PerServ) that performs the processing. Any future input devices on the watch will be processed using the PerServ rather than the watch itself. The current prototype is shown in Figure 1, which is at the desired size (the microcontroller and Bluetooth modules are mounted under the display) but still needs to be packaged into a watch-style casing.

For our watch architecture, we leverage the power of handheld computers without having to limit our capabilities to a processor that can fit inside a watch housing. This paper describes our Personal Server architecture - a PerServ is an application operating on an additional hardware device to offload most of the watch's processing tasks. The PerServ is responsible for all complex program logic, and the watch device only receives, decodes, and displays raw image data. Connections to external data sources such as the Internet, other computers, or even household devices are always between the PerServ and the data source, and not the watch device itself. The watch is only connected to the PerServ, and even external input devices connect to the PerServ instead of the watch. Figure 2 shows an illustration of this concept. From this separation the following advantages are expected:

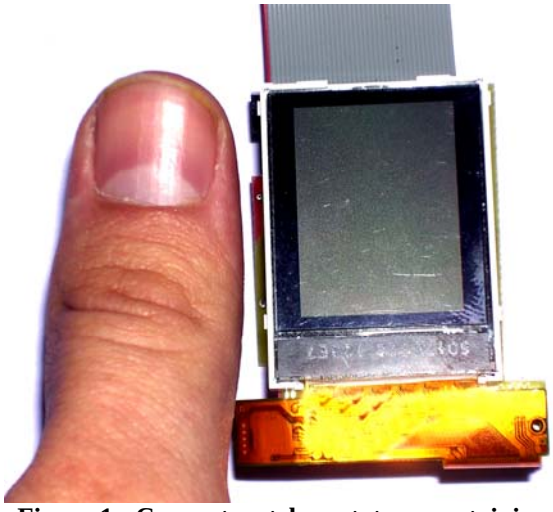

Figure 1 - Current watch prototype containing an LCD screen, MSP430 microcontroller, Bluetooth radio, and a temporary debugging cable.

1) Less processing power is needed on the watch, and therefore reduced energy requirements while the software is running.
2) The reduced computation allows simpler, more energy-saving, and cheaper hardware to be used for the watch.
3) By offloading tasks to a more powerful device, faster response can be given for complex applications.
4) Increased range and higher bandwidth for connectivity to the Internet and other devices.
5) A simpler interface for application programmers to use the watch as an output device.
6) A simpler way to install applications that can be used on the watch.
7) A separate and more powerful device would allow more sophisticated applications to be operated than is possible on the watch hardware alone.

In (Uemukai, Hara et al. 2002), a concept is described where a mobile screenless device makes use of brainless ubiquitous displays for displaying data. Our PerServ concept is similar to Uemukai et al., except that the remote display is carried with the user all the time. The concept can also be compared to the operation of SunRay terminals where they are dumb terminals that only process user input and screen output (Sun Microsystems 2004). The SunRay terminal receives simple frame buffer updates over a network, but the actual processing of X11 based applications is performed by X server "clients" running on a machine that services a set of machines. Our PerServ concept prepares the image data for the watch and sends it over to the watch for display only, which makes it a simplified implementation of the SunRay architecture.

The remainder of this paper starts with a description of the latest generation of watch technologies. This is followed by a more in depth explanation of the PerServ architecture and user interface framework. Five applications are presented to show different features of the PerServ and the user interface models. The paper finishes with some concluding remarks and a description of future work.

2 Advanced Watch Technology

Several projects have attempted to replace the watch with a multi-functional computing device. Research work has also been performed in the measuring of the social weight of a watch device combined with a PDA (Toney, Mulley et al. 2003). "Social weight" measures the impact the use of an item has on social interaction between two or more people. Additionally, other authors (Narayanaswami, Raghunath et al. 2001) ask the question of what applications an all purpose computer-in-a-watch could be used for. Their list of applications ranges from simple time keeping to health monitoring and games. In this section, we will give a short overview of some of the watches on the market and their important features. This is a selection of complex computerised watches that are indicative of the various design philosophies currently in use. The current limitations of these watches are then discussed.

2.1 Commercially Available Watches

Of the six different watches discussed in this section, five are commercially available while the IBM Linux Wristwatch is a research prototype. All six watches have slightly different functionality, and Table 1 depicts a number of the different features provided.

2.1.1 MatsuCom OnHand PC

The Matsucom OnHand PC was introduced in 1999 (Matsucom 2004) and has a monochromatic display of 102x64 pixels. A set of over 30 programs is preinstalled, containing different clock faces, a world clock, PIM features, and even games. The OnHand PC runs a custom operating system named W-PC-DOS and provides a rich API for programmers. Synchronisation is implemented using a serial cradle and specialised software, and wireless connections are possible via an infrared interface. Seiko also manufactures the OnHand PC under the name "Ruputer".

2.1.2 IBM Linux Wristwatch

A major goal of the IBM Linux Wristwatch project (Narayanaswami, Kamijoh et al. 2002) was to build a wrist computer with a standard operating system instead of one that uses custom developed low-level code. The final design incorporated a Linux kernel which was ported to the StrongARM processor and watch hardware. The graphical interface is rendered using a standard X-Windows server (Scheifler and Gettys 1986) which is commonly used on Linux platforms. Modifications were made to the X server, kernel, and support libraries to fit the code within the watch's limited memory and storage resources. There are two different versions of the watch: first version has a 96x120 LCD in portrait format, and the second version a VGA OLED in landscape format. Both versions have touch-screens divided into four parts, acting as button and jog-wheel input devices. Bluetooth and infrared interfaces are provided for connectivity, and a microphone and a small speaker are used for audio input and output.

2.1.3 Fossil Wrist PDA

Fossil's Wrist PDA is a small computer based on PalmOS 4.1, and therefore the PIM applications are compatible with standard Palm PDAs. The interaction with the

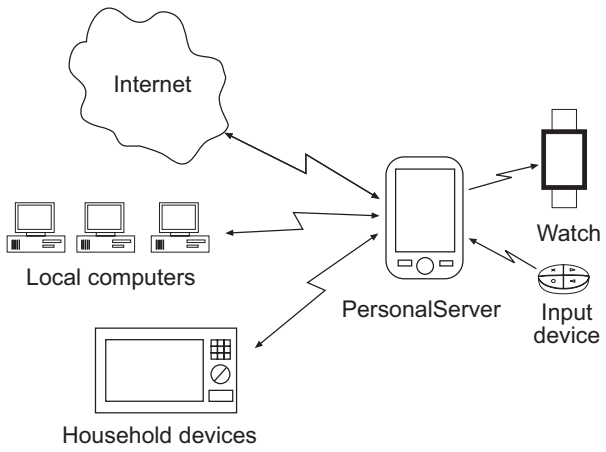

Figure 2. An overview of the Personal Server design, showing our separation concept.

device is performed over a 1" touch screen display with a stylus, in a similar way to normal-sized PDAs. Data may be downloaded onto the watch via an infrared connection for synchronisation.

2.1.4 MSN Wrist Net

The MSN Wrist Net approach is quite different from the previously described watches because very little processing is performed on the watch. These watches are a part of Microsoft's Smart Personal Object Technology (SPOT) initiative (Microsoft-Research 2004). The goal is not to have a miniature computer on the wrist, but instead more of a digital equivalent of a radio. The watch is able to have information sent to it via FM radio waves from preconfigured channels, but no other connectivity is built into the watch. Since the information is broadcast to all watches and is filtered for the user, it is not completely personalised. Limited personalised information is available via receiving MSN Messenger messages and schedules over a plug-in for Microsoft Outlook. Due to the radio broadcast transmission method used, the delivery of the messages cannot be guaranteed to be reliable or in real-time.

2.1.5 Timex USB Datalink

The Timex Datalink tries to form a balance between a sports watch and a PDA (Timex Corporation 2004). It provides standard time keeping functions such as stop watches, alarms, and different time zones, but also the possibility of saving personal schedules and telephone numbers. The display of the Datalink is not a purely dot matrix based LCD like the previously mentioned watches, but instead contains a seven segment display and two dot matrices (11x5 and 42x11 pixels). Synchronisation is performed via USB using either custom software or a Microsoft Outlook interface. When the watch is being worn on the wrist, there is no synchronisation possible because no wireless interface exists. Editing schedules directly on the watch is not possible with the limited user interface and processing capabilities available.

Vendor	*Matsucom*	*IBM*	*Fossil*
Device	OnHand Computer	Linux Wristwatch	Wrist Net
Battery life	2 days	2 hours	min. 2 days
API	C (proprietary)	C (Linux)	n/a
Range	infrared	ca. 10 m	Radio based
Buttons	4	4 touchscreen	5
Other input	Joystick	n/a	n/a

Vendor	*Fossil*	*Timex*	*Field Tech*
Device	WristPDA	Data Link USB	Smart Watch
Battery life	4 to 5 days	2 years [2]	n/a
API	PalmOS API	Assembler	n/a
Range	infrared	n/a	Infrared
Buttons	3	3	5
Other input	rocker wheel & touchscreen	n/a	n/a

Table 1. Commercially available watches comparison.

2.1.6 Field Technology CxMP Smart Watch

The Smart Watch manufactured by Field Technology CxMP contains a 256 colour LCD with 72x64 pixels and comes in several different colours for the youth market (Field Technology CxMP Ltd. 2004). The basic software configuration consists of a scheduler, timer, stopwatch, image viewer, and melody player. The watch can be synchronised using the provided software but does not interface to existing applications. Synchronisation is only performed using a serial cable and is therefore not possible while wearing the watch on the wrist.

2.2 Limitations of current watch designs

This section has introduced several of the latest watch devices available, but all of them still contain a number of limitations that need to be overcome before this technology will become practical. This subsection will analyse these problems, and Table 1 compares the features of all six devices.

A major limitation of these watches is the high power consumption and the constant dependency on fresh batteries. Apart from the Timex Datalink USB, the life of the battery is typically measured in hours, especially when the watch is being actively used. While graphical user interfaces and operating systems are nice features, they consume a lot of power to operate. With short operating times, these watches must be recharged regularly to remain operational. The user has to make an effort to ensure that they recharge the watch often, similar to recharging PDAs and mobile phones. The Timex USB Datalink watch is unique in that it will operate for two years with one battery, but this is at the cost of much more limited functionality with a less powerful processor.

The speed of the processor in the watch is a major contributor to the amount of power consumed from the battery. By reducing the clock speed, battery power can be saved, but this comes at the expense of functionality. Specific CPU designs are also more power efficient than others, and so selecting the correct platform is very important. For certain operations like reading in large streams of network data or processing video, some CPUs may not be adequate for this and therefore this functionality will not be possible. Using a more general purpose CPU such as in the IBM watch ensures that standard programming tools can be used, while more power efficient designs tend to require specialised APIs or assembly language to use.

The design of the user interface should have power usage in mind as well - power can be saved by keeping the number of lit up pixels as low as possible. The developers of the IBM watch (Narayanswami and Raghunath 2000) describe a number of considerations such as different fonts and layouts which can help to improve power usage. In *Energy Trade-offs in the IBM Wristwatch Computer* (Kamijoh, Inoue et al. 2001) several conventional watch faces are listed to show the relation of pixels in use compared to the battery usage of the display. The manual for the Fossil Wrist Net also states that “simple watch faces [...] take less power than elaborate watch faces, such as those that use animation” (Fossil 2004). The Smart Watch saves power by disabling the colour display after a fixed timeout, but this does not allow the watch to

be viewed. The biggest power savings can be made by removing the need for a back light. Reflective and transflective LCD displays exploit ambient light for pixel illumination, while backlit and LED displays actively emit light which allows the use of the display in darkness, but can consume a large amount of power.

Providing an interface so a watch can share data with other devices is a difficult problem. The use of serial cables or infrared makes the device difficult to connect to while it is being worn by the user. The use of network technology such as Bluetooth allows truly wireless capability to a distance of a couple of metres, although with the associated extra power drain. Maintaining a permanent Internet connection over Bluetooth remains difficult however, with a mobile user moving in and out of the short range available.

Interaction with watch devices is quite similar amongst all the available designs, mimicking standard watches with buttons mounted on the outside. The Matsucom and Fossil watches implement extra interfaces such as small joysticks or rocker switches for scrolling through information. The IBM watch implements a touch screen for button-like functionality, and the Fossil watch supports a stylus although it is difficult to operate in a limited area. A good alternative for input would be speech recognition (Narayanaswami, Kamijoh et al. 2002), although with current technology it is not possible to implement this on a low powered device yet.

3 Custom Watch Hardware

For our research, we have developed our own custom hardware for use as a watch, and use a standard HP iPAQ as the PerServ platform. This section describes this hardware in detail. This watch hardware forms part of the architecture overview shown in Figure 2.

3.1 The PerServ Device

The PerServ software operates on an iPAQ h5450 running either Linux or PocketPC 2002, and it has enough computing power along with the ability to support wireless technologies such as Bluetooth and WLAN. Since the operating system abstracts the hardware, we were able to implement the PerServ in a way that was mostly hardware independent. This enables us to port the same software onto a number of different devices in the future. Figure 3 depicts how applications are insulated from the hardware on the PerServ device. The application uses a plug-in API that is tightly connected with the PerServ framework. Access to the watch hardware is performed by the OS on the device. The PDA provides all input processing functionality for this version of the watch, since the watch itself is a pure display device and does no processing of its own. In the future we plan to add simple button and finger mouse controls (Hans and Smith 2003). These input devices will communicate directly to the PerServ via the Bluetooth module, and the PerServ then sends updates to the display.

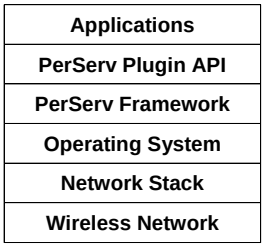

Figure 3. The PerServ system architecture.

3.2 The Watch Device

As previously mentioned, we have designed our architecture so that a less powerful processor is required on the watch, which will result in considerable energy savings. Our goal is to allow two weeks of operation without the need to recharge the batteries. The microcontroller used in our watch prototype is a Texas Instruments MSP430. The primary reason for this choice was its low power modes and ease of availability. We employed the F1491 model that contains 64 Kb ROM for program code and 2 Kb RAM for dynamic variables (Texas Instruments 2004). Additionally, it has two serial Universal Asynchronous Receiver-Transmitter (UARTs) for connecting to the Bluetooth chip and to the display. The 16 bit processor operating at 4 MHz needs only 280 µA in active mode, making it a very low powered device. The MSP430 is programmable with either assembly language or C. The watch and PerServ system are able to perform complex tasks via a “parasitic computing” (Narayanaswami, Raghunath et al. 2001) paradigm.

One of the goals of our project is to allow some form of video streaming through the watch, and although the watch CPU is not capable of performing complex decompression, the PerServ is and so it can transmit optimally encoded frames to the watch for display. The display used on the prototype is an Epson L2F50176T00 LCD, shown in Figure 1. The display's resolution is 120x160 pixels with a screen size of 2 x 2.6 cm (1.3” diagonal), or approximately 150 DPI. The LCD supports RGB colour and our current implementation supports either one bit monochrome or 16 bit colour.

For communication between the watch and the PerServ, we employed the Mitsumi WML-C09NBR Bluetooth chipset without an antenna. Like the LCD, the Bluetooth chip is connected over a UART interface with the maximum wireless data rate being 721 kbps (Mitsumi 2004). The WML-C09NBR is a class 2 chip, allowing a range of up to 10 m, but we found through testing with a standard USB Bluetooth adapter the maximum distance was just three to four meters for reliable operation. One important goal of our project is to connect the watch to the Internet. By using a PerServ, technologies like WLAN may be used indirectly. The typical transmission power of WLAN is 30 – 50 mW (Golem 2004), which is considerable for a purely wrist based device. Since Bluetooth only needs 2.5 mW to operate (Bluetooth SIG 2004), it is much more suitable for miniaturised devices. By splitting the functionality over two devices, Bluetooth can be used on the watch and WLAN on the PerServ. Since the PerServ is carried concealed in the user’s pockets, the watch is easily within the maximum of range of the Bluetooth transmitter. The PerServ software then routes data from the WLAN connection over the Bluetooth connection to the watch, thus making WLAN available to the watch without the associated high power requirements.

4 User Interface Framework

The user interface framework is broken down into a number of different hardware and software systems. In this section, we describe the software framework we have designed to run many different kinds of applications.

4.1 The PerServ Framework

The novel contribution of our watch is not only the hardware, but the combination of applications and the easy development of new ones to support a widespread application domain. We envision many applications displayed on the watch operating via a mobile PerServ, such as an iPAQ for example. These applications must work seamlessly and efficiently.

We have developed an application framework that facilitates the construction and execution of a broad range of applications. For each application, two virtual displays are created, one for local display on the PerServ and one for the remote display on the watch. The framework running on the PerServ is a simplified operating system that is used to control the watch - a traditional operating system provides an abstraction to the hardware and the possibility of operating multiple applications simultaneously. In our framework, there are hardware abstractions for the display, the touch-screen, and the keyboard of the PerServ, and also an abstraction for the display on the watch. Simple calls are provided to access the display of the watch with drawing functions that can render graphic primitives and text. This is mandatory since an application should not have to "know" which pixels to light up when drawing a line across the display. The framework ensures that only one application can send information to the watch at one time, thus keeping the display content consistent. Figure 4 shows how the framework handles redraw attempts. Applications (a, b, c) send redraw requests to the framework, and depending on the currently active application, the screen is redrawn. External applications can be used in the framework through wrapper applications which provide a suitable abstraction layer.

We have implemented a version of the framework in Java under Microsoft PocketPC and in C under Familiar Linux 0.7.2; thus the framework is able to render applications using the drawing functions of Java Swing/AWT or X Windows. Both libraries are quite simple and provide the ability to transmit either AWT canvases or X pixmaps, allowing the use of standard drawing primitives.

One important aspect of the framework is to make installing and executing new programs as simple as possible. This is already simplified by separating the watch and the PerServ. Ideally, no installation routines are necessary; applications are copied onto the iPAQ and are available immediately. Un-installing should work as simple as deleting the application files from the PerServ. This is solved with a plug-in system in both implementations. Although it makes updating, installing and removing applications very simple, this method has a drawback: the PerServ is completely separated from the applications and their logic, and it does not know which application is responsible for which task. However, this can be solved by adding another auxiliary application that starts up the requested application on demand.

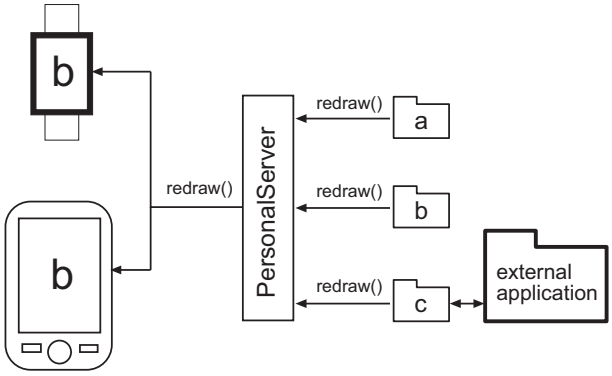

Figure 4. The PerServ framework.

4.2 Task Manager

As previously mentioned, a framework is provided for the applications to allow easy access to the watch's hardware. Each application renders its output to an off-screen canvas or pixmap. Due to the display size, only one application can access the display at one time, and the application to display is decided by the PerServ. The clock application is the exception to this rule and has a small fixed area on the display that is not accessible by the other applications. The Task Manager provides the functionality to switch applications at runtime. A well-known approach is to implement a list-based task manager like the one used in Microsoft Windows, KDE, and other window managers. We have implemented in the PerServ a similar concept to change applications on our watch. The user can view a list of the running applications and then bring one in the foreground by using the touch-screen on the iPAQ.

A more common method to change applications would be through the controls on the watch. To realise the task manager on the watch at least one button is required. By pressing this button the task manager is started and shows the applications. While holding the button the task manager cycles through the applications and on release the currently selected applications appears on top. Early cell phones used this method to write SMS messages. A proper speed to scroll is required, and must be customisable to some extent by the user. A second solution of using two or three buttons is preferable here:

Figure 5. The task manager application running on the current watch prototype.

one button to start the task manager and to confirm the selection and one or two buttons to cycle through the list. Figure 5 shows the watch prototype displaying a list of tasks currently running on the PerServ, but since there is currently no input capability on the watch it is not possible to select the task using the watch.

4.3 Smart Application Switching

Our watch system supports the switching of applications automatically depending on the priority. To help define the priorities, we have defined four categories of applications:

- The first category of applications is those that display for a very long time but do not have important information at one specific point in time. An example would be a video stream from a baby monitor. While the information provided over a long time is relevant, the information at one specific second is not especially important or different from the previous.
- The second category is an application that provides highly important information but only for a very short time. Email notifiers are in this class of applications - within five seconds after an email has arrived, this information is very valuable, but once read there is no information worth displaying until the reception of the next email.
- The third category of applications is those having information useful for a limited amount of time. A video stream sent from a door bell has important information over a time of about 30 seconds until the visitor opens the door or leaves.
- The last category of applications is when the value of the information changes over time. A countdown on a timer application for example is more important when it is close to the end than at the start.

Switching applications automatically is especially important when there are no input devices connected to the watch. It allows the watch to respond to the user's context without the need for the user to interact with the PerServ device.

5 Example Applications

In this section, some applications are described that were implemented in either the C or the Java version of our framework (or both). Each of these applications is an example of how to process different types of input data for a practical use. The *Timer* application is an example of the watch system connecting to common household appliances. The *MP3 Title Display* application shows how to connect the PerServ with an external application that plays music. The *RSS Feeds* application is an example demonstrating how applications can fetch live data from the Internet such as news feeds. In contrast, the *Alarm* application demonstrates the use of file-based input data containing appointments. Finally, the *Baby Monitor* application implements a still image and streaming video capability. A clock application is always running parallel to the others, showing the current time and date on the bottom of the display.

5.1 Direct Hardware Communication - The Timer

The PerServ is designed to be able to communicate directly with other hardware devices. We implemented a flexible timer application to potentially operate with many common household devices. This timer shows that future household devices only need slight modifications to be able to communicate and make use of the PerServ architecture. The scenario we used for inspiration was someone watching television, who decides to make some microwave popcorn in the kitchen. During a commercial break, the user leaves the television room to place a packet of popcorn in the kitchen microwave oven. The user sets the timer on the microwave oven and presses the start button. Once the oven starts to cook the popcorn, the oven communicates with the user's PerServ to start a countdown timer displayed on their watch. This way the user may return to view the television, and their watch will inform them when the popcorn is ready.

The Timer application is a proximity-based interaction to support countdown timer functions. The goal of this form of application and interface is to be automatic, invisible, and lightweight. The actions of the user dictate the start and duration of the timer. Once the timer has started, the users return their focus to a different activity. The countdown is displayed on their watch and the user does not have to stay within audible or visual distance of the household appliance being timed. The extension to existing household appliances to support such communication and functionality is fairly simple to implement. Cheap microcontrollers are able to send these simple messages over a Bluetooth network, and Bluetooth chipsets are fairly common nowadays. More and more household devices already have network connectivity and we expect that in the future most appliances will be connected to a local or global computer network.

The use of a countdown timer may be confusing when multiple devices have started timers; for example a coffee machine or a microwave oven could use the watch to show the remaining time simultaneously. To overcome this problem, the household device needs to send additional information to the timer application so that the PerServ can differentiate between the update messages. We have designed a simple message format to support the Timer application, and the format is shown in Figure 6. In our implementation, a message has seven or more bytes, where the first two bytes are used as an identifying sequence. Four bytes are used to define the time in seconds, thus allowing a maximum of over four billion seconds (or approximately 136 years). This enables a range of devices with diverse countdown times to use the same message type. In addition to those bytes, the packet specification allows an identification string of a maximum length of 255 characters to display the symbolic name of the source of the timer packet.

0	15	16	31
0xFF	0xFF	Len	Sec1
Sec2	Sec3	Sec4	…

Figure 6. Structure of a timer packet, containing header bytes, number of seconds remaining, and the identifier string of the device.

Figure 7. The timer application showing five different timers in operation.

The timer application is not necessarily locked to providing timing information for a physical device. Since the message type is quite abstract, a software application can use the Timer to display remaining time and additional information. For example, when downloading a large file off of a web, the remaining time and amount of data downloaded is displayed to the user on their watch. This frees the user from sitting in front of the computer to observe the progress of the download. The timer application is built to show multiple countdowns at the same time, sorted by finishing time, see Figure 7. The development of this application was quite simple because the plug-in API we have developed is designed to abstract away most of the functionality like the GUI and the network interface. We emulated the different household devices with a desktop PC workstation with a standard USB Bluetooth adapter.

5.2 Communicating to other Applications - MP3 Title Display

To make a watch-based display system useful, external applications should be able to easily communicate with PerServ applications. One example is the MP3 Title Display application we have developed that consists of both a PerServ application as well as a plug-in for the X MultiMedia System (XMMS). XMMS is a popular MP3 Player for Linux, which provides a rich API for extensions. The plug-in retrieves information about the current song and passes it on to the PerServ application. An example of song information displayed on the watch is shown in Figure 8.

The communication between the MP3 Title Display application and the XMMS plug-in is quite simple to implement. Information about the music playing on XMMS is polled once per second; this includes such information as the artist, the title, the length of the song, and the position in the play list. This information is sent as a set of UDP packets, thus allowing the plug-in to be always active even though no PerServ application is connected. Information about the song is contained in seven individual UDP packets which contain a name and value pair, as shown in the following example:

```
Packet 1: ARTIST: Ben Harper
Packet 2: TITLE: Faded
Packet 3: PLAYLIST: 409
Packet 4: POSITION: 339
Packet 5: TOTALTIME: 289
Packet 6: TIME: 118
Packet 7: PLAYING: 1
```

This example shows that the current song is “Faded” from the artist “Ben Harper” at position 339 of 409 possible songs in the play list. The song has been playing for 1:58 minutes (118 seconds) already, and the length of the song is of 4:49 minutes (289 seconds). The last line defines whether XMMS is playing (1) or stopped/paused (0) at the current moment.

The ability to communicate with non-PerServ applications extends the functionality of the watch to a larger application domain. A similar system to the MP3 Title Display could be used to connect the user’s mail client with an application on the PerServ. It would be possible to get information about the user’s mailbox without the need to implement protocols such as POP3 or IMAP. Modern e-mail clients, such as Mozilla and Outlook provide a plug-in system that allows for such a feature. The communication between the PerServ applications and the external application could happen either via networking or via a common shared library. For simplicity and to ease portability, the MP3 Title Display communicates with the XMMS plug-in via a network. This allows the MP3 Title Display to communicate with other players with plug-in interfaces such as WinAmp.

5.3 Live Data Streams - RSS Feeds

Live data streams are vitally important to many business people today. For example, data streams can consist of the latest stock quotes or headline news. A widely used method is Really Simple Syndication (RSS) (Harvard Law 2004), a dialect of XML that conforms to the XML 1 standard. A large number of news sites support RSS, such as BBC, The Wall Street Journal, and the New York Times. Smartmoney.com provides a free RSS service for financial information.

Figure 8. The MP3 title display application showing the current song in progress and song information.

A big advantage in RSS's XML structure is that it supports the use of pack-and-go XML parsers without the need to implement an individual parser. This facilitates the use of more than one stream at the same time. Aggregation of RSS feeds is important as free feeds often have low update rates. Combining different feeds with different update rates allows for an overall better update frequency.

The output of an RSS news feed can be either graphical or textual, depending on the user's context and the information itself. If it is necessary to monitor stock prices over a longer time interval, a graphical interface such as line diagrams may be preferred. However, if there are too many lines on the diagram, it may become difficult to read. Complicated graphs on a display as small as the one on the watch may prove difficult to provide a proper legend. In this case it is better to display the information in text format.

When the RSS feed only provides textual data, the decision is then to either represent the data as a ticker or as fixed text blending from time to time. With fixed text, the problem on small displays is that longer headlines are difficult to render. It is possible to display about 15 to 20 words on the display, depending on the length of the words. Text has to be optimised for the small display size, but in general most news feeds are not optimised by default. Especially long words would have to be separated in two, but to follow grammatical rules, computationally intense algorithms are required. This makes the fixed text less attractive for longer messages.

It is well known that animations on a user interface may be distracting to the user (Baecker, Small et al. 1991; Thomas and Calder 2001). However, news feeds do not suffer from this problem as they require the full attention of the user, and the animations on the watch will typically not be largely visible to the user at other times. Setting a proper scrolling speed for the ticker is difficult to determine for all users. We found it challenging to find a relation between too fast to read and too slow to wait for the next news line. The best solution to this is to let the user define the scrolling speed as every user will find a different speed optimal. Our application has a configuration file for this to be persistent over multiple sessions.

Another way of presenting the data would be using audio output. This would require speakers or headphone plugs on the watch, which are both non-existent on the current prototype. Also, the social weight of an audio enabled application on the watch is different to an application using purely visual output.

5.4 Personal Information Management - Alarm Clock

In contrast to the applications previously mentioned, the Alarm Clock application is an example of how a permanent connection to external devices and/or networks is not required (using only a mobile PerServ). This is necessary for mobile situations, such as when on public transport where there may be no Internet connection. Instead, the Alarm Clock uses a data file to retrieve the user's schedule. This data file is an exported file from the PIM application KOrganizer. The Alarm Clock uses the priority redraw to display upcoming events when necessary. In addition, it uses the PerServ's APIs to fetch information from the global application configuration file. Synchronising the Alarm Clock with the user's organiser is simple. KOrganizer exports ".ics" files, which can then be copied onto the PerServ directly without modification. Our implementation had KOrganizer operating on a standard PC workstation, connecting to an iPAQ operating as the PerServ. An example of the Alarm application running on the watch is shown in Figure 9, with three appointments shown.

When the Alarm application is active, the application shows the next five entries within the next two hours. Thirty minutes before a scheduled event is due, the event will be displayed with high priority to notify the watch user.

5.5 Video Streaming - Baby Monitor

The third generation of mobile phones has shown the appeal of highly mobile video. We propose an alternative to using the phone's display, which is to display the video stream on the display of the watch - the user can use their handset as an audio input device and the watch as a video output device. Video streaming is useful in other domains as well, such as doorbell cameras or baby monitors to survey a child while it is sleeping in its bed. This section describes a generic video transfer application from the PerServ to the watch.

There are a number of challenges to providing video streaming to a watch device. The first is that since there is no camera mounted onto the watch, the remote user will not receive any live video from the local user. This might not be important in most situations, though it is in business conversations where this might be considered impolite. The second problem occurring with streaming video is the required high data transfer rate, which is typically greater than what hardware like the MSP430 and Bluetooth can support. Only with complex compression codecs and low frame rates is it possible to transfer video without introducing large delays or preventing real-time operation. Since the watch uses only a very simple CPU, it is impossible to decode typical video compression algorithms such as MPEG-4 without dedicated hardware support. Additionally, in the case of

Figure 9. List of current appointments on the watch.

Figure 10. Example of a picture shown on the display.

video telephony, the video would have to be transferred by the PerServ device twice, from the phone to the PerServ and then from the PerServ to the watch. If enough Bluetooth bandwidth was available, it would be possible for our PerServ to support a separate Bluetooth enhanced video camera. This detachable video camera could be placed in other locations that might be more convenient than the user's wrist.

In our watch system, we can only support applications where the required frame rates are very low, such as in the case of a baby monitor. Although the camera might provide 25 fps or more, it is possible to drop frames down to a frame rate of 1 fps or even less without losing much of the information. For observing a sleeping baby, one frame per second is satisfactory, and the PerServ can even monitor this and only send new frames if any movement occurs.

Dropping frames to reduce the frame rate has to be done as early as possible to maintain a consistent reduced frame rate. If one frame needs one second transfer time, a 25 fps video stream will take 25 seconds to transmit one second of video over the network. We desire real-time operation for our application, as mentioned previously. In our implementation, we have a Linux PC which captures video frames, transmits them to the PerServ, which then transmits them to the watch. The implementation on the Linux PC uses the Java MediaFramework and monitors the transfer rate and drops frames dynamically depending on how fast the PerServ and watch are able to handle them. The PC software reduces the video frames to fit the size of the LCD and packs it in a format that is ready for display directly to the watch's LCD, so the PerServ simply acts as a router.

6 Conclusion

In this paper we have presented lightweight user interfaces for watch based displays, which are simple to develop and use. Our design philosophy was that our new watch device should not be more difficult to use than the user's current watch. We have taken the approach that there will be no one killer application in the watch domain, but an accumulation of functionally and contextually appropriate applications. The ability to support a vast array of applications requires an open API and an open set of protocols. The open architecture we have developed allows easy development of applications from a wide community of software developers. It is only by allowing multi-vendor solutions will the quantity and quality of applications become adequate enough for the consumer to make use of this technology. We see HTTP as a model of how a simple protocol facilitates the ability to provide functionality across a number of application domains and hardware platforms. Where simple protocols are appropriate, they assist in the adoption of new technologies.

The watch is a display-only device, and as such a very simple protocol is able to support all the functionality required. This simplicity allows for the design and construction of a low cost physical platform for the watch that may connect to a number of different PerServs. As an example, we envision the user's mobile phone providing personal server support in the future.

In answering our original question "Who is going to tell Bruce his coffee is going to be twenty minutes late this morning?", we have developed the following: a custom wrist display device appropriate for a watch, a personal server software system, a lightweight framework to develop and execute watch-based applications, and a set of five example applications. A prototype for a new watch which contains only a display, limited processing capability, and no input devices was constructed. We employed Bluetooth to wirelessly communicate with an external device (a Personal Server) that performs the processing.

Our lightweight framework allows developers to build their watch applications using traditional development methods, and with interfacing to the different applications kept very simple. For example, we use information push technologies and/or proximity of initiate an application. In this paper we presented the following five example applications:

1. The *Timer* application communicates with common household appliances to initiate a countdown timer displayed on the watch.
2. The *MP3 Title Display* application communicates with an external application that plays music and the watch displays information about the current song playing.
3. The *RSS Feeds* application displays live data feeds from the Internet on the watch.
4. The *Alarm Clock* application communicates with a PIM and displays on the watch the user's current appointments.
5. The *Baby Monitor* application implements a streaming image and video capability.

The two major areas of investigation for the watch platform in the future are a more powerful processor with similar power consumption specifications and a replacement for Bluetooth wireless networking. The Bluetooth protocol is quite burdensome in terms of software support, and we feel that much lighter-weight wireless solutions such as ZigBee (Craig 2004) would be more appropriate.

7 Acknowledgements

We would like to thank HP Labs for their support on this project. This project was made possible by the continuing support of the School of Computer and Information Science at the University of South Australia. Aaron Toney provided a great sounding board and wealth of information about programming with low powered devices. We would also like to extend a big thank you to all the people in the Wearable Computer Lab for their ongoing support.

8 References

Baecker, R., I. Small, et al. (1991). Bringing Icons to Life. ACM CHI '91 Conference on Human Factors in Computing Systems, ACM.

Bluetooth SIG, I. (2004). Bluetooth Core Specification https://www.bluetooth.org/spec/.

Craig, W. C. (2004). Zigbee: "Wireless Control That Simply Works", ZigBee Alliance http://www.zigbee.org/resources/documents/2004_ZigBee_CDC-P810_Craig_Paper.pdf.

Field Technology CxMP Ltd. (2004). Smart Watches. Room 2714-2716, 27/F, Hong Kong Plaza, 186-191 Connaught Road West, Hong Kong http://www.smart-watches.com/.

Fossil (2004). Wrist Net Reference Guide http://www.fossil.com/text/content/tech/downloads/wristnetreferencequide.pdf.

Geiger, C., B. Kleinnjohann, et al. (2002). Mobile AR4ALL. The First IEEE International Workshop Augmented Reality Toolkit.

Golem (2004). WLAN-Karte mit 100 mW Sendeleistung von Allnet http://www.golem.de/0304/25188.html.

Hans, M. C. and M. T. Smith (2003). A Wearable Networked MP3 Player and "Turntable" for Collaborative Scratching. Seventh IEEE International Symposium on Wearable Computers, White Plains, New York, USA, IEEE.

Harvard Law (2004). Really Simple Syndication (RSS) Specification, Berkman Center, http://blogs.law.harvard.edu/tech/rss.

Kamijoh, N., T. Inoue, et al. (2001). Energy Trade-offs in the IBM Wristwatch Computer. 5th International Symposium on Wearable, IEEE.

Martin, T. L. (2002). Time and Time Again: Parallels in the Development of the Watch and the Wearable Computer. 6th International Symposium on Wearable Computers, IEEE.

Matsucom, I. (2004). onHand/Ruputer. 1642 South Parker Road, Suite 212, Denver, Colorado, 80231 U.S.A.

Microsoft-Research (2004). Smart Personal Objects. One Microsoft Way, Redmond, WA 98052, USA http://research.microsoft.com/spo/.

Mitsumi (2004). BluetoothTM Module WML-C09 2-11-2, Tsurumaki, Tama-shi, Tokyo 206-8567, Japan, http://www.mitsumi.com/.

Narayanaswami, C., N. Kamijoh, et al. (2002). "IBM's linux watch: The challenge of miniaturization." IEEE Computer: 33-41.

Narayanaswami, C., M. T. Raghunath, et al. (2001). What Would You Do with a Hundred MIPS on Your Wrist?,, IBM Research Division, Thomas J. Watson Research Center

Narayanswami, C. and M. T. Raghunath (2000). Application Design for a Smart Watch with a High Resolution Display. International Symposium on Wearable Computing, Atlanta, Georgia.

Scheifler, R. W. and J. Gettys (1986). "X WINDOW SYSTEM." ACM Transactions on Graphics **5**(2): 79-109.

Sun Microsystems (2004). Sun Ray 1g Ultra-Thin Client http://wwws.sun.com/sunray/sunray1/.

Texas Instruments (2004). MSP430x13x, MSP430x14x, MSP430x14x1 MIXED SIGNAL MICROCONTROLLER. Post Office Box 655303 Dallas, Texas 75265, Texas Instruments

Thomas, B. H. and P. R. Calder (2001). "Applying cartoon animation techniques to graphical user interfaces." ACM Transactions of Computer Human Interactions **8**(3): 198-222.

Timex Corporation (2004). PO Box 310, Middlebury, CT 06762, http://www.timex.com/.

Toney, A., B. Mulley, et al. (2003). "Social Weight: Designing to minimise the social consequences arising from technology use by the mobile professional." Personal and Ubiquitous Computing **7**(5): 309-320.

Uemukai, T., T. Hara, et al. (2002). A Remote Display Environment: An Integration of Mobile and Ubiquitous Computing Environments. Proceedings of IEEE Wireless Communications and Networking Conference, Orlando, Florida, USA.