A Comparative Study of Two Bluetooth APIs for Implementation in an Automated Wireless Identification System (AWIS)

Submitted in partial fulfillment of the requirement of the degree Bachelor of Science (Honours) of Rhodes University

Samantha Knights

8 November 2004
Abstract

In order to minimize human interference, an Automated Wireless Identification System (AWIS) is proposed to monitor individual animals’ interactions in the wild. This is based on a Bluetooth enabled collar (device) fitted to each animal. When one device comes into the 10 meter range of another device, a peer-to-peer interaction or “Encounter” is logged.

Application developers can utilise the underlying capabilities of the Bluetooth Protocol Stack via an API – Application Programming Interface. The Bluetooth Specification makes no mention of how a simple exposed API is defined, leading many manufacturers to add their own unique APIs. This project investigated Bluetooth Microsoft APIs (Winsock and Virtual Serial Ports) and a proprietary API from Widcomm, with a view to evaluate their support for monitoring mobile device Encounters in real time for the AWIS.

In order to compare Bluetooth APIs, it was necessary to use the tools available directly, to observe how the APIs enable Bluetooth development. Winsock and Virtual Serial Ports are the recommended Windows APIs and Widcomm is one of the many available proprietary APIs. Experimental applications were written to test the most suitable API to use in an AWIS application. A criterion is used for testing was the functionality required by the AWIS device discovery, service discovery and data-transfer, as well as flexibility and enabling rapid prototyping.

The Widcomm API was found to be the most suitable API for the AWIS implementation as it allows for a great deal of low-level functionality while hiding any unnecessary detail from the developer. Virtual Serial Ports provided limited functionality and flexibility, and Winsock provided the least functionality.
Acknowledgements

I would like to thank my supervisors Dr Greg Foster and Professor Peter Clayton for their guidance and unlimited patience. Thank you to Bradley Clayton and Adam King for their help and sharing of knowledge; and to my family and friends for their support and proof-reading. My pod-mates and Brad Herholdt for their unending support. I am also very grateful for the Andrew Mellon Foundation for funding my Honours degree this year.

The AWIS project was sponsored by an Embedded Systems Research grant from Microsoft Research. It was performed under the umbrella of the Center of Excellence (CoE) in Distributed Multimedia at Rhodes University, which is sponsored by Telkom SA Ltd.; Business Connexion (Pty) Ltd.; Converse SA, Inc; Verso Technologies, Inc and THRIP (The Technology and Human Resources for Industry Programme).
# Contents

1. Introduction.............................................................................................................1  
   1.1 CURRENT ANIMAL TRACKING METHODS.................................................1  
   1.2 AWIS.................................................................................................3  
   1.3 CHOICE OF BLUETOOTH OVER OTHER WIRELESS TECHNOLOGIES ..................................................3  
      1.3.1 IEEE 802.11b (Wi-Fi)........................................................................4  
      1.3.2 ZigBee....................................................................................4  
      1.3.3 Bluetooth .............................................................................5  
   1.4 OVERVIEW OF BLUETOOTH TECHNOLOGY..............................................6  
      1.4.1 Goals and Vision ........................................................................6  
      1.4.2 Bluetooth Specification: Architecture .........................................7  
      1.4.3 Bluetooth Stack .......................................................................8  
      1.4.4 Bluetooth Specification: Profiles ..............................................9  
   1.5 BLUETOOTH APPLICATION DEVELOPMENT ..............................................9  
      1.5.1 Bluetooth Application Programming Interfaces (APIs)..................9  
      1.5.2 Windows Application Programming Interfaces ................................11  
      1.5.3 Widcomm Bluetooth API.........................................................12  
   1.6 PROJECT AIMS..................................................................................12  
2. Methodology .....................................................................................................14  
   2.1 MATERIALS ..................................................................................14  
      2.1.1 Hardware ............................................................................14  
      2.1.2 Software ............................................................................15  
   2.2 APPROACH ..................................................................................18  
      2.2.1 Iterative development of Test Application............................19  
      2.2.2 Comparison Criteria for APIs ............................................21  
      2.2.3 Steps Toward the Automated Wireless Identification System ....22  
      2.2.4 Field Testing the Final AWIS Application ...........................23  
   2.3 SUMMARY ....................................................................................24  
3. Results .............................................................................................................26  
   3.1 GENERAL DEVELOPMENT EXPERIENCE .............................................26  
      3.1.1 Hardware Related Problems ............................................26  
      3.1.2 Visual Studio 2003 Development Environment ......................27  
   3.2 WINDOWS API’s ............................................................................27  
      3.2.1 Virtual Serial Ports ................................................................27  
      3.2.2 Winsock .............................................................................29  
   3.3 WIDCOMM API .............................................................................30  
   3.4 FUNCTIONAL DESCRIPTION OF THE FINAL AWIS APPLICATION ....32  
   3.5 FIELD TESTING THE AWIS FINAL APPLICATION .............................35  
      3.5.1 AWIS Device Discovery .......................................................35  
      3.5.2 Time Taken for Encounter Log ...........................................36  
      3.5.3 Encounter Log Upload and Interference .................................36  
   3.6 SUMMARY ....................................................................................37
4. Discussion ............................................................................................................ 38
  4.1 GENERAL IMPLEMENTATION ISSUES AND LIMITATIONS .......... 38
  4.2 COMPARISON OF WINDOWS APIs and WIDCOMM API .................... 39
    4.2.1 Device Inquiry ................................................................................. 40
    4.2.2 Service Discovery ........................................................................... 40
    4.2.3 Data Transfer ................................................................................ 41
    4.2.4 Functionality and Flexibility .......................................................... 41
    4.2.5 Compatibility ................................................................................ 45
    4.2.6 Stack Interaction and Rapid Prototyping ....................................... 45
  4.3 FIELD TESTING FINAL AWIS APPLICATION ................................ 46
    4.3.1 AWIS Device Discovery and Time Taken for Encounter Log ....... 46
    4.3.2 Encounter Log Upload and Interference ....................................... 47
  4.4 SUMMARY ............................................................................................... 48
5. Conclusion and Future Work ............................................................................ 49
  5.1 FINDINGS AND RECOMMENDATIONS ............................................ 49
  5.2 FUTURE WORK .................................................................................... 50
  5.3 AWIS APPLICATION AND EXTENSIONS .......................................... 51
  5.4 SUMMARY ............................................................................................... 52
References ............................................................................................................. 53
List of Figures

Figure 1.1: Outline of the Bluetooth Stack .............................................. 8
Figure 2.2: Windows CE Bluetooth Stack .............................................. 17
Figure 2.2: Overview of Bluetooth for Windows CE using Widcomm ....... 18
Figure 3.1: Relationship between AWIS Application Classes and Widcomm SDK Classes ................................................................. 35
List of Tables

Table 1.1: Comparison of Wireless Technologies.................................4
Table 3.1: Comparison of Bluetooth APIs on an iPAQ h4150..................29
Table 3.2: Extract of Closed Encounters Log..................................36
Table 4.1: Listing of Methods available for the Virtual Serial Ports API......43
Table 4.2: Listing of Classes available for the Widcomm API...............44
Chapter 1

Introduction

Individual animal identification and social interaction monitoring is necessary to understand animal behaviour patterns. There are a number of automated animal tracking methods (World Wildlife Fund, 2004; Amstrup et al., 2004), which all require human intervention. Even transponder tags require some human interaction of observations made by researchers in the field.

1.1 CURRENT ANIMAL TRACKING METHODS

The World Wildlife Fund (WWF) currently uses radio collars to track polar bears movements in their natural habitats (WWF’s Polar Bear Tracker, 2004). The collars are tracked via satellite and used to ascertain the range of travel of individual animals, as well as their adaptations to changing climates. According to Stephenson (2003) elephants are also collared and tracked in national parks in central Africa. Data such as home ranges, movement patterns and habitat usage are collected. This is used for following “elephant migration corridors” and the development of “transboundary anti-poaching” strategies.
According to Amstrup et al. (2004) fitting a device to individual polar bears for such a large area is expensive and “unacceptable as routine management practice”. Instead, a polar bear’s own body heat could be detected with FLIR (Forward-Looking Infrared) Imagery technology. Inhabited dens should thus produce a pronounced “heat signature”. This was successful but only under the correct conditions; bad weather, sunlight (and other major heat sources) and depths too great to be detected led to unfavourable results.

A study conducted by Alibhai et al. (2001) suggests that radio collars are not ideal for black rhinos. Routine radio-collaring was reported to have a number of drawbacks: they were expensive to replace, often resulted in wounding and were difficult to maintain as expertise and time were required when transmitters become faulty. There was also possibility of compromise in female fertility. The authors suggest that radio-collaring is not a viable long-term strategy for protection and monitoring.

Steiner et al. (2000) attempted to use GPS technology to track homing pigeons and small animals (dogs). By mounting a GPS logger onto the back of the animal, the accuracy of this was only limited to 10m. Tracking the dogs was less successful than pigeons, particularly when they entered denser mediums such as forests. The main issues were power and size constraints. It is suggested that these devices could be used to test hypothesis of pigeon homing, spatial behaviour and orientation.

A commercial company in Sweden, TVP Positioning AB (2004), manufactures various GPS collar products. Their GPS collars, weighing between 350g to 1000g, allow for remote data download and reprogramming via GSM, satellite or radio. The features of these collars suggest what would be required for any animal collar: a design that minimises impact on the animal (comfort, size), low battery indicator, mortality indicator, memory larger than battery life-time – to avoid data loss also non-volatile memory, temperature tested and tracking schedules.

“BlueTrak” is a patented automated system using Bluetooth to gather interaction data for both animals in the wild as well as other applications such as monitoring physical
CHAPTER 1. INTRODUCTION

conditions of the elderly in nursing homes. (Song & Wu, 2002). BlueTrak is a commercial system and limited technical detail is provided by the authors. Although the design specification is similar what is being attempted in the AWIS project, the implementation differs: the development tools used in this project was not necessarily available at the time of BlueTrak’s development.

1.2 AWIS

An Automated Wireless Identification System (AWIS) has been proposed as part of this study that will provide a novel method by which each animal will be fitted with a Bluetooth enabled collar. Each collar will collect information about the other collars when in range. This peer-to-peer interaction is defined as an “Encounter”. When a tagged animal passes a “Beacon” or base station at fixed sites, for example watering holes, historic information such as a log of previous Encounters will be automatically transferred.

There are thus three main elements to an AWIS application:

1. Enable an inquiry to find all other Bluetooth devices currently in range.

2. Log when another device was found – i.e. an Encounter. Keep track of when this device goes out of range (close Encounter). Build up a list of these Encounters.

3. Send this list of Encounters to a Beacon when one is in range.

1.3 CHOICE OF BLUETOOTH OVER OTHER WIRELESS TECHNOLOGIES

Dursch et al. (2004) suggest 4 other main wireless technologies available: Induction Wireless, Infrared Wireless, ZigBee, IEEE 802.11b (Wi-Fi). Dividney (2003) considers Bluetooth, Wi-Fi and IrDA been the most popular standards in 2003. A comparison of potential wireless technologies for the AWIS is provided in Table 1.1. Induction Wireless would not be suitable for the AWIS as it has a maximum radius of 3m and slower speeds
than Bluetooth (Dursch et al., 2004). Infrared Wireless is limited to two participants, point-to-point connection and requires line-of-sight to function (Barnes, 2003). Infrared is therefore a non-viable option for animal tagging as there could be a variety of obstacles between the animals themselves, for example trees, boulders.

Table 1.1: Comparison of Wireless Technologies. Modified from Legg 2004 and Diviney 2003

<table>
<thead>
<tr>
<th></th>
<th>Bluetooth</th>
<th>Wi-Fi</th>
<th>ZigBee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Medium</td>
<td>2.4 GHz RF</td>
<td>2.4 GHz RF</td>
<td>915 MHz, 868 MHz, 2.4 GHz RF</td>
</tr>
<tr>
<td>Physical Range</td>
<td>10 m</td>
<td>100+ m</td>
<td>100 m</td>
</tr>
<tr>
<td>Data Rate</td>
<td>1 Mbits/s</td>
<td>10 Mbits/s</td>
<td>250 Kbits/s</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>Low</td>
<td>High</td>
<td>Minimal</td>
</tr>
</tbody>
</table>

1.3.1 IEEE 802.11b (Wi-Fi)

Wi-Fi is a standard specification for wireless LAN and works on the same frequency band as Bluetooth (Dursch et al., 2004). Wi-Fi power consumption is considerably larger: currently a Bluetooth device’s minimum current requirements are one tenth of Wireless LAN (Pico Communications, 2001). Bluetooth’s range can be extended with the concession of greater power consumption. This is possible because Bluetooth devices are divided into three classes based on their range; Class 1 has a maximum range of 100m, Class 2 range of 15m and Class 3 the smallest – 10m. Nevertheless Wi-Fi’s “always on connection model” limits its use in PDAs, phones, and other lightweight mobile devices (Dividney, 2003). Bluetooth devices are also much smaller and cheaper than Wi-Fi units. (Barnes, 2003).

1.3.2 ZigBee
CHAPTER 1. INTRODUCTION

ZigBee has similar characteristics to Bluetooth, its main design goals are low power, low cost and robustness. (Legg, 2004). Power is saved by going into “sleep” mode with the capacity to switch into active mode in less than 15msecs. Bluetooth has a more significant switching mode delay of around 3 seconds. Its range can be extended to 134m operating on a low data transfer rate. ZigBee operates at same frequency band as Bluetooth using frequency-hopping spread spectrum (FHSS) but fewer hops (25 every 4MHz), and slower data transfer at 250Kbps. ZigBee Chips are being manufactured by major semiconductor companies and Atmel have recently released a ZigBee-specific radio and microcontroller (Page, 2004).

1.3.3 Bluetooth

Bluetooth supports wireless connections limited to 10 m apart with low-power consumption for battery operated mobile devices. It provides voice, data and audio connections (Bluetooth SIG, 2004). Bluetooth operates in unlicensed 2.4 GHz radio spectrum allowing for global use and compatibility. To reduce interference the protocol implements full-duplex frequency hoping at 1600 hops per sec for 79 frequencies spaced at 1Mhz intervals. The data rate quoted as 1Mbits/s, allowing for point-to-point connectivity for up to 7 simultaneous connections. The key elements to consider when comparing Bluetooth to other wireless technologies are power consumption and intended use. (Bluetooth SIG, 2004).
CHAPTER 1. INTRODUCTION

Bluetooth had a number of characteristics which provided motivation for its use in this application:

**Limited range** - Bluetooth range is important as animals with Bluetooth collars will have the ability to come in and out of contact with each other.

**Low power usage** - It would be impractical and intrusive for humans to have to regularly change and charge devices attached to the animals. Bluetooth was designed to have low-power consumption for battery operated mobile devices (Haartsen et al., 1998).

**Compact size** - The small size of the Bluetooth unit will help to reduce the interference and discomfort of animals under observation. The radio collars used on animals can cause injury and are difficult to maintain (Alibhai et al., 2001).

1.4 OVERVIEW OF BLUETOOTH TECHNOLOGY

1.4.1 Goals and Vision

Bluetooth implements a wireless connectivity, networking and application framework defined by the Bluetooth Specification (Bluetooth SIG, 2004). It was specified from the outset that the Bluetooth module should be small enough for integration into portable devices and should not “significantly compromise battery lifetime of the device” (Haartsen et al., 1998). The specifications also recognised the dynamic nature of Bluetooth networks for detection and connection establishment (Haartsen et al., 1998).

Bluetooth specification is divided into two main sections: how the technology works (architecture) and how the technology is used (profiles) (Jones et al., 2002).
1.4.2 Bluetooth Specification: Architecture

A Bluetooth unit contains RF radio, baseband controller and microprocessor on single CMOS integrated circuit (Jones et al., 2002). The single chip design was required to meet the low cost, power and size requirements of the Bluetooth specification. Transmission occurs on the unlicensed Industrial, Scientific and Medical (ISM) band using Frequency-Hopping Spread Spectrum (FHSS) with retransmission always on different channel (Jones et al., 2002). A control processor or host facilitates upper layers of protocol and application software. Both application and stack must fit on the microcontroller, an ARM is often used and could be merely a 64kB processor (Bhagwat, 2001). Data rate of devices is managed by hardware and cannot be directly specified by application (Senese, 2001).

Bluetooth supports point-to-point and point-to-multipoint connections (Haartsen et al., 1998). Enabled devices have automatic communication – no specification is needed. When units come into range, a connection will be established and detection if there is any information to be passed. Pairing is the initial communication process when creating a new connection with an unknown device (Dursch et al., 2004). Once connections have been established, virtual channels are defined using pseudo-random hop sequences (Bluetooth SIG, 2004).

Piconet and Scatternets: A piconet consists of two or more Bluetooth devices sharing the same channel. For each piconet there must be one master device and one to seven slaves. Inactive slaves will exist in stand-by mode (this is to save power – Haartsen et al., 1998). Masters initiate Bluetooth communication links but a slave may request to become a master. Multiple overlapping piconets are called scatternets (Jones et al., 2002). Too many piconets result in more collisions, and can lead to falling data rates. Algorithms for forming scatternets and symmetric procedures for establishing connections are both popular topics of Bluetooth research (Bhagwat, 2001).
1.4.3 Bluetooth Stack

Hopkins 2003 describes the Bluetooth stack as a “controlling agent” that implements Bluetooth protocol and allows the user to control a device from a software point of view. This stack comprises of layers (sub-protocols) and profiles, allowing the user to communicate with other devices and control their own device (Figure 1.1). Application Developers can utilize underlying capabilities of Bluetooth Stack via an API – Application Programming Interface (Gratton, 2003).

![Figure 1.1: Outline of the Bluetooth Stack (modified from Boling, 2002)](image)

The Host Controller Interface (HCI) interfaces between the host and controller. LMP (Link Manager Protocol) is the protocol that handles link establishment between Bluetooth devices and BB (Baseband) enables the physical radio frequency (RF) link between Bluetooth units. (Hopkins, 2003; MSDN, 2004). The Logical Link Control and Adaptation Layer (L2CAP) handles data transmission from higher layers and SDP (Service Discovery Protocol) discovers Bluetooth devices and services in the surrounding
area. RFCOMM (Serial Cable Emulation Protocol) is the layer providing for the creation of virtual serial ports and stream data.

1.4.4 Bluetooth Specification: Profiles

Bluetooth Profiles define which features of the Bluetooth Stack are required and how they will be used (Gratton, 2003). These form basis for Bluetooth interoperability and are applications that allow short-range communication (Senese, 2001). A profile can be visualised as “a vertical slice through the protocol stack.” and may depend on other profiles (Jones et al., 2002). Some examples of profiles are: Generic Access, service discovery, serial port, headset, generic object exchange, and object push profile (Bhagwat, 2001).

1.5 BLUETOOTH APPLICATION DEVELOPMENT

Before developers can create end-user applications, the following system requirements must be integrated (Senese, 2001):

1. The Protocol stack and operating system,
2. A transport driver that interfaces the stack to the radio hardware,
3. Support software developed that is responsible for establishing connections to other Bluetooth devices (Application Programming Interfaces).

The Bluetooth Stack can now be used to implement the application, system control and bidirectional data flow (Hopkins, 2003).

1.5.1 Bluetooth Application Programming Interfaces (APIs)

According to (Hall, 2003) there is no definition in the Bluetooth Specification suggesting how application on a higher level should “talk” to the underlying stack. The Bluetooth Specification makes no mention of how a simple exposed API is defined leading many
CHAPTER 1. INTRODUCTION

manufacturers to add their own unique APIs. The Java API is one of the few standard industry APIs. The authors claim that for any C/C++ based Bluetooth SDK, the developer is “at the mercy” of the vendors (Hopkins, 2003).

There are an abundance of vendors who provide Bluetooth Stacks and APIs in the form of Standard Development Kits (SDKs).

Among the more widely used:
• Widcomm: provides one of the few APIs that allows direct access to several layers of the Bluetooth Stack.
• Socket Communications: allows developers to create Windows CE applications.
• Sourceforge (2004) also provides a long list of Open Source Bluetooth APIs to use on a variety of platforms in a number of different environments and programming languages. Some perform more specific tasks than others – BTNode System Software for example is an autonomous wireless communication and computing platform based on a Bluetooth radio and a microcontroller used for research in mobile and ad-hoc networking.
• Nokia, Motorola, Phillips, Intel, IBM – all major manufacturers provide their own unique APIs.
Listing provided by Palo Wireless (2004).

Cole (2004) suggests that what is needed is a common applications programming interface that “is industry wide, crossing boundaries of OS, application and most importantly, CPU architecture”. Despite the reduced number of competing groups backing specific architectures; OS and development tools, developers are still in a difficult position. Each different product requires “rethinking all the way down to the programming level”, requiring the developer to consider which platform is being used.

Evers (2004) reports that Koninklijke Philips Electronics and Samsung Electronics are launching a software standard that all manufacturers are invited to employ. They have noted there is no standard API for developers to write to. The Universal Home Application Programming Interface (UH-API) has been created to attempt to prevent
developers needing to consider “the unique hardware requirements of every product being developed.”

The Embedded Linux Consortium (2004) also attempting to create a standard Embedded-Linux API. They envision a platform standard for API's that will allow interoperability and successfully challenge “proprietary or home-made solutions.”

1.5.2 Windows Application Programming Interfaces

According to MSDN (2004) documentation on Bluetooth Application development:

In Windows CE the primary way an application can use Bluetooth is through the Winsock Interface, which exposes RFCOMM protocol. Virtual COM ports are also exposed, but this interface is designed to be enabled under existing OS services such as UNIMODEM or terminal emulators.

There are several design considerations when programming with Windows CE. Asynchronous functions are not supported but a socket can be placed into non-blocking mode. Windows CE does support multithread programming. Windows CE does not provide a method to expose a raw socket – and thus cannot deal directly with IP layer of TCP/IP protocol, making pinging impossible. To send an echo request, ICMP Internet Control Message Protocol must be used. (Boling, 2002).

1.5.2.1 Winsock

Winsock defines a network programming interface for Windows, based on the “socket paradigm” popularized in USB Unix. (Quinn, 1998). Winsock 2.0 is the most current version providing more functionality than the previous version, Winsock 1.1. Sometimes Winsock 1.1 is preferred because of its smaller size Windows CE utilizes Winsock 2. The API shields developers from the underlying layers of the Bluetooth Stack.
There are also two types of Sockets: Stream (Connection Orientated Connection) and Datagram. A Stream Socket can be described as a “data pipe”; once initial connections have been set up, data can be sent back and forth without the need for further addressing. Datagram Sockets are compared to “mail slots” where separate packets of data are sent to specific addresses (Boling, 2002).

1.5.2.2 Virtual Serial Ports

Winsock is recommended rather than Virtual Serial Ports for Bluetooth development (Boling, 2002). This is because despite their familiarity to programmers, serial ports have issues with driver names and their implementation is often unique. They are useful when legacy support is needed. According to MSDN documentation the COM port emulator is the “top most layer” of the Bluetooth Protocol Stack. It provides access to the RFCOMM layer based on a virtual COM Port (MSDN, 2004).

1.5.3 Widcomm Bluetooth API

Widcomm, “Wireless Internet and Data/Voice Communications”, offers several software products that provide a “complete solution” for building interoperable Bluetooth capability to embedded systems, Windows and Windows CE. Developers can use Widcomm software to create applications that will cut their time-to-market using “innovative and feature-rich products” (Widcomm FAQ, 2004).

1.6 PROJECT AIMS

The overall goal of this project is to investigate available Bluetooth application programming interfaces that will support device connectivity, basic data transfer and a flexible feature-rich environment for Bluetooth application development.
CHAPTER 1. INTRODUCTION

An investigation will be conducted to discover the Bluetooth-specific functionality provided by the Windows and Widcomm APIs, the limits to their flexibility and the feasibility of developing a custom-specific API. Comparisons will be made by writing experimental applications to test the usability and rapid-prototyping possibilities of each API. Using this information, the most suitable API to use with AWIS will be assessed.

Further sub-goals are mastering wireless application development, testing the viability of implementing iterative test methodology, gaining insight into Bluetooth protocol and implementation from both a research and development point of view.
Chapter 2

Methodology

A Bluetooth device requires some make of Embedded operating system and a development platform for application development. There should be software available for connecting the device to a PC and downloading any files as needed. Development of these devices should be straightforward and hardware should not present any issues. For the several APIs selected, criteria for comparison should be created according to the functionality requirements of AWIS as well as general implementation issues.

2.1 MATERIALS

2.1.1 Hardware

An iPAQ Pocket PC h4150 (h4100 series) was selected for development as it offers a familiar user-interface and allows the developer to ignore the hardware details and concentrate on the software. Ideally the hardware should be ignored, this is because in the actual implementation of AWIS a possible custom made device would probably have to be manufactured (thus hardware constraints should not be an issue yet). Ultimately, in the actual implementation of AWIS, a headless device would probably used instead of an iPAQ to allow for the least amount of power consumption, reduction in size and physical robustness.

This iPAQ had an Intel 400 MHz processor, 32 MB Flash ROM and 64 MB RAM (56 MB available to users), thus providing sufficient processing power. The device also provided a convenient environment for testing as it was easily connected to the PC via
Active Sync software and was small and portable. A DLink dongle was used to connect the PC to the iPAQ when necessary. These were not possible to program as a separate development kit is needed, but were useful for testing the detection of broadcasting Bluetooth MAC (Media Access Control) Addresses.

2.1.2 Software

2.1.2.1 Development Environment

There has been much debate over Embedded Linux versus Embedded Windows CE. According to SixNet (2004) Windows CE is an embedded kernel which does not allow for the simple addition of applications. Embedded Linux’s strengths are its maturity (based on 30 year-old UNIX), stability, availability of source code and low cost. Linux does not rely on a GUI and thus may be more suitable for some Embedded applications such as headless devices. Despite this popular belief that Windows CE does not support headless devices (Hall & Maillet 2004), it is possible when custom building a custom OS using platform builder to leave out the graphics drive interface GWES (Graphical Windowing and Event Subsystem). Thus it is possible to create headless Windows CE devices (Intelligraphics, 2004).
Krasner (2003) claims that Embedded Windows CE fairs better from the point of view of number of developers needed and time taken for development as well as the amount of available support for developers. A rebuttal by Lehrbraum (2003) points out that there plenty of Linux support available and some skewness in Krasner’s findings – a bias towards Windows. Statistics shown by Lanfear and Balacco (2003) do agree to an extent with Krasner, as the number of software engineers to each project and time-to-market is greater for Linux than Windows.

Thus, in the interests of rapid development, Windows CE was the platform selected. In addition, this platform offers a small footprint for devices and provides support for Pocket PC 2003 (Hall, 2003). Microsoft also offers an IDE and build environment which is described by a Windows critic Lehrbraum (2003) as “very good” and “sophisticated”. The development environment used was Microsoft Embedded Visual C++ 4.0. The Pocket PC 2003 SDK provided the Virtual Serial Ports and Winsock APIs; and the Widcomm Pocket PC SDK provided Widcomm API.

2.1.2.2 Application Programming Interfaces

According to Boling (2002), working with Windows APIs isn’t “easy, clean, or quick.” and this is due to the flexibility of the Blue2tooth standard and the complexity of the service discovery protocol. Despite this, the Windows APIs offer Windows CE support and are recommended by Microsoft (Boling, 2002; MSDN, 2004). Both Winsock and Virtual Serial Ports are mature, well understood protocols (Quinn, 1998).

There are many proprietary Bluetooth SDKs available (Chapter 1). Widcomm is probably the most widely used API as most Bluetooth devices have the Widcomm stack deployed on them (Experts Exchange, 2004) and would thus be a good choice as a propriety example API for evaluation.
Further to this, these two APIs have different approaches to Bluetooth application development. Widcomm offers developers “protocol-layer direct access” to the following layers of the Bluetooth stack: L2CAP, RFCOMM, OPP, FTP, SDP, LAP and OBEX (Widcomm, 2003). Whereas the Windows APIs attempt to insulate the developer from the Bluetooth Stack completely (Quinn, 1998). Each approach to development may contribute to the API’s performance in terms of rapid development and functionality. Furthermore, this data would be relevant to Bluetooth developers due to the lack of a standard Bluetooth API (Gratton, 2003). Both Protocol Stacks are shown in Figure 2.1 and 2.2.

Figure 2.1: Windows CE Bluetooth Stack (from MSDN, 2004)
2.2 APPROACH

According to Experts-Exchange (2004) the only way to compare Bluetooth APIs is to use and test the tools available directly, in order to observe how they handle enabling Bluetooth development. Thus it was necessary to develop a test application using each API and make several measurements during development as well as the results returned by the application.


2.2.1 Iterative development of Test Application

The approach followed was to examine sample code and available documentation to write small test applications that would run on the iPAQ. The applications developed were not necessarily complex, but attempts were made to get as low down in the technology hierarchy as possible, without significantly affecting speed of development. The development of these applications was an iterative process. For each iteration, the effects of the application were studied and any insight gained was applied to the next iteration. Attempts were made to simulate the types of routines that would be required for the AWIS.

2.2.1.1 Sample Code

2.2.1.1.1 Virtual Serial Ports
A simple “tty” (Terminal Type) emulation application was obtained from the Windows CE SDK. This was a terminal emulator that, if a HyperTerminal application was running on a PC with a Bluetooth dongle, the iPAQ running the application was able to communicate with the HyperTerminal. Any input typed on the PC would appear on the iPAQ and vice versa. This application was chosen as it incorporates the following:

- Scanning for available Bluetooth serial communications ports
- Connecting to ports: open, configure and close
- Sending data: perform read/write operations

2.2.1.1.2 Winsock
BTSearch was also provided by the Windows CE SDK. This demonstrated an implementation of simple Bluetooth device discovery via the Winsock 2 API. A listbox control displayed addresses of any discovered Bluetooth devices. Functionality that was provided:

- Device Inquiry
BTHello program provided by Boling (2002). The following requirements were stated: “The application must be running on two Windows CE devices that use the Windows CE stack for the application to work.” This application could act as a client or server. Initially the client application searched for Bluetooth devices in range and displayed them in an output Window. The user could then click the “Say Hello” button and the device would connect to the BTHello service on the server device. Thus another device had to be set-up as a server by running the program and ticking the “Server” option. Once the client was connected to the server, the client could send the server a string and close the connection. The server then read the string and displayed on its output window.

This application demonstrated the following functionality:

- Device Inquiry: client could discover any devices in range.
- Service Creation: set-up and register service to be advertised in the SDP record.
- Service Discovery: server service discovered by client.
- Setting up of Sockets for communication: binding to a Bluetooth address.
- Simple data-transfer: using the socket created.

2.2.1.1.3 Widcomm

The BlueTime application was provided by the Widcomm SDK is an L2CAP Time monitor, creating a client-server L2CAP connection. Each second each side sent its local time to the other side. Once the user established a connection, the server created a new service and then waited for a client to connect. When the client performed a device inquiry and service discovery, the two sides connected and were able to communicate. BlueTime performed the following tasks:

- Device Inquiry: client could discover any devices in range.
- Service Creation: set-up and register service to be advertised in the SDP record.
- Service Discovery: server service discovered by client.
- Open and close an L2CAP connection – client side and server side.
- Simple data-transfer: write data to the L2CAP port.
CHAPTER 2. METHODOLOGY

A custom application, with increasing functionality, was written using the L2CAP class (which interacts with the L2CAP layer of the stack) that would scan for any Bluetooth devices in range. Since it operates on a lower layer and offers packet-orientated connections, experiments in the L2CAP class would be ideal for the AWIS - the communication required was be fairly simple. Any devices found would be logged as an Encounter with a time stamp. The application also checked to see if any devices had gone out of range. For uploading information onto a Beacon or another device offering an “upload” service, the L2CAP Connection class was used. The following information was written to a text file to track the progress of the application; whether the all the class objects had been instantiated correctly, the status a connections to a Beacon (if any), how many devices discovered and any Encounters logged.

2.2.2 Comparison Criteria for APIs

For each API, the applications were assessed and compared according to the following criteria:

1. In terms of enabling development of AWIS:
   1.1 Device Inquiry
   The results of this search should return the Bluetooth MAC Address and possibly the device name of each device in range.
   1.2 Service Discovery
   This is should allow the search of available services on each discovered device. It should also be possible to create a custom service and publish it in the SDP record for discovery by other devices.
   1.3 Data-transfer
   This would be fairly simple; sending or receiving an Encounter either as an object or merely as a String, as well as accepting data.
2. General development

2.1 Functionality – how much low-level versus high-level functionality is provided? Does the level of detail hinder or aid the developer?

2.2 Flexibility – how many options is the developer given? Are there many constraints when programming?

2.3 Compatibility - will this run on any device? What kind of limitations are there?

2.4 Rapid prototyping – how painstaking is development? Is it possible to roll out a possible solution in a small amount of time? Will this fit in with the iterative methodology?

2.5 Relation to stack - Where does the API sit in relation to the stack? How much interaction is available and does this enable more effective development?

2.2.3 Steps Toward the Automated Wireless Identification System

There were two main areas of development – Bluetooth device discovery and data transfer. The final application needed to log all Encounters and upload them to a Beacon or device offering and “upload service” when in range. The following stages were undertaken to develop the AWIS test application, each stage built on from the previous one:

Stage 1: An application for simple inquiry was written, searching for a Bluetooth device in range. Once a device was discovered, its Bluetooth MAC address was displayed on the screen.

Stage 2: The application was modified to discover more than one device and display a static list of all MAC addresses once the inquiring device had finished its inquiry.
CHAPTER 2. METHODOLOGY

Stage 3: The list of devices in range was made dynamic to allow for the removal of devices when they went out of range. This was accomplished by writing an algorithm for searching for devices, storing them in a temporary current Encounter list and then running the inquiry again. If the MAC Address was not discovered after two more inquiries, known as “touches”, the device must have gone out of range and was removed from the list of current Encounters.

Stage 4: An Encounter class was created. This held the following properties such as local and remote Bluetooth MAC addresses, start time and state.

Stage 5: As inquiries took place a list of such Encounters accumulated in a “store” object. When a discovered device went out of range, the Encounter was marked as closed.

Stage 6: Bluetooth service discovery was built in to allow devices to differentiate between Beacons (for Encounter log upload) and other Bluetooth devices. Further to this an upload service was advertised on several Bluetooth devices to allowing for “swopping” of closed Encounters.

Stage 7: A separate server application was written to accept a list of Encounters.

Stage 8: If a Beacon was found, the device could send all of its logged or “closed” Encounters to the Beacon. The Encounters in the list would then be marked as “uploaded” on the sending device.

2.2.4 Field Testing the Final AWIS Application

The application was tested by running the application on the iPAQ. Dongles and other iPAQs were brought “into range” and then “out of range” (either by moving more than 10 meters away or by switching the Bluetooth radio off). Simple message boxes were used to show the current state of detection. For example; each time another device was
discovered a message would pop-up displaying the discovered device’s MAC address. The program was run in a loop and if a device had gone out of range a message would display the time of last discovery, duration of Encounter as well as the MAC addresses of the local and remote devices.

The field test of the final AWIS application was done by involving as many as Bluetooth devices as possible. All iPAQs were loaded with the AWIS application and all other Bluetooth devices were switched on. The iPAQs collected Encounters. A Beacon – i.e. an iPAQ with a wireless LAN connection, was attached in a prominent position to collect Encounters. As any iPAQ passed it, Encounter logs were automatically uploaded to the Beacon. The AWIS applications on the iPAQ’s were also able to swap Encounter logs if they came into range of each other.

An Encounter log file was written on each device and examined for the following:

1. **AWIS Device Discovery**
   - Did the AWIS application collect all the devices in range?
2. **Time Taken for Encounter Log**
   - What sort of time did the Encounter logging take?
3. **Encounter Upload**
   - Was the information uploaded correctly? Were there any Encounters lost?
4. **Upload Interference**
   - Did any AWIS applications on iPAQs interfere with one another when attempting to upload to the Beacon?

### 2.3 SUMMARY

The iPAQ and Windows CE environment were selected for their ease of use, and thus comparisons of APIs could be done in a familiar and straightforward environment. Winsock and Virtual Serial Ports are the APIs recommended by Microsoft. Widcomm API was chosen as a relevant proprietary API due to its dominance in the Bluetooth device market and its contrasting approach to application development. The functionality
offered by each API was demonstrated by several sample applications. A comparison criterion was formulated according to the functions needed to create AWIS as well as general development issues such as flexibility, compatibility and rapid prototyping. The AWIS application was built up in stages of increasing functionality and would need to be field tested to measure its success.
Chapter 3

Results

The investigative work for this project yielded a range of findings, some in the form of practical development experience, and some in the form of more qualitative comparisons between APIs. This chapter reports on the practical experience gained and the results of more rigorous evaluations according to criteria set out in the previous chapter (Section 2.2.2).

3.1 GENERAL DEVELOPMENT EXPERIENCE

3.1.1 Hardware Related Problems

The sample code provided by the Windows CE SDK for Winsock would not run when the DLink dongle was plugged into the PC. It was also not possible to program this particular dongle as they do not support the Windows Bluetooth API. When attempts were made to explore Bluetooth devices using the iPAQ’s native software, the dongle was not discovered. It was only discovered by the iPAQ when it was first connected via Active Sync.

It was found that the iPAQ uploaded the build files fairly quickly. The only issue arising from programming with the iPAQ was that the Bluetooth connections could not be closed from any of the Bluetooth applications. This means that occasionally the Bluetooth radio
CHAPTER 3. RESULTS

would stop functioning correctly and a soft-reset of the iPAQ was necessary. It was often found that the iPAQ was unable to detect any other Bluetooth devices, even when they were in close proximity to each other, i.e. less than a meter away. The only possible solution to this problem was to reset the iPAQ, after which it would be able discover other devices again.

3.1.2 Visual Studio 2003 Development Environment

Development in the Embedded Visual C++ development environment was straightforward as the class Wizards built most of the code required, in particular the general Windows application step-up and graphical interface controls were automatically completed for the developer. Although an emulator was provided in the Pocket PC 2003 SDK, it did not support Bluetooth development and thus could not emulate any Bluetooth functionality. There were a few issues with Windows CE and Pocket PC backward compatibility. It was not possible to program an iPAQ device with the Pocket PC 2003 SDK if the iPAQ was running an operating system older than Windows CE 4.2.

Files are automatically uploaded once they are built in C++ and the resulting connection of the iPAQ to the PC, via ActiveSync software, was found to be very quick. Programming and error checking was thus rapid, since errors often only occurred as runtime errors when running applications on the iPAQ.

3.2 WINDOWS API’s

3.2.1 Virtual Serial Ports

A terminal type sample application provided by the Windows CE SDK was used to experiment with Virtual Serial Ports over Bluetooth. It was necessary to manually set up a serial connection between the iPAQ and dongle using the iPAQ’s native Bluetooth
Manager. No connection could otherwise be made as no device inquiry routines were provided by the Virtual Serial Port API. Once Bluetooth devices were discovered using the iPAQ’s Bluetooth Manager software, a paired connection was made using the Generic Serial Profile.

The API provides methods for scanning for an available port to connect to but this was found to work irregularly in the Terminal Type program. The correct port to connect to via the dongle thus needed to be hard coded into the program using the CreateFile() method. According to the Bluetooth settings on the iPAQ, inbound was COM 8 and outbound COM 6, since Bluetooth uses one port for incoming data and one for outgoing. A Bluetooth connection was achieved by setting the Terminal Type program to connect to COM 6 for sending data and configuring the HyperTerminal to receive on COM 5. These connections were not closed properly by the program and the iPAQ needed a soft-reset.

Once everything was connected correctly, data could be sent and received using the ReadFile() and WriteFile() functions. Low-level settings such as the baud rate, number of data bits, parity and stop-bits had to be identical on both applications (running on the PC and iPAQ). Thus most of the stages of AWIS (see Chapter 2) could not be completed using Virtual Serial Ports, except for Stage 7 and 8, where connections and data-transfer were required (Table 3.1).
Table 3.1: Comparison of usage of Bluetooth APIs on an iPAQ h4150

<table>
<thead>
<tr>
<th>AWIS</th>
<th>Virtual Serial Ports</th>
<th>Winsock</th>
<th>Widcomm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Inquiry</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>RegisterDevice(addr</td>
<td></td>
<td>StartInquiry()</td>
</tr>
<tr>
<td></td>
<td>of Bluetooth device)</td>
<td></td>
<td>OnDeviceResponded()</td>
</tr>
<tr>
<td>Service Discovery</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AddService()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReadDiscoveryRecords()</td>
</tr>
<tr>
<td>Data-transfer</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>ReadFile()</td>
<td></td>
<td>L2CapConn</td>
</tr>
<tr>
<td></td>
<td>WriteFile()</td>
<td></td>
<td>OnDataReceived()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Obex</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RFComm</td>
</tr>
<tr>
<td>GENERAL</td>
<td>Functionality</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Low-level only</td>
<td></td>
<td>Low-level and high-level</td>
</tr>
<tr>
<td>Flexibility</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Limited</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatibility</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>Runs with Widcomm and Windows Stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rapid prototyping</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Relation to stack?</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Direct protocol access</td>
</tr>
</tbody>
</table>

3.2.2 Winsock

Each time attempts were made to run either the BTSearch or the BTHello Winsock sample programs, the following error occurred: “Error 10050: Network down”. According to the Winsock documentation, this indicates that “A socket operation encountered an unavailable network” (MSDN, 2004).
 CHAPTER 3. RESULTS

It is suggested that the user should check Winsock, protocol stack, network driver, or network adapter configuration. The iPAQ’s network component was found to be working correctly as it was possible to connect to other Bluetooth devices using the iPAQ’s Bluetooth Manager.

What was discovered, however, was that the iPAQ Bluetooth protocol stack was a Widcomm implementation as opposed to the Windows stack required by Winsock (Experts Exchange, 2004). This stack is only deployed on a few Bluetooth devices; currently most mobile devices use the Widcomm stack (Hall & Moore, 2004). None of the stages of AWIS (Chapter 2) could thus be completed using Winsock and a test application could not be developed (Table 3.1).

3.3 WIDCOMM API

The Widcomm SDK allowed for the full completion of each stage of AWIS and thus an iterative test application was successfully completed. Device inquiry, service discovery and data-transfer were all accomplished using this API (Table 3.1).

The CBtIf class (main class for device inquiry) provided a StartInquiry() method which called OnDeviceResponded() each time a Bluetooth device in the area responded. Since there could be more than one OnDeviceResponded() call for the same device, it was necessary to disregard duplicates.

The following data was returned:

- Bluetooth MAC Address of remote device,
- Class of responding device,
- User assigned name of device,
- Whether or not the remote address was currently connected to the local device.
CHAPTER 3. RESULTS

StartDiscovery() requested service discovery for a specific device with its Bluetooth MAC address supplied. The discovery results were stored in a discovery database and results were only removed if a device failed to respond to an inquiry. Once discovery was completed, in order to ReadDiscoveryRecords(), the developer had to supply the following: Bluetooth address, number of discovery records to read and a place to store the information returned by the function.

From the server side, it was necessary to advertise a service. For AWIS, the Beacon would advertise an upload Encounters service to the other Bluetooth devices. The CL2CapIf class (the main class for communicating with the L2Cap layer) works in conjunction with the CL2CapConn class; the main class for creating L2Cap connections. The client and server had to register a PSM value (Protocol and Service Multiplexer) with the L2Cap Protocol layer. The server called AssignPsmValue() without the PSM parameter to allow the L2Cap to assign a value. During service discovery the client called the identical function but with the PSM value previously assigned. Before any connections can take place Register() must be called and the security level set. Since security is not important for AWIS, no security was set using the BTM_SEC_NONE value.

The CL2CapConn class controls L2Cap connections. Using this class, the assigned server device can Listen(), Accept() or Reject() incoming connections. OnIncomingConnection() and OnDataReceived() were further functions used by the server to wait for connections and respond when data was received from a client. The client created a connection to the server by calling Connect(). Write() writes data once the L2Cap connection has been set up, allowing the data to be sent as a string to the server.

In addition to the methods mentioned above, the Widcomm API provided many other classes that could be used to extend the functionality of a Bluetooth application. The following accompanying functions gave the developer an additional level of control:
CHAPTER 3. RESULTS

- GetLocalDeviceInfo() – the information of the local device, was stored in public instance variable. This was used for generating an instance of an Encounter: both the local and remote device information must be stored in the log.
- SwitchRole() – allowed the device to switch from a master to a slave or vice versa.
- OnStackStatusChanged() – this provided for some manual or automatic recovery if the power was removed.
- RadioOn()/RadioOff() – it was possible to control the Bluetooth radio in this way. The stack could also be reloaded with ReloadStack().
- Reconfigure() – client or server applications could call this to reconfigure an existing connection.
- GetConnectionStats() – this was used for testing. It retrieved the following current connection statistics:
  - Returned Signal Strength Indicator (increasing negative values if too far apart, 0 for optimal distance and increasing positive values as the devices are closer together).
  - Bytes sent and received – number of bytes since the connection was established. Only bytes sent or received by application were counted.
  - Duration – time since connection was established.

Further explanation of above functions can be found in the Widcomm Documentation (Widcomm, 2003).

3.4 FUNCTIONAL DESCRIPTION OF THE FINAL AWIS APPLICATION

The AWIS application comprised of six main classes: Encounter, Store, AWISTag, UploadController, InteractionController and BTDev.
• **Encounter**
The data structure which contained the interaction details: local device MAC address, foreign device MAC address, foreign user-assigned name, time of creation, “touch time” (last time device responded to inquiry) and state (*open*, *closed*, *uploaded* or *deleted*).

• **BTDev**
This provided a data structure for storing information of any Bluetooth device that responded to an inquiry. The MAC address, device class, user assigned name and whether it was connected or not to the local device were all provided by the Widcomm API. The following members were later assigned by other classes: a PSM value (assigned for L2Cap connections) and whether the device was a Beacon or a device willing to accept Encounters.

• **Store**
Contained an array of Encounters and controlled the memory management of the Encounters. Additional functions provided:
  - o **TouchEncounter**: updated the touch time if a device responded to the last inquiry.
  - o **UpdateStore**: if a device did not respond for the last two device inquiries the Encounter’s state was changed to “closed”.

• **AWISTag**
This class’s Run method was called repeatedly by the main program and performed an inquiry (details of responding devices were stored in BTDev) and service discovery to find out if there were Beacons or other devices advertising Upload services.

• **UploadController**
This attempted to deposit Encounters via Bluetooth by trying to connect to an available AWIS server (either a Beacon or other device).

• **InteractionController**
CHAPTER 3. RESULTS

Received a list of all responding devices and attempted to “touch” each Encounter marked as “opened”, i.e. check if a device responds to a further inquiry. This class also called UpdateStore and attempted to swap data with any available servers.

The basic relationship between the AWIS classes and the Widcomm SDK classes is shown in Figure 3.1. This illustrates where the main Widcomm classes (BtIf and L2CapConn) were used to enable the development of the AWIS application as well as the AWIS class relationships. Note that the SdpDiscoveryRec class has been omitted for clarity but was used by the InteractionController and UploadController classes. For further implementation details and limitations of the application, the reader is referred to Clayton (2004) and King (2004). The full listing of the AWIS application code is included on this project’s CD.
CHAPTER 3. RESULTS

3.5 FIELD TESTING THE AWIS FINAL APPLICATION

3.5.1 AWIS Device Discovery

Initially the AWIS application could not discover all the devices in its range. This was due to a practical implementation problem: if a particular device is in inquiry mode, no other device was able to discover the inquiring device. The solution was to allow each discovering device to sleep for a random amount of time before going into discovery mode.

Figure 3.1: Relationship between AWIS Application Classes and Widcomm SDK Classes
CHAPTER 3. RESULTS

3.5.2 Time Taken for Encounter Log

Because of the discovery problems above, the time for an interaction to occur was quite long. It was also found that service discovery also took a significant amount of time to accomplish. Sleeping, device inquiry and service discovery combined made the entire process fairly slow. In order to discover devices in range and potentially upload Encounter logs if the relevant service was discovered, the AWIS application needed to hover near the Beacon for a few minutes.

3.5.3 Encounter Log Upload and Interference

A log was written to a text file on the iPAQ each time an Encounter was created. This file was used to check that the Encounter information had been correctly uploaded. By comparing this file to the actual course of events, for example purposefully bringing certain Bluetooth devices in and out of range at recorded times, it was clear that Encounter information was correctly collected. All the Encounter logs from the iPAQs running the AWIS application were collected and all the closed Encounters compiled into one text file. An extract of file is shown in Table 3.2.

Table 3.2: Extract of Collected Closed Encounters Log

<table>
<thead>
<tr>
<th>LOCAL MAC</th>
<th>FOREIGN NAME</th>
<th>FOREIGN MAC</th>
<th>LAST TIME DEVICE RESPONDED</th>
<th>ENCOUNTER CLOSED</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:0:28:a0:1a:1a</td>
<td>SamWireless1</td>
<td>0:80:c8:26:6d:ac</td>
<td>26/10/2004 21:00:37</td>
<td>26/10/2004 21:00:37</td>
</tr>
</tbody>
</table>
This was the type of information that was uploaded onto a Beacon by each AWIS application. The Beacon was designed to block one AWIS application if two came into range of the Beacon at the same time, thus if the application stay at the Beacon for long enough, there was a possibility that information would not be uploaded. With the exception of any device problems, for example the memory being full, the loss of data was unlikely provided the device passed the Beacon at another time (Clayton, 2004).

3.6 SUMMARY

Although offering basic data-transfer and Bluetooth connections, the Virtual Serial Port API was found to be limited in its capabilities for developing an AWIS application. No comparisons could be performed using the Winsock API as it was found to be incompatible with the Widcomm stack installed on the iPAQs. The Widcomm API used to develop the final AWIS application for the field test and a log file was used to confirm that the application was correctly capturing Encounters as Bluetooth devices came into range.
Chapter 4

Discussion

The information collected in the previous Chapter could now be compared against other sources. Possible reasons for the suitability and limitations of the hardware and software used will be explored. The extent of the functions provided by each API is listed as well as their flexibility, compatibility, rapid prototyping support and level of stack access are compared in this Chapter.

4.1 GENERAL IMPLEMENTATION ISSUES AND LIMITATIONS

The iPAQs were a good choice for proof of concept development enabling the developer to come to grips with AWIS’s potential and limitations. iPAQs were sufficient for a pilot study but not an ideal choice for the actual implementation of AWIS due to the reliance on a GUI. The GUI would be excessive for AWIS as a graphical display is unnecessary and a power consumer when attached to an animal. This result, however, corresponds to the project’s overall methodology of iterative development. From this proof of concept stage, the high level steps of how the system should behave have been laid out. The next step of implementation could therefore be considered, such as the development of a headless device as alternative hardware to the iPAQ.
The main limitation of using the iPAQ was the limitation placed on the APIs as Winsock could not communicate with the Widcomm stack. In addition to this there were a few difficulties with the Bluetooth radios as occasionally they could not detect other Bluetooth devices that were within range. This could be due to the “ad-hoc nature of Bluetooth networking” (Clayton, 2004), or limited to the radio’s performance (Chapter 3).

### 4.2 COMPARISON OF WINDOWS APIs and WIDCOMM API

Since the Widcomm API operates on a lower layer and offers packet-orientated connections, experiments were done with L2CAP interface. This would be the most appropriate for the AWIS as the communication required would be fairly simple. The Virtual Serial Ports API was the only Windows API compared to the Widcomm API because the iPAQ’s Bluetooth protocol stack was a Widcomm implementation and was not compatible with Winsock. It was thus not possible to program the iPAQ using the Winsock API. This also offers a possible explanation of why the port scanning and closing of Bluetooth connections in the Virtual Serial Port samples did not execute as expected on the iPAQ.

When examining the Winsock example code given, many of the methods appear to be quite complicated to set up. Boling (2002) states that “Although the function names imply a simple iterative search, the parameters required for the search are daunting”. The reasons cited for this are the flexibility and complexity of the Bluetooth protocol for routines such as service discovery (Boling, 2002). Winsock is described as “cumbersome” but offers most of the functionality required by a Bluetooth developer (Holgate, 2003). Despite the level of detail demanded by Winsock, its stack interaction is limited to the very top layers.
4.2.1 Device Inquiry

There was no facility in the Virtual Serial Port program to initiate inquiry, therefore the two devices needed to be manually paired using the native software on the iPAQ. There was also no means to scan for all the devices in range, thus this is probably not a viable option for AWIS. During client/server connection set-up the server-side did not need to be aware of the client, but the client had to know the Bluetooth address and GUID of the service of the server in order to register a virtual COM port driver (MSDN, 2004). If a client wanted to search for a server’s Bluetooth address, Winsock is needed to provide WSALookupService functions (Boling, 2002).

To initiate a scan for devices in the area using the Widcomm API, all that was needed was the StartInquiry() method. For each device response OnDeviceResponded() was automatically called. This function allowed the programmer to add their own instructions and provided useful data such as the Bluetooth MAC address and user specified name.

4.2.2 Service Discovery

By examining sample code and documentation, it was found that service discovery is not supported by the Virtual Serial Ports API. When setting up a server, a virtual COM port driver must be registered and once the driver is opened the RFCOMM channel is queried (Boling 2002). RegisterDevice() registers the Bluetooth stack with the COM port. (MSDN 2004). The client must still have the address of the device and GUID of service before being able to discover any available ports.

Widcomm supports service discovery and devices therefore could be interrogated for their available services, as well as register their own services using the L2capIf and SdpDiscoveryRec classes. During the set-up of the service a PSM value is required
by the Bluetooth protocol. The set-up of this was complicated (Chapter 3) but necessary as a PSM value is used by the L2Cap layer to identify RFCOMM traffic and used by a client to connect to a server (Hole, 2004).

4.2.3 Data Transfer

Data can be sent using Virtual Serial Ports once a connection between the server and client has been created and CreateFile() was called. WriteFile() or Readfile() was used to send or receive data in the form of a character stream. The Widcomm API differs by providing several different classes (L2Cap Connections, Obex and RFComm) to achieve data transfer. RFComm was not selected as the COM port emulation facility in Windows CE has its own separate layer on the stack above both Obex and L2Cap (Figure 2.2). The Obex class requires an object to be created and a number of header fields set. This object, nonetheless, is sent over the L2CAP layer and thus Obex has a higher overhead than L2CAP. The L2CAP Connection (L2CapConn) class operates at the lowest layer to communicate information and thus may be the most viable option as Encounters can be sent as strings.

4.2.4 Functionality and Flexibility

The types of functions provided by the Virtual Serial Ports API present a fine level of control for serial communication; the developer is able to control the stream of data explicitly (Table 4.1). Thus low-level functionality for Virtual Serial Ports is well supported, for example, one character can be sent at a time and the port can be instructed to wait on a specific event.

The nature of the connections themselves could be seen as flexible as data-transmission settings are down to the hardware level (baud rate, number of data bits, parity and stop-bits), in contradiction to Senese’s claims (Chapter 1). Both sides of communication, i.e. both client and server device, must be configured manually in a similar way (MSDN, 2004).
Despite this low-level functionality, Bluetooth-specific functions such as device inquiry and service discovery are not supported by the API. This therefore limits the overall functionality of the API as only general serial communication functionality is provided (Table 4.1).
Table 4.1: Listing of Methods available for the Virtual Serial Ports API (Microsoft, 2004)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegisterDevice()</td>
<td>Registers a new device. A handle to a new device is returned if successful.</td>
</tr>
<tr>
<td>DeregisterDevice()</td>
<td>Deregisters a device</td>
</tr>
<tr>
<td>CreateFile()</td>
<td>Creates a COM port. Returns a handle that can be used to access the port.</td>
</tr>
<tr>
<td>ReadFile()</td>
<td>Reads data (characters) from a port, assuming the port has been successfully opened.</td>
</tr>
<tr>
<td>WriteFile()</td>
<td>Writes data to a serial port.</td>
</tr>
<tr>
<td>CloseHandle()</td>
<td>Closes COM port.</td>
</tr>
<tr>
<td>ClearComError()</td>
<td>Retrieves information about a communications error and reports the current status of the device.</td>
</tr>
<tr>
<td>SetCommMask()</td>
<td>Specifies a set of events to be monitored by the serial driver (allows one to wait for specific event).</td>
</tr>
<tr>
<td>GetCommMask()</td>
<td>Retrieves the event mask for a specified device</td>
</tr>
<tr>
<td>WaitCommEvent()</td>
<td>Waits for an event to occur on the device – the events monitored are contained in the event mask associated with the device handle.</td>
</tr>
<tr>
<td>SetCommState()</td>
<td>Configures device according to specifications in device-control-block structure (baud rate, parity and so on).</td>
</tr>
<tr>
<td>GetCommState()</td>
<td>Fills in the device-control block structure with current control settings for a specified device.</td>
</tr>
<tr>
<td>GetCommTimeouts()</td>
<td>Retrieves the time-out parameters for all read and write operations on the device.</td>
</tr>
<tr>
<td>SetCommTimeouts()</td>
<td>Sets the time-out parameters for all read and write operations on the device.</td>
</tr>
<tr>
<td>SetupComm()</td>
<td>Initializes the communications parameters for the device: handle to the device, size of its internal input and output buffer.</td>
</tr>
<tr>
<td>GetCommProperties()</td>
<td>Returns information to determine the capabilities of a device.</td>
</tr>
<tr>
<td>SetCommBreak()</td>
<td>Suspends character transmission – stops serial stream and puts port in a break state.</td>
</tr>
<tr>
<td>ClearCommBreak()</td>
<td>Resumes transmission, places port in non-break state</td>
</tr>
<tr>
<td>PurgeComm()</td>
<td>Clears all characters from output or input buffer of the device (serial driver).</td>
</tr>
<tr>
<td>EscapeCommFunction()</td>
<td>General method of controlling serial driver, allowing setting and clearing the state of specific signals on the port.</td>
</tr>
<tr>
<td>GetCommModemStatus()</td>
<td>Returns the status of the modem control signals</td>
</tr>
<tr>
<td>TransmitCommChar()</td>
<td>Transmits a single character inserted at the front of the transmit queue.</td>
</tr>
</tbody>
</table>
CHAPTER 4. DISCUSSION

Widcomm provides a number of useful functions that are not provided by the Virtual Serial Ports API (Chapter 3). This illustrates the level of control the developer is given. There is less fine control of actual data transmission but more high-level functionality is provided as well as core Bluetooth-specific functionality such as device inquiry and service discovery as be seen in Table 4.2.

Table 4.2: Listing of Classes available for the Widcomm API (Widcomm, 2003)

<table>
<thead>
<tr>
<th>DK Class</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBtIf</td>
<td>Provides interface level management functions, e.g., methods for doing inquiry and service discovery.</td>
</tr>
<tr>
<td>CL2CapIf</td>
<td>Interfaces to L2CAP for Protocol/Service Multiplexor (PSM) allocation &amp; registration, and security settings.</td>
</tr>
<tr>
<td>CL2CapConn</td>
<td>Controls L2CAP connections.</td>
</tr>
<tr>
<td>CSdpService</td>
<td>Manages an SDP service record.</td>
</tr>
<tr>
<td>CSdpDiscoveryRec</td>
<td>Contains an SDP discovery record and methods to query it.</td>
</tr>
<tr>
<td>CRfCommIf</td>
<td>Interfaces to RFCOMM for Service Channel Number (SCN) allocation and security settings.</td>
</tr>
<tr>
<td>CRfCommConn</td>
<td>Controls RFCOMM connections.</td>
</tr>
<tr>
<td>CFtpClient</td>
<td>Provides the client-side interface for FTP.</td>
</tr>
<tr>
<td>COppClient</td>
<td>Provides the client-side interface for OPP.</td>
</tr>
<tr>
<td>CLapClient</td>
<td>Provides the client-side interface for LAN access using PPP.</td>
</tr>
<tr>
<td>CSppClient</td>
<td>Provides the client-side interface for SPP COM port connections.</td>
</tr>
<tr>
<td>CSppServer</td>
<td>Provides the server-side interface for SPP COM port connections.</td>
</tr>
<tr>
<td>COBexServer</td>
<td>Provides the server-side interface for OBEX</td>
</tr>
<tr>
<td>COBexClient</td>
<td>Provides the client-side interface for OBEX</td>
</tr>
<tr>
<td>COBexHeaders</td>
<td>Container class for all OBEX header structures</td>
</tr>
<tr>
<td>COBexUserDefined</td>
<td>Container class for the user defined type of OBEX header</td>
</tr>
<tr>
<td>CDunClient</td>
<td>Provides the client-side interface for DUN connection.</td>
</tr>
<tr>
<td>CPrintClient</td>
<td>Provides the client-side interface for printing.</td>
</tr>
</tbody>
</table>
4.2.5 Compatibility

Despite its implementation on Windows CE being unique, the Virtual Serial Ports were found to run on both the Windows Bluetooth Stack as well as the Widcomm Stack (Hall & Moore, 2004). Widcomm’s weak point is its lack of compatibility. The API will only communicate with a Widcomm Stack. Its functions are unique, whereas Virtual Serial Ports can be used with both Windows and Widcomm stacks. The non-standardization of API’s is a problem with the Bluetooth development field as a whole (Cole, 2004).

4.2.6 Stack Interaction and Rapid Prototyping

Stack interfaces are not exposed by the COM port emulator but an API layer at the top of the Bluetooth stack is provided for opening connections to a remote Bluetooth device. Thus a virtual server or client COM port is created to “accept incoming or create outgoing RFCOMM connections” (MSDN, 2004). This is the limit to the level of stack interaction a developer has when using Virtual Serial Ports. The developer therefore does not need knowledge of the Bluetooth stack and its function to program with serial ports. This insulation can, however, severely limit the developer in terms of making full use of the Bluetooth protocol to accomplish most networking tasks.

In a different approach, the Widcomm API allows the developer access to most layers of the Bluetooth stack (see Figure 2.2). This links back to the functionality and flexibility provided by the API. Although the developer programs with knowledge of the Bluetooth stack, the functions provided to interface with each layer of the stack are high-level. This allows for the most functionality without hindering speed of development.
CHAPTER 4. DISCUSSION

The Virtual Serial Port API has the advantage of being familiar to programmers as it makes use of simple opening and closing of ports (Boling 2002). The set-up of the connections for COM ports is straightforward as the developer has to provide the Bluetooth MAC address and GUID. Serial ports are thus ideal for setting up one connection with a known device. The AWIS system by contrast, requires multiple connections to many devices whose MAC Address must be discovered initially.

The Widcomm Development Kit is conveniently divided into classes each providing separate functions, allowing the developer to use the corresponding class to the current system demands (Table 4.2). Some of the classes are inter-dependent but their relationships are fairly intuitive, as long as some knowledge of the Bluetooth specification is known. This does not need to be extensive, a basic understanding of the functionality provided by each layer of the Bluetooth stack is sufficient. Due to the reduced level of detail demanded on the developer, programming with Widcomm is faster and the developer has the opportunity to learn more about the Bluetooth’s protocol.

4.3 FIELD TESTING FINAL AWIS APPLICATION

4.3.1 AWIS Device Discovery and Time Taken for Encounter Log

According to the Widcomm technical support (Widcomm Technical Support, 2004) there is no limit on how many devices can respond, provided they are detected within the ten second inquiry timeframe. While testing the AWIS application this was found to be true with any number of devices being discovered, provided they responded within ten seconds.

According to Experts Exchange (2004), the number of devices discoverable will be limited by the number of services offered on each device. A list of available services must be discovered for each device discovered resulting in a long time for service discovery. Nevertheless, service discovery is an important part of the Bluetooth
specification (Gratton 2003) and has a higher overhead than just device discovery. Despite this, service discovery would be a necessary function in AWIS to upload Encounter data onto Beacons. There are alternatives to this method of distinguishing Beacons from other Bluetooth devices. The Beacons could possibly be given the same name and thus the device performing the inquiry will know to only perform service discovery on the devices with a certain name.

A device performing device inquiry could not be discovered by any other devices in range. This was confirmed by Widcomm technical support: “When a device does a scan for new devices normally no other devices will see it”. (Widcomm Technical Support, 2004) In order to work around this problem, the application was modified to sleep for a random amount of time before performing an inquiry. This added to the time taken for device inquiry and service discovery resulted in the time taken for Encounter logging to be between half and one minute (King, 2004).

4.3.2 Encounter Log Upload and Interference

Upon inspection of the Encounter logs it was possible to ascertain that the Encounters were correctly uploaded. From data similar to the data shown in Table 4.2, useful information can be obtained such as what the average time for Encounter could be calculated, which two devices had the most interaction (i.e. the most Encounters logged), which had the least interaction.
4.4 SUMMARY

The approach of iterative development of test applications on the iPAQ was found to be not only instructive of the functionality and flexibility of each API, but also their limitations. The high-level access to the Bluetooth stack in the Widcomm API contributed to the flexibility and rapid prototyping possibilities for Bluetooth development. The Virtual Serial Ports were found to support minimal Bluetooth-specific functionality, only typical networking functionality such as connections and data-transfer were possible. The differences in results suggest further investigation into other APIs and possible standardization of APIs or the stacks they reside on. Field testing of the AWIS application indicated that Encounters were successfully logged and possible extensions could be made to the system.
Chapter 5

Conclusion and Future Work

The purpose of this study was to serve as a basis for what developers would need to consider when choosing a development environment and programming interface. In this instance, developing and building a test application proved to be a viable method to compare APIs. Each API was compared by testing basic Bluetooth specific functions and implementing an AWIS application to discover general development concerns.

5.1 FINDINGS AND RECOMMENDATIONS

It was found that the Widcomm API offered the most functionality; i.e. low-level connectivity including device discovery, data-transfer and general flexibility. Several limitations for the Virtual Serial Ports API were found. Despite a fine level of hardware control for data transmission being provided, Bluetooth device inquiry and service discovery were not possible. The Winsock API failed to respond on the iPAQ due to the incompatibility of the Widcomm Bluetooth protocol stack. The Winsock API could thus not be tested on the iPAQ.
In terms of hardware, the iPAQs were well chosen for rapid development of a proof of concept system, despite the stack incompatibilities. Windows CE and Embedded Visual C++ provided an undemanding development environment for building, debugging and testing Bluetooth applications. The user interface on the Windows CE platform was initially chosen for its familiarity. This did, however, present some hindrances when testing the AWIS application, as all that was needed were text messages to show the progress of the program. A command-line interface would have been less cumbersome for testing purposes and is recommended for any future work.

Implementation issues and limits to the API only became apparent as the test application was developed. In this way it was possible to determine how the technology functions and investigate whether there were any ways to overcome any of the API’s limitations. Using this iterative methodology, it was discovered how the functionality provided by the APIs could best be applied to the AWIS.

5.2 FUTURE WORK

The possibility of extending the API was found to be unlikely. None of the Widcomm source code (as well as Winsock) was available to the developer. The only feasible extension would be for developers to write their own classes which could be utilized by the application, for example an Encounter class.

There are still many Bluetooth APIs available that were not considered in this study. For example Embedded Linux offers generic headless device support, or the standardized Java API claims to offer a similar environment to Windows CE. Many APIs are proprietary and their compatibility with the devices selected for development should be considered, as this was found be a significant obstacle in this study.

Since the scope of the comparison of APIs was limited to the functionality required by AWIS, further investigation as to how the APIs supply functionality to other types of wireless systems may be appropriate. For instance, more complex data exchange and
further software demands could be considered when some type of database synchronization is needed between a central server and a mobile device.

A further extension suggested is a formalized methodology for classifying, comparing and testing APIs from the developer’s point of view. Benchmarking is done in terms of strength testing, but some higher level testing would be likely to improve the state of not only Bluetooth application development, but wireless networking development in general. This would help the technologies to mature and possibly contribute to a standardization of programming interfaces. Specific research in the field of application development was fairly limited, with the only available sources of information on manufacturer’s websites or forums describing developer’s personal experiences.

5.3 AWIS APPLICATION AND EXTENSIONS

According to the field test, the AWIS application was found to work correctly and, besides the slow interaction time, functioned as expected. This successfully demonstrated a proof of concept and it is recommended that work on AWIS could continue with further investigation into possible hardware implementation.

AWIS could also be applied to other situations besides animal behavioural studies. Psychologists may be able to use AWIS when monitoring human behaviour: for example the behaviour of humans in prison situations. Tracking the movement of individuals with and between groups could be collected as well as time spent in particular areas. Despite possible ethical and privacy issues, employers could track how much work is being done by employees and how much time spent around a coffee machine for instance.

With some modification to the system, the movements of groups could be monitored and alarm sounded when any person deviates from the group, in other words moves out of range. This could be used for keeping track of tour groups, particularly for people less able to look after themselves, such as small children or the elderly.
Other wireless networking protocols such as ZigBee or Induction Wireless technology could be investigated as an implementation alternative to Bluetooth in the AWIS application. The possibility of integrating other technologies with Bluetooth could also be considered, for instance Wi-Fi which has a larger range. The device’s position could be introduced as an additional attribute to an Encounter through the use of GPS. There may be however, interference issues with radio transmission and additional power demands for GPS as well as the possibilities of integrating application development over various platforms.

5.4 SUMMARY

This project was able to compare Bluetooth application interfaces in terms of the flexibility, functionality and compatibility they provide. Both the Virtual Serial Ports and Widcomm APIs were found to support device connectivity and basic data transfer. The Widcomm API however, supported the most suitable environment for Bluetooth application development. The Virtual Serial Ports API was found to support generic serial communications but not the comprehensive Bluetooth-specific functions provided by the Widcomm API. Rapid prototyping and enabling an iterative programming methodology were also found to be enabled by the Widcomm API, allowing for an AWIS application to be successfully developed.
References


[Cole, 2004] Cole B. “In search of a common API for connected devices”.

*Embedded.com*
REFERENCES


TechOnline 2004


REFERENCES

<http://www.windowsfordevices.com/articles/AT9854948533.html> 1 November 2004


REFERENCES


REFERENCES


[Widcomm Technical Support, 2004] Widcomm Technical Support, "SPPRT00007180 : What devices that will respond - only those in immediate range, or those not in immediate range of the calling device" E-mail to Adam King. 27 September 2004.

<http://www.panda.org/about_wwf/where_we_work/arctic/polar_bear/tagging_bears.cfm>