Device Tracking on a Scattered, Bluetooth-Enabled Network

Chris Dawson

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Chris Dawson, May 2005

ABSTRACT

Bluetooth is a short-range wireless communication technology, designed as a method of cable replacement between devices. Bluetooth technology can be found in a huge number of devices, including but not limited to: mobile telephones, portable computers, cellular headsets and entertainment equipment. A design feature of the technology means that when enabled, a device can be found and questioned, revealing its unique address (as well as other information). Very often, portable applications of the technology leave Bluetooth enabled constantly as the default, allowing others to scan for these devices.

Existing research into this area has attempted to use Bluetooth technology to accurately locate a device in a room, by using multiple broadcasting stations to triangulate its position. This method has proved to be very inaccurate due to the erratic nature of any signal indicators, which are adversely affected by the surrounding environment.

A commercial project recently implemented in the Aalborg Zoo, Denmark, relates more to the scope of this thesis, using a very large scale Bluetooth infrastructure to find specific Bluetooth devices in the zoo. This project is designed to allow parents to locate their lost child.

This thesis describes methods to utilise Bluetooth technology to provide a range of location sensitive applications. The primary goal of the project is to provide a tracking implementation using a scattered Bluetooth-enabled network of broadcasting stations; this has not previously been attempted with Bluetooth technology.

CONTENTS

TABLE OF FIGURES

CHAPTER 1 – INTRODUCTION

1.1 - General Aims

The general aim of this thesis is to use Bluetooth technology to provide an accurate way of finding a device (and thus person) in any environment. The techniques will then be extended to enable tracking of devices around the environment, for example moving between rooms in an office, or between regions in an amusement park.

The short range and low power usage of Bluetooth radio technology makes it suitable for this application, as signals generally do not pass through walls, allowing devices to be located with sufficient accuracy.

The system will implement a general tracking and locating system, on which further developments can be made, enabling indoor location-sensitive computing with everyday technology. An important aim is to allow devices to query a server for its location; indoor location sensing is a highly sought after technology in computing, where GPS is unavailable.

1.2 - Specific Goals

The goals of this project are:

1. To create a Bluetooth broadcasting station, capable of enumerating devices within its range
2. To create a server with full knowledge of the network and devices in its vicinity
3. To implement a tracking strategy, to follow the movements of a device between broadcasting stations
4. To implement a method of server triggered location-sensitive computing, whereby the movements of a device can trigger actions (e.g. a door opening)
5. To extend to a self-organising network, by utilising the movements of devices to determine the positions of unknown broadcasting stations

1.3 - Usefulness

With the range of personal devices now using Bluetooth, such as PDAs and mobile phones, the possibility arises to locate and track the movements of a person. There are many uses of locating and tracking a person or object, in a variety of contexts; such as following the movements of a child around an amusement park, locating colleagues in an office or tracking the movement of luggage through an airport.

The extensions to the project, in terms of location-sensitive computing, offer a wide range of technical and consumer applications. Allowing a device to query its location allows uses such as indoor tour guides or other location-specific programming. Server controlled actions mean that the movements of a device can be monitored and used to set off certain events, such as turning on a stereo and lights when someone arrives home.

This new method of location detection and tracking will use widely available consumer equipment, making it accessible and suitable for any scale and situation.

1.4 - Structure of Thesis

The remainder of this thesis is structured as follows:

Chapter 2 details the background to this thesis, looking at previous research in the field and methods of locating devices using Bluetooth technology.

Chapter 3 discusses the design and specification of the project, examining choices that have been made and the reasons behind them.

Chapter 4 goes into the implementation of the project, with different techniques and algorithms used in key parts.

Chapter 5 provides the results of tests on the system, showing how the system performs and what it is capable of, drawing conclusions on the project's validity and possible application.

Chapter 6 discusses possible future work and extensions to this project.

CHAPTER 2 – BACKGROUND

2.1 – Location Detection

There are a host of options available for detecting the location of devices or equipment. These technologies can be paired with a network protocol to allow server-side tracking of devices. Three of the most popular technologies that work on this premise are: GPS, GSM Positioning and RFID tagging [1].

2.1.1 – GPS

GPS (Global Positioning System) is a satellite technology specifically designed to provide accurate location detection as a coordinate on the Earth's surface. GPS is the most popular and widely used method of electronic location detection, and is available in handheld units, PDAs and equipment for cars. Each satellite contains an atomic clock; the satellite continually broadcasts the time, along with some administrative information such as its movement path. The receiver works by listening to time broadcasts from satellites; the differences in timings between these satellites gives the distance to each satellite, which allows the use of trilateration to find the location of the receiver.

GPS requires no external infrastructure (such as beacons), but would still require a networking system of some sort, to communicate positions to the server for tracking. GPS is very accurate, providing an ideal method of locating and tracking people outdoors [1]. However, a GPS receiver requires a largely unobstructed view of the sky to allow clear communication with overhead satellites. This fact makes GPS unsuitable for indoor location sensing [2] and tracking, for example in an office building. It is very rare to gain a GPS position fix inside a building, unless the receiver is positioned next to a window.

2.1.2 – GSM Positioning

Mobile phones and other GSM equipment communicate with the GSM network through relay stations. The times at which signals arrive together with the angle of arrival (from at least three sources/stations), allow location detection of a device using triangulation [3]. In highly industrialised countries such as the UK, GSM stations are very commonplace, meaning that a location is always available, indoors or out.

The main problem with GSM positioning is its accuracy. Due to the power of the GSM stations and their coverage, it is hard to position a device with accuracy less than 50 metres (in urban areas) [3]. In rural areas, this accuracy can fall to 2500m, as GSM stations are more spaced out. Even the lower estimate makes this technology unsuitable for this project, as accuracy of 50 metres would likely cover the majority of an office building, rather than implementing tracking throughout a building (which is the primary focus of this project).

2.1.3 – Active RFID

Radio Frequency Identification (RFID) is a method of remotely storing and retrieving data using devices called RFID tags. An RFID tag is a small object consisting of a silicon chip and antenna [1], usually in the form of an adhesive sticker. Passive tags are most common, being used in shops and warehouses and have a normal operating range of around 30-40cm, but this can vary from 10mm to 5m. Active tags contain a battery that enable broadcasting, with a range up to 300 metres, although the normal usage is within 10 metres.

RFID is a common method of tracking, first introduced in the 1980s, although the use of active RFID is more modern. Passive RFID tracking is very common in shops and libraries, where tags are attached to products and are checked as they leave the shop by passing through receivers near the doors. Active RFID is popular in warehouses and other locations such as airports, where a larger range is needed. The benefits of the technology are an ultra-low power usage; good range and no need for line-of-sight [4]. RFID tags can also be read at remarkable speeds, making them useful for tracking movement.

While RFID tags are very cheap, small and suitable for tracking objects or people, the sensors are considerably more expensive and would require extensive configuration and software installations on detecting computers. Another downside is common objects or other radio waves can block RFID signals [4]. These reasons make RFID tracking unsuitable for device or person tracking, particularly in small scale, low budget tracking implementations.

2.2 - Location Detection using Bluetooth

The concept of Bluetooth is the replacement of cable between different devices; it is a short-range radio technology utilizing the 2.40-2.45 GHz ISM band, which is also used for wireless networking (802.11b/g). The main difference between Bluetooth and other wireless technologies is its design as a short range, low power, and ultimately low-cost standard. 802.11 (also called WiFi), is the most popular wireless technology but is not practical to fit into a product the size of a mobile phone.

Major players behind Bluetooth technology include Ericsson, IBM, Intel, Lucent, Microsoft, Motorola, Nokia, and Toshiba, although there are over 2100 companies currently developing Bluetooth products. At the time of writing, over a million Bluetooth units are sold every week, going into a huge range of consumer equipment from mobile phones and computers to stereo equipment and telephony. Bluetooth has an extremely low power usage due to its limited range requirements and is available in three versions:

- Class 1: 100 metre range
- Class 2: 10 metre range
- Class 3: Well within 10 metres (usually around 3)

Virtually all devices use class 2, with class 3 being adopted for some smaller devices such as Bluetooth telephony headsets and class 1 rarely being used for desktop USB dongles.

It is the low power usage and (consequent) low range of Bluetooth technology that makes it suitable for locating and tracking systems in an office environment. Other technologies such as wireless networking have too large a range, making locating very difficult as it would be hard to pin down a device with accuracy better than 20-30 metres. Wireless networking would only enable locating of other computers, which is not useful or the premise of this project. Low power usage also allows integration with a large range of battery-powered devices (such as mobile phones), increasing the scale and possible uses of a tracking system.

The Bluetooth Specification [5] details over fifty device types with room for more under umbrella headings such as 'uncategorised' or 'miscellaneous'. A sample of these devices is shown in the table below; some are more applicable to this project than others, for example, cellular phones rather than HIFI audio.

Printer	Laptop	Wearable Computer	Wearable Headset
Scanner	Desktop	Cellular Phone	Hands-free Device
Camera	Handheld PC	Cordless Phone	HIFI Audio
Camcorder	Video Conferencing	Gaming/Toy	Keyboard

2.3 - Approaches to Bluetooth Location Detection

The main aim of this project is to create a system that allows location detection and awareness using Bluetooth technology; tracking builds on this functionality and is dependent on it. The most important problem that must be overcome is how to find the location of a Bluetooth device.

2.3.1 - Shared client ranges

A Bluetooth radio has a spherical range around it, within which it can 'see' any other enabled Bluetooth device. This means that using the overlapping of ranges, we can roughly position a device. The more client stations we have, the higher the accuracy of positioning.

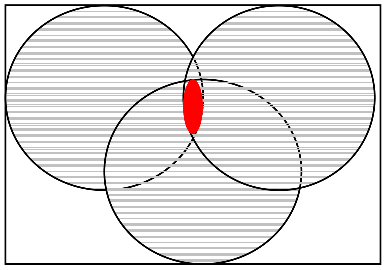


Figure 1. If three clients (grey) can see a device, it must sit within the red area.

This method works on the basis that we know the device is within our range, but the **distance-to-device** (DTD) is unknown (i.e. we know it is within <client range> metres, but that is all).

2.3.2 - Range-finding using RSSI

Received Signal Strength Indicator measurement is the first and most obvious method of measuring actual DTD. Knowing the DTD will allow the use of triangulation and more accurate positioning.

RSSI is a hardware function detailed in the Bluetooth specification [5] and provided by Bluetooth device manufacturers. The RSSI of a device can be queried and measures the strength of the signal between the client and device; this could be configured and used as a measure of DTD (assuming the signal degrades uniformly as distance increases). Several previous implementations of Bluetooth device location have used this measure to find the distance to a device.

Unfortunately RSSI suffers greatly in the presence of surrounding objects and, for example, glass or metal. Even in a clear room, the RSSI can fluctuate dramatically due to reflections; meaning a linear relationship between DTD and RSSI is impossible in all but perfect conditions. This inability to form a clear relationship between distance and RSSI means that it is unsuitable for range-finding: any solution using this method would be expensive (to compensate for fluctuations) and inaccurate. Ana Zapater et al. [6] showed that an RSSI-based distance finding algorithm is possible, but requires extensive configuration of a client in order to give meaningful results.

2.3.3 - Speculative range-finding

A further approach to device locating is by using the range overlap method in conjunction with a speculated range of the target device. The Bluetooth specification [5] details a huge list of possible Bluetooth devices, from laptops and desktops to handheld computers, mobile phones, headsets and home stereo equipment.

Each Bluetooth device must display its type, which can be queried from a remote client. Once the type has been decoded we can speculate on the maximum DTD (for example, a mobile phone is almost certain to have class 2 Bluetooth chip (maximum range of 10 metres), whereas a Bluetooth headset is more likely to use a class 3 chip (maximum range of 3 metres)).

We can find the minimum of the (speculative) distance to device and the range of the client to find the most accurate maximum distance from the client (i.e. if the device range is 10 metres but the client range is 3 metres, the device must be within 3 metres). This method, like all others, is likely to produce more accurate results when used with multiple clients. The method is not accurate to more than a few metres, but is easy to calculate, almost flawless (save for e.g. a high powered headset with a 10m range) and unaffected by environment conditions.

2.3.4 - RSSI based range refinement

Provided a client can see at least two devices, it may be possible to use RSSI information to selectively limit the maximum possible distance to a device. This method could be used as a further refinement to the speculative distance approach [2.2.3] and would improve accuracy (due to higher maximum DTD accuracy). RSSI is unaffected by the class of the device, so a class 1 (high powered) device will not necessarily have a stronger RSSI than a class 3 device in the same conditions.

The concept behind the method is that if RSSI fluctuates between the client and one device, it may fluctuate similarly between the client and another. If we monitor the RSSI of both devices and measure them relative to each other (rather than trying to map them into a linear range-finding function) we can assume that the device with the higher RSSI (stronger signal) is closer to the client.

Once we have devices in order of distance from the client, we can use information about their speculated ranges [2.2.3] to refine other device ranges. For example, if we find that the most distant device from the client has a range of 3 metres, we know that every other device known by the client must be within 3 metres (as they are closer than the most distant), so we can refine their speculative ranges.

This is an experimental method and, based on previous knowledge about RSSI-usage, its performance is unlikely to be satisfactory. This method will be implemented as a research concept and may be used if found to be beneficial.

2.4 - Technical Issues Involved

1) Communicating with a Bluetooth device and enumerating devices within range
2) Decoding of device type allowing speculative range lookup
3) Range-finding algorithms
4) Distributed client-server system allowing multiple clients working together to form a tracking/locating network
5) Triangulation algorithms on the server side to show the geographical location of a device on a map
6) 'Clever' client positioning algorithms, to infer a client's position if it has not been positioned (or its location is not initially known)

2.5 - Previous Work: Bluetooth Positioning Systems

There have been several attempts to make a Bluetooth-based positioning system, all based on RSSI measurements with different levels of complexity and configuration.

Ana Zapater et al. [6] introduced a technique for range-finding a remote device using Bluetooth as part of a networking infrastructure development, designed to allow Bluetooth devices to automatically switch to the nearest or most powerful Bluetooth AP

(Access Point) available. The range-finding method used an RSSI-based algorithm comparing RSSI readings from the device to RSSI readings from known reference devices (RD). Silke Feldmann et al. [7] continued this work as a standalone Bluetooth location project, utilising 'triangulation methods using the received signal power strength of the surrounding Bluetooth access points'. The method was found to be flawed even in the most basic situations, as it required a positive RSSI feedback, which meant the system was only reliable within 8 metres. Even inside this boundary and with a minimum of three Bluetooth access points (one listener and at least two reference points), resolution was found to be around 2.08 metres, giving more than a 25% error margin. The Institute of Communications Engineering at Hanover University continues this area of research, attempting to improve accuracy using a combination of an inertial system and an adequate Kalmann filter; results are yet to be published.

Most Bluetooth location systems use the afore mentioned techniques or a simplified version mapping an RSSI measurement directly to a distance (without using reference devices). The latter method is obviously less accurate and in the best case scenario can only give a distance from the receiver, not an actual position. A position could be approximated using this system and triangulation, provided the device is within range of at least two receivers.

'Bluetooth might be a viable technology for triangulation, but definitely not for calculating or measuring accurate distances' [8].

The first commercial application of Bluetooth locating is by a company called Bluetags, which produces small custom Bluetooth tags that can be tracked and located by its system [9]. The Aalborg Zoo in Denmark has deployed the technology as a consumer product to visitors, who can rent a tag for their child. The Bluetags system is integrated with an SMS (mobile phone messaging) gateway, enabling parents to text-message the system and enquire about the location of their child, should it become lost. The system then replies to the sender with up to date information on the child's location. Although commercial secret prevents accurate knowledge of the system, it is expected that the solution does not provide any range measurement, and instead works on a high number of short-range Bluetooth listening stations, using the range-overlap method to give parents a general area (e.g. "Your son is by the gorilla cage, map ref: E5").

"Among all the ideas floated at the Bluetooth Congress, the zoo tracker stood out as an exemplary application to showcase the wireless technology's social benefits [9]"

This project differs from previous location methods in that the emphasis is not on accurate positioning of a device in a room, but rather on knowing its general proximity to one or more clients. More than one client will allow the use of triangulation to more accurately place the device. Whereas previous systems have focussed on pinpointing the location of a device in e.g. a hall, this system will serve to locate and track devices in an office environment, where we only need to know which clients a device is in range of to position it. For example, the afore mentioned project would attempt to tell you exactly where "Nokia9210" is in Conference Room 2; this project is more concerned with telling you whether the device is in Conference Room 2 or another room (and tracking movement between rooms). This project will therefore require much less configuration, which is beneficial for implementation and deployment, as well as much less infrastructure and investment, as there will be no need for reference devices.

The project will be similar to the system produced commercially by Bluetags, but will focus more on providing graphical feedback rather than textual. It is hoped that this will provide a more user-friendly experience for the tracking system administrator. The system will also provide full graphical tracking, a feature not available to Bluetags users; as well as accuracy increasing range-finding methods.

Background research is included in this project in Appendix B, which seeks to demonstrate the prevalence of Bluetooth technology in modern consumer equipment, as evidence supporting the possibility and usefulness of a tracking system.

CHAPTER 3 – SPECIFICATION AND DESIGN

3.1 – The Bluetooth Tracking Network

The BTN (Bluetooth Tracking Network) is fundamentally made up of three entities:

CLIENT **SERVER** **DEVICE**

A device is any Bluetooth enabled equipment that is not connected to the network (i.e. not a client or a server). Most devices will be mobile and could be mobile phones, PDAs, headsets or other equipment.

A client is a broadcasting and listening station that continually searches its surrounding vicinity for devices and copies any retrieved data to the server over a permanent connection.

A server hosts connections to one or more clients and collates the information from them, processing the information and implementing the tracking system and extensions.

Extensions to the BTN introduce two further entities:

FEED LISTENER **NOMAD**

A feed listener is a different type of client that receives information on the movement of a tracked device. For example, the server may tell all feed listeners: "The server is currently tracking 'John Smith's Mobile', which is currently in the vicinity of 'Paul Smith's Office'", then as the device moves, it will update the feeds with "John Smith's Mobile has now left Paul Smith's Office", "John Smith's Mobile has now entered Tony Robinson's Office".

A nomad is a special type of device that connects to the network through a separate port on the server. A nomad can query its position, or the position of any other device on the network; the server will respond with the resolved position (e.g. "Paul Smith's Office"). This will implement the extension allowing indoor location sensitive computing (a program can know where the computer is in the building).

3.2 – Choosing a network structure

When attempting to create a tracking or listening network, there are two fundamental entities that must be considered: that which is doing the listening and broadcasting, and that which is monitoring movement through the network (thus having information of all listening stations). There is no need for each listening station to have knowledge of the network, meaning the tracking system requires a simple client-server network structure.

The client must constantly check the surrounding area for devices, updating the server with any changes (e.g. a new device has been found, or one has disappeared). The server must maintain a connection to each client and have knowledge of the whole network. Clients will be geographically separated to provide a more extensive cover of the terrain. The more clients available, the more reliable and accurate tracking is available (as device positions can be sampled more regularly).

The network will be built on standard networking technology, using TCP connections between client and server. This approach means that the network will be scalable and ready to use on a LAN or over the Internet.

The topography of the network is shown in the diagram below. The diagram shows Bluetooth radios utilised by the clients, which maintain a permanent connection with the server (for the duration of execution).

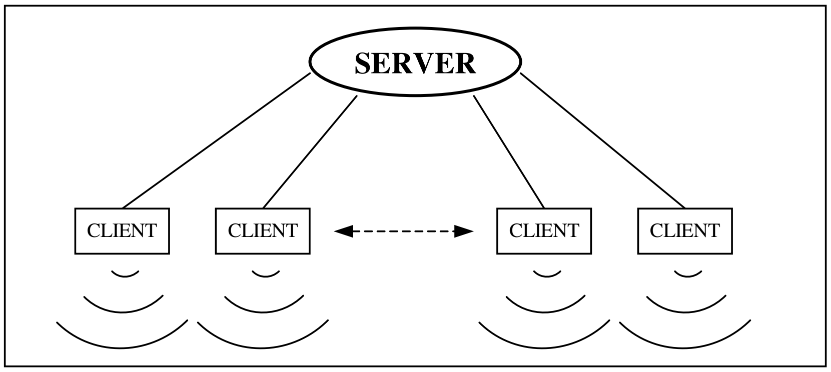


Figure 2. The proposed topography of the tracking network

3.3 – Listening Station (Client) Specification

As the Bluetooth module, the client is the most important part of the project. The client must run quietly or silently in the background, not tying up system resources (so it can be used on normal workstations). This means that a GUI or user interface is not required after the client is configured.

The client must continually 'ping' its vicinity, checking for devices and returning their addresses, names and types. The type can then be decoded into human-readable format. The client will maintain a list of previously seen devices, and will update the server with any changes to this list (e.g. a new device has entered the vicinity of this station, or a device has left).

3.3.1 – Developing the Client

The programming language of the client is restricted by the limited support in developing Bluetooth applications on different platforms. There are very few free frameworks available for developing Bluetooth applications, so the programming language and platform must be chosen to fit in with the best of these frameworks.

The main option on Linux is the open source Java Bluetooth stack Bluez. This is highly supported through forums and provides many features usually only found in commercial stacks (such as object push). The problem with this stack is that it requires a relatively new version of the kernel, either meaning a kernel update or use of a new OS release (it is bundled in Fedora Core 2). This means it would be hard to implement a client built on the Bluez stack in an office environment. Most office workstations use Windows; using Bluez would not only require Linux, but would also require the installation of the stack on each workstation and possibly a kernel update. A further limitation is the abstraction of the stack; because Bluez is available for Java, some more technical uses are abstracted away from the programmer – aspects of the technology that are only available in a lower level language such as C or C++. This fact may reduce the ability to upgrade or fine tune future releases of the client.

Until recently the only option for development on Windows was using commercial Bluetooth stacks, which are designed for business, so are prohibitively expensive. The release of Service Pack 2 for Windows XP changes this, providing native Bluetooth support with Microsoft drivers. This effectively means that Windows XP SP2 can use many Bluetooth devices without third party drivers (although usually with less features than third party drivers). XP SP2 provides the Bluetooth stack and framework, meaning that a relevant Bluetooth program can run without the need for any other software installations. Further, the Bluetooth hardware (e.g. USB dongle) will be automatically installed when it is plugged in. In order to develop Bluetooth applications to work on XP SP2, the Platform Software Development Kit (Platform SDK) is provided as an addition to the .Net software. Once this is installed and configured, programs can be written in C++ within the Microsoft .Net environment. A drawback is that support for Bluetooth development within the Platform SDK is very scarce.

Using Bluez requires stack installation, possible kernel updates and multiple Linux workstations, which are uncommon in an office environment. My personal preference is to work on Windows, which requires at most only a service pack update (far simpler) and is commonly found in an office environment. For these reasons, the client has been designed as a Windows-based application in C++, programmed with reference to the Microsoft Bluetooth stack.

3.3.2 – Technical Client Design

There are two distinct components of the client, the Bluetooth broadcaster/listener and the network interface to talk to the server. The client is designed to be multi-threaded to allow the two components to be kept independent. One thread will handle messages from the server (which implements blocking IO), while another will implement the broadcaster and send messages to the server.

The broadcasting thread will enter a permanent loop in which it will check the surrounding area for devices, compare this to the known list (of devices it saw last time) and update the server with changes before updating the list. Before updating the server, the client retrieves the device type bit pattern and decodes it using the Bluetooth specification. The resolved device type is provided to the server (in human-readable format e.g. “Cellular Phone”). The client also sends the server a prediction of the maximum distance that the device is at. The prediction is made using two refinements discussed in the background, and detailed in chapter 4.

The messaging thread will also enter a permanent loop in which it checks for incoming messages and then processes them. The client will only ever receive two types of message from the server: a RUN or KILL command. The RUN command will instruct the client to run a program or open a file on the client computer, this could be an executable file that turn on a light (via other hardware) or a music play list. The KILL command could be used to destroy a process or stop a client plug-in.

3.3.3 – Client Interface

The client requires no GUI or user interface once it has been configured. The program has been designed as very slim line, using very little processing power or memory. This means that it is not unreasonable to break out of the program to exit (since it is designed to run permanently).

The client will be a console application that takes several arguments on the command line:

1. The server name or address
2. The port on the server on which the server application is running
3. The range of this station

The station range can be determined by two factors. In a completely open area, the range is limited only by the hardware. Class 1 devices have a maximum range of 100m, while Class 2 devices have a maximum range of 10m and Class 3 devices have a maximum range of around 3m. In an office environment, the limiting factor is often the size of the room (Bluetooth signals cannot pass through walls), so the range could be 2m for a small room.

The program will identify the Bluetooth adapter and ask the user for the name of the surrounding area (e.g. ‘Paul’s Office’), this name will be passed to the server for ease of use and human understanding.

3.4 – Server Specification

The server is the part of the project that implements the network features of location and tracking, as well as the more advanced extensions to the system.

Due to the nature of Bluetooth, there are very few things that the client can actually do or retrieve. The most useful data is the address of the device, how far away it is and its name. The key in developing this project is the processing and manipulation of the data on the server.

3.4.1 – Developing the Server

The server has no restrictions on language or platform, as with the client, so the choice of language and platform can be based purely on the requirements.

The only requirements of the server that may affect the choice of language are:

1. To provide a graphical interface, allowing both pictures, controls and simple shape drawing
2. To implement simple network communications (acting as the server)

The obvious language to meet these requirements is Java, which provides all required graphical functionality as well as libraries supporting network communication. Another benefit of Java is platform independence, meaning the server could be run on any Java-enabled system.

3.4.2 – Technical Server Design

Unlike the client, the server is responsible for many more technical tasks and algorithms, so is made up of several key modules, comprising the main system and the extensions.

The server modules are:

1. The network interface
2. The data processing module
3. A mapping system
4. A database of network device names
5. A database of location sensitive actions
6. A live feed module
7. A location request module
8. A route finding module
9. A client position inference module

The network interface must implement communications with connected clients. This interface will be multi-threaded, with each client assigned to a thread. The server can then run all clients in parallel with automation. Clients will connect to a single port on the server, before being moved to another free port for permanent connection. The network interface must periodically check each connection for incoming messages, then process them and hand them up to the main program.

The data processing module must take all data from the connections and introduce them to the user in an organised and relevant manner. Information from clients may contain data about new devices, devices that have left the network or device name updates (if the device's name has changed). This information must be organised so that the user can see if a device is in range of one or more stations, or watch its movements between them, this requires the collation of the incoming data.

The mapping system must be capable of showing a map of the network terrain. This will most likely be a floor plan of the building, with separate levels shown on different maps. The mapping module must be capable to drawing clients and paths over the map, overlaying it with the required information. Drawn clients will have a fixed location and a surrounding range; device paths (found by tracking a device) will be plotted between these clients. In order to correctly place the clients and draw ranges accurately, the scales of the maps will be required. The scale will be written in a text file, specifying the number of pixels corresponding to one metre on the map. As well as the map picture, a secondary, highly simplified terrain image will be produced for each map. This image will consist of only red, blue and green areas for use in the route-planning module. The terrain and route planner will be discussed in detail in the implementation chapter.

The device name database will store names by which the system can reference devices, to improve user interaction and knowledge of the network. The saved device name will take priority over the device's actual name (resolved by the client). This will allow the network users to view devices as, for example, "John Smith's Phone" rather than "Nokia6310".

The location sensitive actions database stores actions or commands that the server can trigger after specified events. An event could be a device entering the vicinity of a station (client), or leaving the vicinity of a station. If the event corresponds to a saved event in the database, which is made up of a station address and device address, the relevant action will be triggered. An action is an identifier, RUN or KILL, with a command. The action is sent to the relevant client (specified in the database), and is performed by that client. The RUN identifier causes the client to run the executable file or open a file (e.g. a play list) specified in the command (e.g. "RUN C:\playlist.m3u") using the client's default programs (e.g. the play list may load in Winamp). The KILL identifier will either destroy a process (turning off a play list), or as a project extension, could be used to stop a client plug-in that has control over external hardware (e.g. controlling doors or lighting).

The live feed module provides information to the previously described feed listeners. The module runs on a separate port, providing movement updates on a tracked device to any connected viewer (for example, allowing someone with a PDA to track down the device). The server automatically prints out information on the tracked device onto a server window; this information is copied to any connection feed listeners.

The route finding module has two purposes. The first purpose is to provide realistic paths through the map, when a device has moved in an unknown course from one client to another. The route finder will plot a path through the terrain, avoiding walls and obstacles instead of just plotting an unrealistic birds-eye route. The second purpose of the route finder is to enable inference of client positions, which is a separate module. The route finder uses the A* algorithm and assumes the movements of the device are those of a person who knows the building and takes the shortest route from a to b. A complex path is more accurate if it is sampled regularly, i.e. the higher the number of clients encountered, the better the path approximation. The route finder avoids walls and obstacles by using a node map, which is a highly simplified terrain image of the map. The specifics of the route finder are explained in the implementation chapter.

The client position inference module is an extension that improves the network by making it a learning network that can use data from clients and device paths to estimate the position of an unknown client. The server administrator must usually know the positions of clients, which can be placed on the map upon connection. If a new client connects, with an unknown location, the system can infer its position by analysing the paths that pass through it. Paths are found using the route finding module described previously. For example, if a device moves from A to C encountering B along the way, we can make a rough estimation of the position of B. If further devices move from D to E encountering B along the way, the estimated position of B is improved. The details of the inference module are given in the implementation chapter.

3.4.3 – Server Interface

The server application must provide a graphical interface to the user, as there is a large amount of information to display.

The Java application takes one command line argument specifying the port (portnumber) on which it should listen for client connections. The Java application will then open up another port at portnumber+1000 to handle feed listeners, and a third port at portnumber+2000 to handle nomads (location requesters).

Upon running, the server will display the main graphical interface to the user. This interface will provide information on clients, devices on the network and graphical information overlaid on a terrain map.

The device and location actions databases will be browsed through separate windows, reachable from the main panel.

Other features that must be provided by the graphical interface are:

1. The possibility to view information about a device, namely its address, name, type, hosts (clients it is in range of) and maximum likely range from the client(s)
2. A window to print feedback to the user, be they error messages, contextual help or information on the movements of a device
3. Controls to allow access to more advanced features of the BTN

When the server window is closed, all clients will be disconnected and the server application will terminate.

3.5 – Protocol Specification

The last specification needed is that of the communication protocol between different clients and the server. The communication will be a high level protocol built on top of TCP/IP, so there is no need for the protocol to provide acknowledgements or other lower level protocol features.

Client/Server communication has been designed to be very simple, as there is only limited information to send, and to improve programmer understanding. The simple communication allows easy implementation of new clients on different systems. It would be relatively easy to create a Linux compatible client (using Bluez), and enable it to communicate with the network.

A message passed between client and server is made up of an identifier, followed by one or more arguments:

Identifier	**Meaning**	**Argument 1**	**Argument 2**	**Argument 3**	**Argument 4**
Client to Server					
STP	Client Set-up	Client name	Client BT address	Client Range	
IN	New device found	Device name	Device address	Device class	Device range
OUT	Device left the area	Device address			
UPD	Device name change	Device name	Device address		
Server to Client					
RUN	Execute command	Command			
KILL	Destroy process	Process name			
Server to Feed					
	Tracking update	Message			
Nomad to Server					
WAI	Where Am I?	Device address			
Server to Nomad					
LOC	Location Reply	The resolved position			

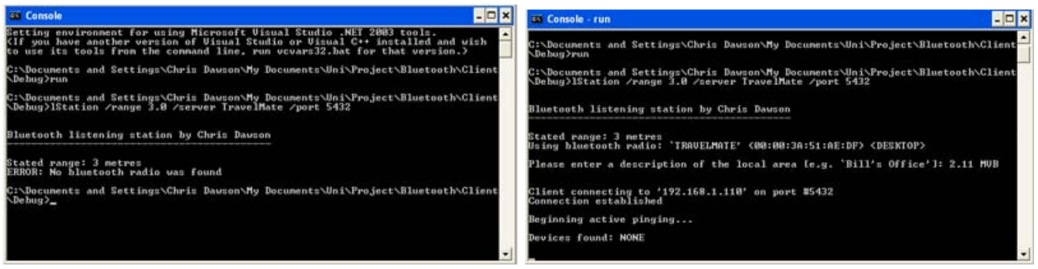
CHAPTER 4 – IMPLEMENTATION

4.1 – Client Implementation

The client was written in C++ within the Microsoft .Net environment. As described in the specification, the client is made up of two modules, the network interface and the Bluetooth operator.

4.1.1 – Client Interface

As described in the specification, the client takes three arguments as input: the server hostname or address, the server port and the range of the server (in metres). If the range is not input on the command line, the program will prompt the user for it. The server will attempt to detect a Bluetooth adapter, and, if successful, prompt the user to enter a name for the surrounding area.



```
Setting environment for using Microsoft Visual Studio .NET 2003 tools.
(If you have another version of Visual Studio or Visual C++ installed and wish
to use its tools from the command line, run vcvars32.bat for that version.)

C:\Documents and Settings\Chris Dawson\My Documents\Uni\Project\Bluetooth\Client
\Debug>run

C:\Documents and Settings\Chris Dawson\My Documents\Uni\Project\Bluetooth\Client
\Debug>lStation /range 3.0 /server TravelMate /port 5432

Bluetooth listening station by Chris Dawson
-------------------------------------------

Stated range: 3 metres
ERROR: No bluetooth radio was found

C:\Documents and Settings\Chris Dawson\My Documents\Uni\Project\Bluetooth\Client
\Debug>_
```

```
C:\Documents and Settings\Chris Dawson\My Documents\Uni\Project\Bluetooth\Client
\Debug>run

C:\Documents and Settings\Chris Dawson\My Documents\Uni\Project\Bluetooth\Client
\Debug>lStation /range 3.0 /server TravelMate /port 5432

Bluetooth listening station by Chris Dawson
-------------------------------------------

Stated range: 3 metres
Using bluetooth radio: `TRAVELMATE' (00:00:3A:51:AE:DF) (DESKTOP)

Please enter a description of the local area [e.g. `Bill's Office']: 2.11 MVB

Client connecting to '192.168.1.110' on port #5432
Connection established

Beginning active pinging...

Devices found: NONE
```

Figure 3. The client interface with and without a Bluetooth adapter

The application then attempts to connect to the named server, on the specified port. If successful, the client will then send a set-up message to inform the server of its details, before beginning 'active pinging', i.e. searching for Bluetooth devices. At the end of every search, any changes are processed and sent to the server.

4.1.2 – Client Network Interface

The network interface connects a single port to the server, through which all communication passes. The receive method implements blocking IO, which requires a separate thread, as it would otherwise pause execution.

The code to create a separate thread for reading incoming messages is simple:

```
CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) checkMessages, NULL, 0, NULL);
```

Message sending and receiving are operations on a socket. When reading a received message, we first read it into a buffer and return the size of the message:

```
if (sock == INVALID_SOCKET){
        cout << "ERROR: No connection with server" << endl;
        exit(1);
}
memset(buffer, 0, BUFFER_LEN);
int retval = recv(sock, buffer, BUFFER_LEN, 0);
if (retval == SOCKET_ERROR) return 0;
return strlen(buffer);
```

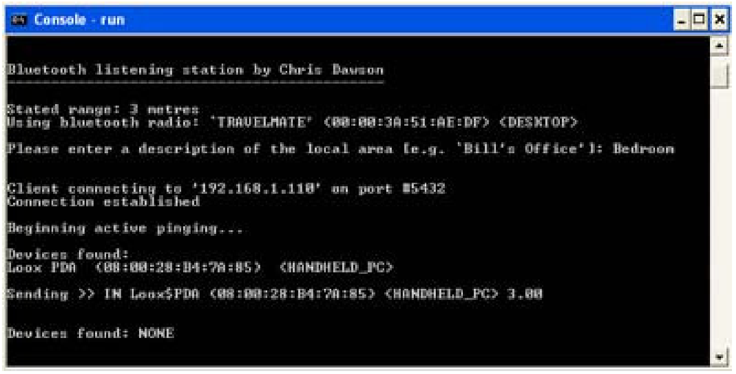
If the size of the message is valid, the contents are read from the buffer.

Sending a message to the server is similar, but does not ‘block’. Messages are therefore sent in the main execution thread, after any changes have been processed.

4.1.3 – Client Bluetooth

The Bluetooth module is responsible for enumerating remote Bluetooth devices and returning their details. Included in this is a decoder that interprets the device type bit-pattern in accordance with the Bluetooth specification, to return the device type name.

The Bluetooth code references the Bluetooth header files included in the Microsoft Platform SDK.



```
Bluetooth listening station by Chris Dawson
-------------------------------------------

Stated range: 3 metres
Using bluetooth radio: 'TRAVELMATE' <00:00:3A:51:AE:DF> <DESKTOP>

Please enter a description of the local area [e.g. 'Bill's Office']: Bedroom


Client connecting to '192.168.1.110' on port #5432
Connection established

Beginning active pinging...

Devices found:
Loox PDA  <08:00:28:B4:7A:85>  <HANDHELD_PC>

Sending >> IN Loox$PDA <08:00:28:B4:7A:85> <HANDHELD_PC> 3.00


Devices found: NONE
```

Figure 4. Client ‘pinging’ for devices

A sample piece of code is shown below, this code is part of that responsible for finding devices. This snippet reads from a set of buffered devices (that have just been found), then reads the device address and pushes the results (containing the name and device type) onto the stack.

```
WSAQUERYSET *pResults = (WSAQUERYSET *)&buffer;

memset(pResults, 1, sizeof(pResults));
result = WSALookupServiceNext(hLookup, flags, &bufferLen, pResults);

if (result == 0)
{
        CSADDR_INFO *pCSAddr = (CSADDR_INFO *)pResults -> lpcsaBuffer;
        char addrStr[48];
        DWORD addrSize = sizeof(addrStr);

        WSAAddressToString(pCSAddr -> RemoteAddr.lpSockaddr,
        pCSAddr -> RemoteAddr.iSockaddrLength, &protoInfo,
                addrStr, &addrSize);

        addDevice(pResults, addrStr);
}
```

The next task of the Bluetooth module is to decode the device type bit-pattern. There are over fifty device codes covered in the decoder. Different parts of the program need to access this component with different information, hence the function overloading:

```
char *GetDeviceClass(LPGUID devClass)
{
        ULONG bits = devClass -> Data1;
        return (GetDeviceClass(bits));
}

char *GetDeviceClass(ULONG bits)
{
    if (bits & (1 << 12)) return ("<UNCATEGORISED>");
    else if (bits & (1 << 10))
    {
        if (bits & (1 << 9))
        {
            if ((bits & (1 << 7)) && (bits & (1 << 6))) return ("<PRINTER/SCANNER>");
```

When searching, the client pushes each device onto a stack of new devices. Once all devices have been found, the stack of new devices is compared to the known stack (which was populated on the last search). If there is a new device, the client will send an ‘IN’ message to the server. If a device has not been seen for two searches, it is assumed to have left the vicinity of the client, so an ‘OUT’ message is sent.

Sometimes, the client is unable to retrieve the name of the device on the first search. In this instance, the name is set to ‘-Resolving-’. If the name of the device is found on subsequent searches, the name will be updated and the client will send an ‘UPD’ message.

It is possible to change the period of the Bluetooth search, which normally takes around six seconds. However, it was found that changing the timeout (to two or three seconds) reduced accuracy, as the radio had less time to retrieve details on devices. Six seconds was found to be accurate as well as sufficient for detecting those walking past the station (at normal walking speed), so the timeout value was set to the default.

4.1.4 – Client Range Refinement

As discussed in the background chapter, this project has attempted to use speculative range refinement combined with RSSI readings to find the maximum likely distance that a device is from the client. Once the device type is known, we can use it to guess at its maximum range, as discussed previously:

```
int GetDeviceRange(char *type)
{
        if (!strcmp(type, "<UNCATEGORISED>")) return 100;
        …
        else if (!strcmp(type, "<WEARABLE_HEADSET>")) return 3;
        else if (!strcmp(type, "<CORDLESS_PHONE>")) return 100;
        return 10;
}
```

If the device range is less than the client's range, we can reduce the estimate of it's maximum distance away. This measure can later be used to triangulate its position.

The client ranges are further refined using their RSSI readings to measure their distance relative to each other. The search function is set to attempt to return devices in order of distance from the client, using manufacturer specific methods that include a measure of the RSSI. By then iterating through the stack of devices, the possible distances can be refined. For example, if a device with a range of three metres is further away than a device with a range of ten metres, we know that the latter must also be within three metres.

4.2 – Server Implementation

The implementation of the server is documented according to the modules discussed in the specification. The server is programmed in Java.

4.2.1 – The Network Interface

The server network interface is implemented using a high level protocol described in the specification, and uses a multi-threaded design to provide communication with multiple clients.

When a client connects, it is moved onto another port in a ServerThread so that other clients can connect. The connection (ServerThread) is then added to a list in the main program so that it can be periodically checked (on a timer) for incoming messages. The ServerThread type encompasses the connection and all methods allowed on it, including reads and writes, as well as other administrative methods. The ServerThread type utilises the network protocol that decodes incoming device messages.

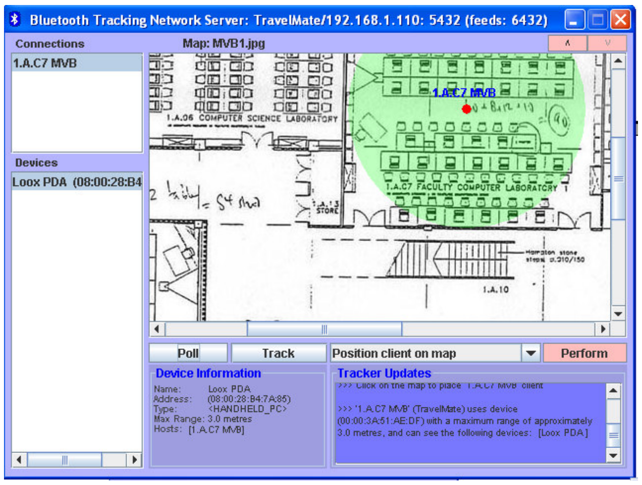
The network code is duplicated in other parts of the program to provide server capabilities to other connections, such as for the feeds and the nomads.

4.2.2 – The Interface and Mapping System

The main program interface provides information on all clients and devices on the network. Also in the interface is a layout of the building or area as a floor plan.

The system uses the floor plan as a convenient and easily understandable method of visualisation. The system draws over the floor plan to provide extra information such as the location of clients and the paths of devices that move between them.

In the screenshot is shown a map of floor 1 of the Merchant Venturers Building. The map is both manually and automatically scrollable; the program will automatically scroll to locate a device or follow the movements of a tracked device. Multiple maps (e.g. floors) can be loaded into the program, allowing a complete building to be covered with a single server. A client has been positioned on the map; it appears as a red circle with a green area around it representing its range.



Bluetooth Tracking Network Server: TravelMate/192.168.1.110: 5432 (feeds: 6432)

Connections: 1.A.C7 MVB

Devices: Loox PDA (08:00:28:B4

Map: MVB1.jpg

1.A.06 COMPUTER SCIENCE LABORATORY
1.A.C7 MVB
1.A.C7 FACULTY COMPUTER LABORATORY
1.A.13 STORE
1.A.10
Poll
Track
Position client on map
Perform

Device Information

```
Name:       Loox PDA
Address:    (08:00:28:B4:7A:85)
Type:       <HANDHELD_PC>
Max Range: 3.0 metres
Hosts:  [1.A.C7 MVB]
```

Tracker Updates

```
>>> Click on the map to place '1.A.C7 MVB' client

>>> '1.A.C7 MVB' (TravelMate) uses device
(00:00:3A:51:AE:DF) with a maximum range of approximately
3.0 metres, and can see the following devices:  [Loox PDA]
```

Figure 5. The main user interface of the server

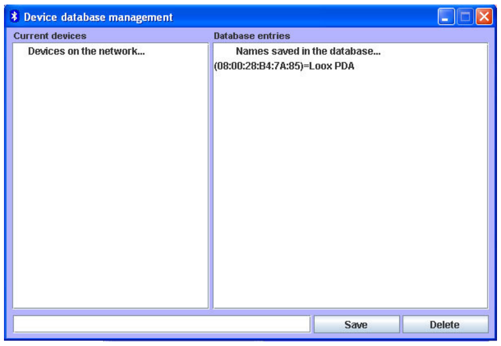
The main program interface communicates a variety of information that has been collected from the clients and processed. On the bottom right of the screen is a tracker update and status box, used to communicate extra communication to the user.

On the left hand side are shown two panes that list all connections and devices. A connection can be 'polled', which will print in the status box the name of the connection, its network address or hostname, the address of the Bluetooth device is uses and a list of the devices it can see. Selecting a device in the device list refreshes the Device Information box (bottom centre) to show the details of the device. The details of the device cover its name, Bluetooth address, device type, maximum range (calculated by clients) and a list of the hosts (clients) it is in range of.

Other commands that are available include toggle device tracking, position a client on the map, locate device, browse device database, browse location sensitive actions and infer client position.

4.2.3 – The Network Name Database

The database is stored in a text file in the form shown in the pane on the right. An address is 'tied' to a user-selected name. All saved names are loaded into a hash table at run time, so that the system can easily check for a name in the database (using the address as a key) and then retrieve the name. The system will look up the name from the database before using that retrieved by the client(s).



Device database management

Current devices: Devices on the network...

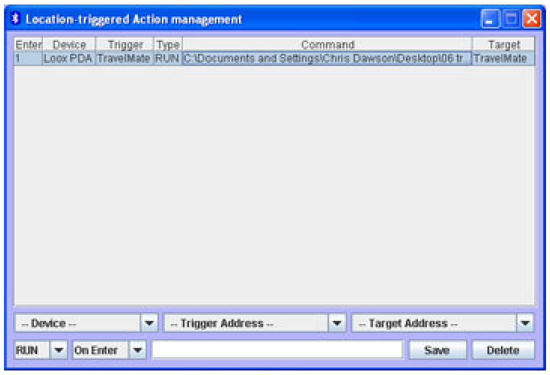
Database entries: Names saved in the database...

```
(08:00:28:B4:7A:85)=Loox PDA
```

Figure 6. The device name database browser shows devices on the system and saved names

4.2.4 – The Location-Sensitive Actions

Shown below is an image of the Location-Sensitive action browser. The table shows the actions, which are stored in a hash table after being loaded from a dollar-separated ($) text file. The key in the hash table is a composite of the 'On Enter/Leave', device address and trigger client; i.e. these parameters make an action unique. An action can be added using the controls in the form, where drop down boxes are populated by the current state of the system (i.e. all devices currently on the system are listed).



Enter	Device	Trigger	Type	Command	Target
1	Loox PDA	TravelMate	RUN	C:\Documents and Settings\Chris Dawson\Desktop\06 tr...	TravelMate

Figure 7. The Location-Sensitive Action browser

When a device enters or leaves the vicinity of a client, the client sends a message to the server notifying it of this change. When processing these changes, the server checks whether the conditions of the change meet the conditions of any location sensitive action in the database. For example, if address (08:00:28:B4:7A:85) enters the vicinity of the client with network hostname 'TravelMate', the server will look in the location actions hash table for '1(08:00:28:B4:7A:85)TravelMate'. If a device leaves the vicinity, the look up term would change to '-1(08:00:28:B4:7A:85)TravelMate'.

In the previous example, the address corresponds to the 'Loox PDA' device. The name is shown in the table for user purposes, the database actually stores the address and looks up the name from the device name database. If the device enters the vicinity of "TravelMate", the server will find a corresponding entry in the database. The server will then look for a client matching that referred to in the 'Target' field, if it finds it, it will send the command to that client in the form e.g. 'RUN C:\Documents and…'.

In the code snippet below, Change.ADD corresponds to an 'On Enter' event, while Change.SUB corresponds to an 'On Leave' event. The code retrieves the command for the event (if there is one) and sends it to the relevant client.

```
if ((c.addition == Change.ADD && locAct.contains(Change.ADD + deviceAddress + clientAddress))
        || (c.addition == Change.SUB && locAct.contains(Change.SUB + deviceAddress + clientAddress))){
                String msg = locAct.getCmd(c.addition+deviceAddress+clientAddress);
                StringTokenizer st = new StringTokenizer(msg,"$");
                String message = st.nextToken() + " " + st.nextToken();
                String target = st.nextToken();

                if (target.compareTo(client.getNetName()) == 0)    client.send(message);
                else{
                        for (int i = 0; i < connections.getSize(); i++)          {
                                ServerThread targ = (ServerThread)connections.get(i);
                                if(targ.getNetName().equals(target)){
                                        targ.send(message);
                                        break;
                                }
                        }
                }
```

4.2.5 – The Live Feed Module

The live feed module uses similar code to the network interface; it is a special group of clients, with which the server has one-way only traffic (messages are sent but not received). As the server picks up changes in a tracked device, it automatically scrolls the map to the location of the device and prints updates in the status box. Examples of such updates are: 'Tracker placed on Loox PDA (08:00:28:B4:7A:85), Loox PDA is currently in the vicinity of 'Corridor 2E', '(08:00:28:B4:7A:85) entered the vicinity of 'James Smith's Office', '(08:00:28:B4:7A:85) left the vicinity of 'Corridor 2E'.

The status box shows a wide variety of information to the user, not limited to just tracker updates. For this reason, the method to print in the status box also takes a Boolean specifying whether the text should also be copied to feeds (if it is a tracker update). If the Boolean is set, the text is copied to the FeedHandler, which writes to all feed connections.

4.2.6 – The Location Request Module (Nomads)

The location request module takes the form of a handler, similar to the feed module, and a group of children connections. The nomad port accepts connections and places them in a 'nomad' class. Nomad connections are two-way, with a very simple protocol consisting of an incoming request and outgoing answer. When a nomad connection requests its location, the Bluetooth address is passed up to the LOCRequestHandler.

The LOCRequestHandler is polled periodically with the server timer (upon which all connections are polled). Any incoming location requests are queued by the Request Handler and passed to the main program when it is polled. The main program looks in the device list for the given address and returns its list of hosts (the clients that it is in range of). This host list is then sent to the nomad that made the request.

```
private void serveRequests() {
        boolean known = false;

        while (locReq.hasRequest()) {
                        String addr = locReq.address;

                        for (int i = 0; i < devices.getSize(); i++) {
                                SharedDevice d = (SharedDevice)devices.get(i);
                                if (addr.compareTo(d.address) == 0)
                                        locReq.sendLocation(addr, d.hosts.toString());
                                known = true;
                        }
                        if (!known) locReq.sendLocation(addr, "Unknown");
                }
        }
```

The timer calls the method above periodically. The method serves all requests from the location request handler. Any request address is read, and then looked up in the network. If the device is found, the list of its hosts is sent back, otherwise 'Unknown' is sent.

4.2.7 – The Route Finding Module

The implementation of the A* Algorithm is contained in four classes within the ‘routefinder’ package. The interface to the program is provided through the ‘pathfinder’ class, which has a terrain and an algorithm class that references the terrain. The A* implementation is accessed from the mapping system, to plot paths of a device from one client to another. To find a path, the mapping class simply calls the pathfinder with a start position and an end position.

The terrain is a tiny (factored down), highly simplified representation of the building in three colours. Impassable areas are shown in blue, passable areas are shown in red and doors are shown in green. The image is much smaller than the high-resolution floor plan, with the sample below being 145x92 pixels.

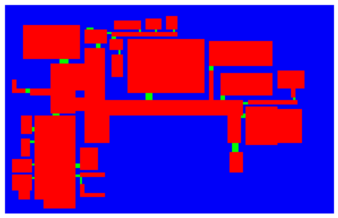


Figure 8. A terrain map of MVB Floor 2

The map is read into the program and a node is created for each pixel; the colour value is used to set the difficulty of passing the node (a feature of the A* algorithm). Blue pixels are impossible to pass (difficulty 9), red pixels are simple to pass (difficulty 1) and green pixels are hard to pass (difficulty 3). Doorways are differentiated from rooms, as a person is generally more likely to avoid doors on a route if possible (due to congestion). Doorways are also differentiated because it is assumed that Bluetooth signals cannot pass through them, so they are coloured differently to allow the program to better locate a device.

The A* algorithm plots the shortest route from one node to another, by passing through other nodes on the way. In this implementation, a node is a pixel, so the resultant route approximates a path through the terrain avoiding walls and obstacles. The shortest path method approximates the unrestricted movements of someone know knows the layout of the building. The shorter the path, the more accurate it is (i.e. the less the possibility of error), so the more clients the better, as more clients will mean more, shorter paths.

When the shortest route is known, it is scaled up to the correct measurements, to match the size of the original map. The algorithm returns a list of points describing the shortest path; this is very costly to plot, as a line must be drawn between every point, of which there may be 1000. The solution to this is to sample the returned path, with a factor of, for example, three, which doesn’t affect the accuracy noticeably but greatly speeds up computation and rendering.

The map implementation described previously keeps a list of paths of the tracked device. The sampled path is added to this list, which is rendered on the map showing the movements of the device.

```
public ArrayList getPath(Point start, Point end)
{
        start = adjustDown(start);
        end = adjustDown(end);

        goal.x = end.x;
        goal.y = end.y;

        Node previousNode = world.getNodeAt(start.x, start.y);
        previousNode.setG(0);
        previousNode.setH(goal.x, goal.y);
        previousNode.calcF();
        openNodes.add(previousNode);

        while(!(previousNode.getXPosition() == end.x && previousNode.getYPosition() == end.y)
                                && !openNodes.isEmpty())
        {
                int indexSmallestH = this.getSmallestH();          // find the openNode with the smallest h

                this.toClosedNodes(indexSmallestH);                // we've evaluated it: move to closed

                previousNode = (Node)closedNodes.get(closedNodes.size()-1);
                this.addNeighbours(previousNode);       // add walkable neighbours to openNodes

        }
        return getSolution();
}
```

The method shown above is responsible for calculating the path from the start point (node) to the end. The method calculates the ‘cost’ of traversing from one node to each of its neighbours. Nodes are checked according to a cost that is a sum of the cost from the start to the node and the predicted cost to the end point. The latter is calculated using a Manhattan distance heuristic. Once all needed nodes have been evaluated, the shortest path is calculated using the ‘getSolution’ method, which starts at the last node and traverses back using the ‘cheapest’ route.

4.2.8 – The Client Position Inference Module

The position inference module is contained in the InferredPosition package. The module is accessed from the mapping module and controls the position of a client whose position we wish to infer. When a device path is added to the map, the program checks whether it encountered the inferred client in the path, i.e. the path may go from A to C, but will only be counted as an inferred path if it encountered B along the way. If the path did encounter the inferred client, it can be used to position it.

The non-sampled path is used for the inference module, as we must have knowledge of every point in the path in order to accurately position the client. The path is sent to the inference class as an array of points. All paths are projected onto an activity matrix, the size of which is equal to the map. Paths are overlaid on top of each other, with multiple paths increasing the ‘activity’ or value of the points it covers. Only unique paths are added to the activity matrix, so that a large volume of traffic between two stations does not skew the results.

The inference class calculates the most likely position of the client after every new path is added, and returns it to the mapping module. The inferred client is then repositioned at this point. The activity matrix works in a similar way to the picture shown below.

The weights of each grid point are modified by the paths that pass through them; these modifications are cumulative.

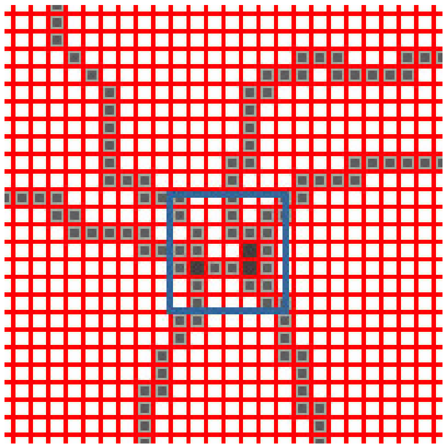


Figure 9. A representation of the activity matrix

When a new path has been added to the matrix, the weights corresponding to the points are modified, as shown in the diagram above which shows at least three unique paths. A mask is then passed over the entire matrix to find the area with the largest activity. The mask size is equal to the range of the client to be inferred, represented in the diagram above with a blue box. The area within the blue box is the most active patch in the activity matrix; the most likely position of the client is in the centre of this patch.

```
int len = infPaths.size();

Point[] temp = new Point[2];
temp[0] = path[0];
temp[1] = path[path.length-1];

for (int i = 0; i < len; i++) {        // We don't want two of the same paths - could increase possiblity of
error
            Point[] stored = (Point[])(infPaths.get(i));
            if ((stored.length > 2 && stored.equals(path)) || (stored.length == 2 && stored.equals(temp)))
                        return null;
}

if (len == 0) infPaths.add(path);
else          infPaths.add(temp);                    // Only store start & end points (all we need)

len = path.length;
for (int i = 0; i < len; i++)     // Modify the activity matrix
{
            activity[path[i].x][path[i].y]++;
}

return getPosition();
```

The code above adds a path to the activity matrix and returns the predicted position of the client, or null if the position has not changed because the path had already been recorded. The ‘getPosition’ method shown below, calculates the position of the client.

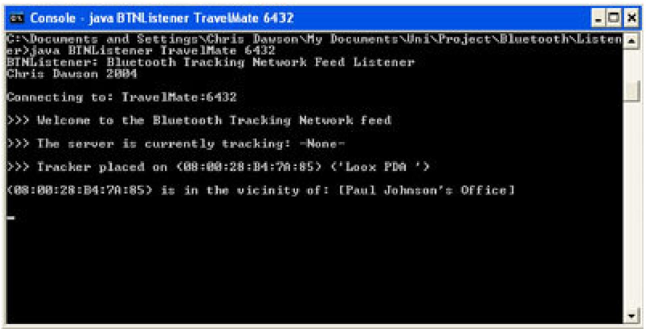
```
int max = 0;
int patchvalue;

for (int i = range; i < activity.length - range; i=i+range/10)    {
        for (int j = range; j < activity[0].length - range; j=j+range/10)    {
                patchvalue = 0;                // Then calculate the 'activity' in the patch
                                                    // (the number of paths that have been through it)

                for (int patchi = i; patchi < i+range; patchi++)   {
                        for (int patchj = j; patchj < j+range; patchj++)    {
                                patchvalue += activity[patchi][patchj];
                        }
                 }

                if (patchvalue > max)   {
                        max = patchvalue;
                        position = new Point(i, j);
                }
        }
}
```

This code fragment is responsible for moving the 'mask' over the activity matrix and calculating the value of each patch. The outermost two loops move the mask over the matrix, while the inner loops calculate the value of the patch. After the loops, the point 'position' provides the top left corner of the patch that contains the maximum activity. Moving this patch down and right by range/2 gives the centre of the patch, thus the predicted position of the inferred client.

If there is only one recorded path, the activity matrix will give an unpredictable result. Although there is no way to know where the client is positioned with a single path, the average position is used, i.e. the point halfway between the start and end points on the path.

4.3 – Listener (Feed) Implementation

The listener is a Java implementation of a feed client, which simply prints out all information that it receives from the server. A single port is opened to connect to the server and port number specified on the command line. The server sends messages relating to and explaining the movements of the tracked device.

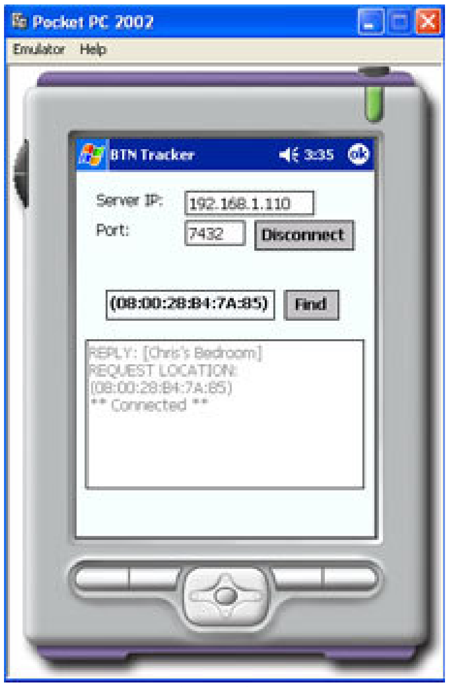


```
C:\Documents and Settings\Chris Dawson\My Documents\Uni\Project\Bluetooth\Listen
er>java BTNListener TravelMate 6432
BTNListener: Bluetooth Tracking Network Feed Listener
Chris Dawson 2004

Connecting to: TravelMate:6432

>>> Welcome to the Bluetooth Tracking Network feed

>>> The server is currently tracking: -None-

>>> Tracker placed on <00:00:28:B4:7A:85> ('Loox PDA ')

<00:00:28:B4:7A:85> is in the vicinity of: [Paul Johnson's Office]

_
```

Figure 10. A feed listener prints out movement updates on a tracked device

4.4 – Tracker (Nomad) Implementation

The nomad implementation was created as a Pocket PC project within the Microsoft .Net environment. This program can be installed on a Pocket PC and allows the user to query the position of the Bluetooth device. This could be automated, with the address hard-coded to allow a program to find its position using the server. The target application requires a network connection to the server (e.g. 802.11) and, if it's finding its own position, a Bluetooth adapter.



192.168.1.110
7432
(08:00:28:B4:7A:85)

```
REPLY: [Chris's Bedroom]
REQUEST LOCATION:
(08:00:28:B4:7A:85)
** Connected **
```

Figure 11. Tracker (nomad) running on a Pocket PC Emulator

The server finds the position of the specified Bluetooth address and returns its position as a list of clients that it is in range of, e.g. "[Chris's Bedroom, The Hallway]".

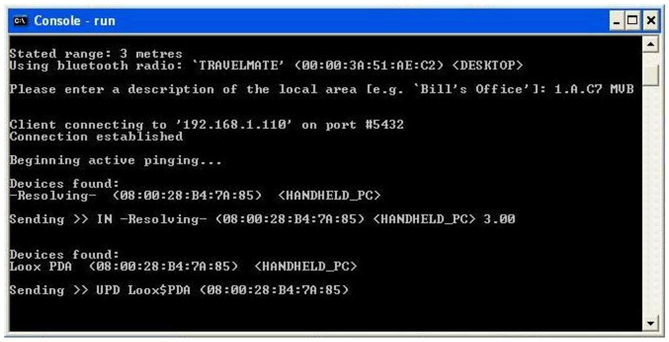
CHAPTER 5 – RESULTS

5.1 – Client Results

5.1.1 – Device Name Retrieval

When implementing the client, it became apparent that the accuracy of the device searching was not perfect. In some circumstances, the name of the Bluetooth device is not returned in the result set, which is obtained from a search. Although this is not a critical feature, as only the address is needed for the system, it helps the user, as a name is far more memorable than a hex address.

When a client cannot retrieve the name of a device, it sends the name '-Resolving-', to notify the user that its name is unknown. In most cases, if the device is still present on the next search, the name will be found. In this instance, the client will send an update message, informing the server of the actual name of the device. An example of this is shown below.



```
Stated range: 3 metres
Using bluetooth radio: `TRAVELMATE' <00:00:3A:51:AE:C2> <DESKTOP>

Please enter a description of the local area [e.g. `Bill's Office']: 1.A.C7 MVB

Client connecting to '192.168.1.110' on port #5432
Connection established

Beginning active pinging...

Devices found:
-Resolving-  <08:00:28:B4:7A:85>  <HANDHELD_PC>

Sending >> IN -Resolving- <08:00:28:B4:7A:85> <HANDHELD_PC> 3.00


Devices found:
Loox PDA  <08:00:28:B4:7A:85>  <HANDHELD_PC>

Sending >> UPD Loox$PDA <08:00:28:B4:7A:85>
```

Figure 12. Update messages sent when a client name was previously unknown

This approach means that the small inaccuracy has no impact on the system, as device names are updated as soon as possible, and the system can still work fully using just addresses (which are the unique device identifiers).

5.1.2 – RSSI Refinement

The task of the client is to enumerate devices within range and keep the server up to date with this information. Previous evidence has shown this to be fully functional and sufficient for the network. The only other part of the client that can be tested is the range finding method that was put forward in the background chapter.

The RSSI (Received Signal Strength Indicator)-Refinement method uses information about the device type to predict its maximum distance from the client. An example of this is shown in Figure 12; the device type has been decoded to 'Handheld PC', with a maximum range of 10 metres. However, the client is known to have a maximum range of 3 metres, so the range of the device is updated, as it must be within 3 metres of the client. The range sent to the server (in the IN message) is 3 metres. If there are multiple devices in range of the client, they are returned in order of distance from the client, using RSSI measures among others, in methods described in the implementation.

Extensive testing would require several different types of Bluetooth device, which were not available for this project. The method has been tested with different devices such as a desktop Bluetooth adapter, mobile phone and handheld pc. The RSSI-Refinement is dependent on the performance of the order of return from the search, as this determines the way in which device range refinements can be made. The order of return works very well as long as the devices are more than 0.5 metres from the client; closer than this and the Bluetooth adapter struggles to work out which device is closest, even if the other is many metres away.

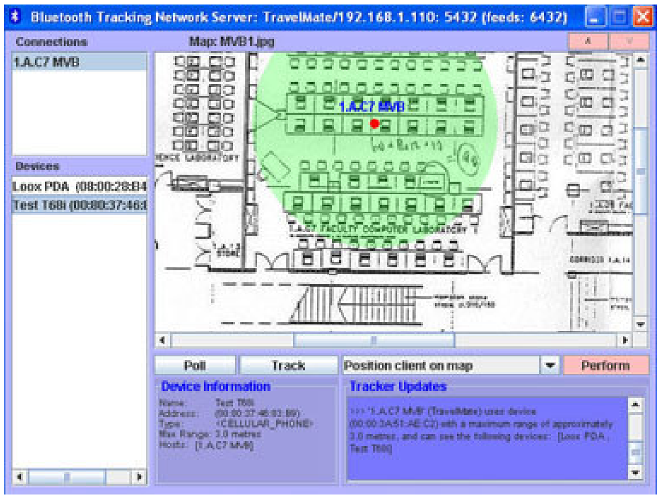
The RSSI-Refinement method has been shown to give a very crude measurement of range, which is accurate to a few metres. The benefits are that it is very simple and quick to implement, and requires no further clients or static beacons, making it a highly effective method of range finding when accuracy is not paramount.

5.2 – Server Results

The server is the main part of this project, and the part where the main locating and tracking network is enabled, through the processing of information from clients. Both the processing and extension modules have been tested at length, the results of which are shown below.

5.2.1 – Server Operation

The screenshot below shows the normal operation of the server. There is one connected client, named '1.A.C7 MVB', which has been placed in that room on the map. The client can 'see' two devices, 'Loox PDA' and 'Test T68i'. 'Test T68i' is selected in the list, updating the device information box in the bottom centre to show its details. The client has also been polled, the printout of which can be seen in the status window on the right.



Name: Test T68i
Address: (00:80:37:46:83:89)
Type: <CELLULAR_PHONE>
Max Range: 3.0 metres
Hosts: [1.A.C7 MVB]

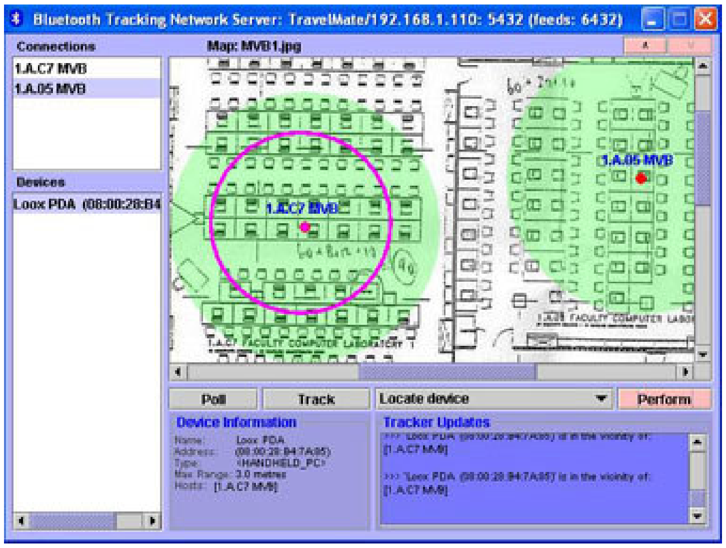
>>> '1.A.C7 MVB' (TravelMate) uses device (00:00:3A:51:AE:C2) with a maximum range of approximately 3.0 metres, and can see the following devices: [Loox PDA, Test T68i]

Figure 13. Normal operation of the server

Multiple maps can be scrolled through using the pink level buttons on the top right of the screen. Clients are shown in the top left pane, while devices on the network are shown on the bottom left.

5.2.2 – Device Locating

The server locates devices by looking at its host list. The host list stores the names of all clients the device is in range of. By knowing the positions of these stations, the server can find the approximate position of the device, which it prints in the status box. The server also automatically scrolls the map to show the device, and places a pink ring around it, highlighting its position.



Name: Loox PDA
Address: (08:00:28:B4:7A:85)
Type: <HANDHELD_PC>
Max Range: 3.0 metres
Hosts: [1.A.C7 MVB]

>>> 'Loox PDA (08:00:28:B4:7A:85)' is in the vicinity of:
[1.A.C7 MVB]

>>> 'Loox PDA (08:00:28:B4:7A:85)' is in the vicinity of:
[1.A.C7 MVB]

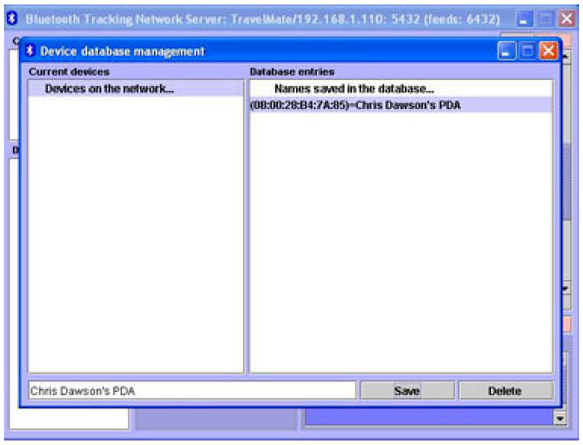
Figure 14. Locating a device on the network

In this example, client '1.A.C7 MVB' hosts the device on the network. Asking the server to locate the device causes the message about its location to be printed in the status window ('Loox PDA (08:00:28:B4:TA:85) is in the vicinity of: [1.A.C7 MVB]'). The server scrolls the map to show the location, and places a pink circle around it. The pink circle represents the range of the device, i.e. it encloses the area in which the device must be.

Because all information is stored and processed on the server, locating a device is instant and always up to date (provided the device is on the network). This is the most fundamental part of the project as everything else depends on effective, up-to-date locating.

5.2.3 – Name Database

The name database is used as an intermediary between the client and the user interface on the server. The screenshot below shows a database entry that has been created for a device address, which ties the address to a name.

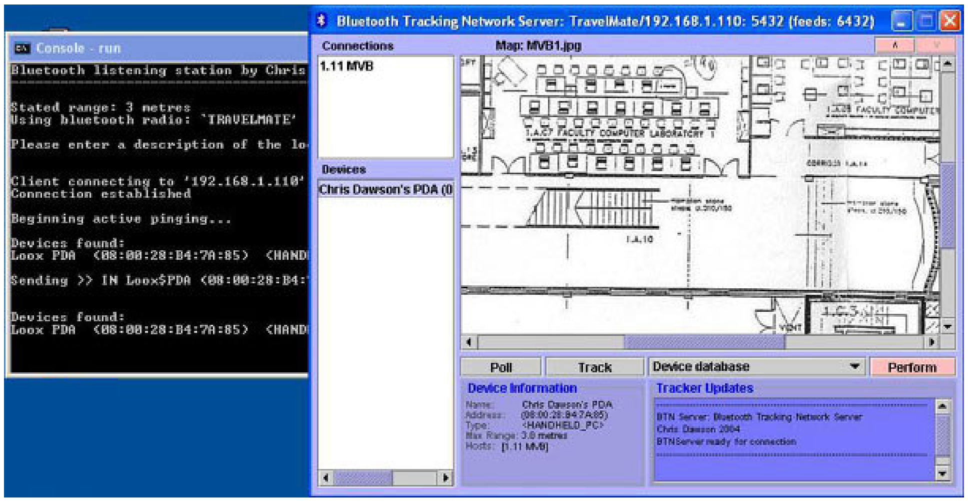

Bluetooth Tracking Network Server: TravelMate/192.168.1.110: 5432 (feeds: 6432)
Device database management
Current devices
Devices on the network...
Database entries
Names saved in the database...
Save
Delete

```
(08:00:28:B4:7A:85)=Chris Dawson's PDA
```

Chris Dawson's PDA

Figure 15. The device name database entry

The next screenshot shows the server's use of the database when the device enters the network. The client runs in the shell in the background; it picks up the device and sends its information to the server, with name 'Loox$PDA'. The dollar symbol replaces the space character. The server receives this information, which can be browsed in the device information window, but replaces the name of the device with that retrieved from the device name database.



```
Bluetooth listening station by Chris
-----------------------------------

Stated range: 3 metres
Using bluetooth radio: 'TRAVELMATE'

Please enter a description of the lo

Client connecting to '192.168.1.110'
Connection established

Beginning active pinging...

Devices found:
Loox PDA  (08:00:28:B4:7A:85)  <HAND

Sending >> IN Loox$PDA (08:00:28:B4:


Devices found:
Loox PDA  (08:00:28:B4:7A:85)  <HAND
```

```
Name:       Chris Dawson's PDA
Address:    (08:00:28:B4:7A:85)
Type:       <HANDHELD_PC>
Max Range: 3.0 metres
Hosts:  [1.11 MVB]
```

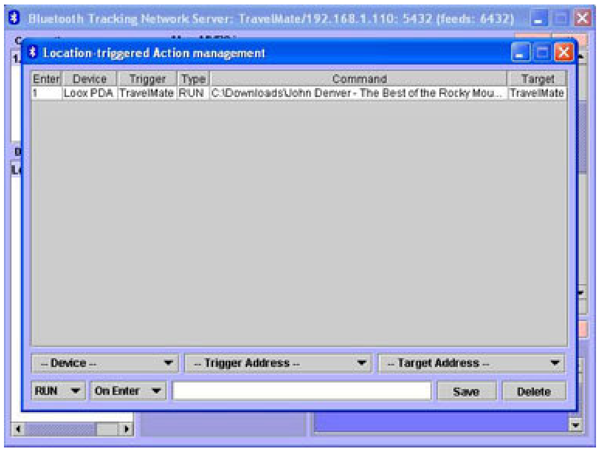
```
-----------------------------------
BTN Server: Bluetooth Tracking Network Server
Chris Dawson 2004
BTNServer ready for connection
-----------------------------------
```

Figure 16. Use of the device name database

5.2.4 – Location-Sensitive Actions

The location sensitive actions are server controlled, as shown in the next screenshot. Similarly to the name server, an action has been added to the database, using the controls at the bottom of the window. This action is displayed in the table.

The example action shown means: if the 'Loox PDA' device enters the vicinity of the computer with network hostname 'TravelMate', the server will send the command to run 'C:\Downloads\John Denver - The Best of the Rocky Mountain\05-john_denver-rocky_mountain_high-aaf.mp3’ to the client with network hostname ‘TravelMate’. In this instance, the trigger client (‘TravelMate’) is the same as the target client.

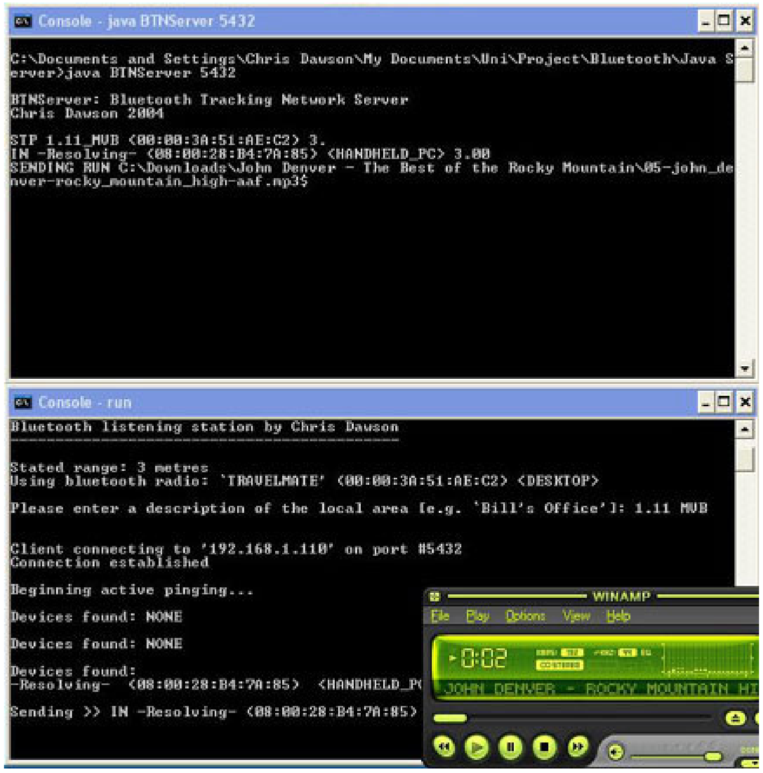

Bluetooth Tracking Network Server: TravelMate/192.168.1.110: 5432 (feeds: 6432)
Location-triggered Action management

Enter	Device	Trigger	Type	Command	Target
1	Loox PDA	TravelMate	RUN	C:\Downloads\John Denver - The Best of the Rocky Mou...	TravelMate

-- Device --
-- Trigger Address --
-- Target Address --
RUN
On Enter
Save
Delete

Figure 17. An example of a location-sensitive action

The screenshot below shows this example action in use. The server window is minimised, as the action is not a graphical one. The server prints all network traffic to the console for debugging and for the information of the administrator. The screenshot also shows the client window. During the running of the client, it picks up the presence of a device and notifies the server. The server checks this against the actions database, finds a match and sends the corresponding command to the target client. This message traffic is shown in the windows below. The client executes the command, which is an mp3 file, which opens by default in a commercial music player. The screenshot shows the execution of the file and the music player running in the side of the picture.



```
C:\Documents and Settings\Chris Dawson\My Documents\Uni\Project\Bluetooth\Java S
erver>java BTNServer 5432

BTNServer: Bluetooth Tracking Network Server
Chris Dawson 2004

STP 1.11_MVB <00:00:3A:51:AE:C2> 3.
IN -Resolving- <08:00:28:B4:7A:85> <HANDHELD_PC> 3.00
SENDING RUN C:\Downloads\John Denver - The Best of the Rocky Mountain\05-john_de
nver-rocky_mountain_high-aaf.mp3$
```

```
Bluetooth listening station by Chris Dawson
-------------------------------------------

Stated range: 3 metres
Using bluetooth radio: `TRAVELMATE' <00:00:3A:51:AE:C2> <DESKTOP>

Please enter a description of the local area [e.g. `Bill's Office']: 1.11 MVB


Client connecting to '192.168.1.110' on port #5432
Connection established

Beginning active pinging...

Devices found: NONE

Devices found: NONE

Devices found:
-Resolving-  <08:00:28:B4:7A:85>  <HANDHELD_P

Sending >> IN -Resolving- <08:00:28:B4:7A:85>
```

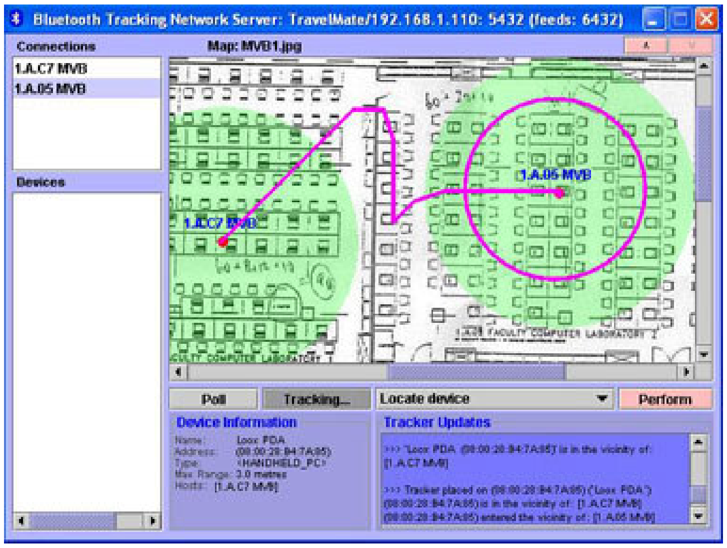
Figure 18. The location-sensitive action in use

Further tests varied parameters such as target clients and different commands (e.g. running executables). The actions are virtually instant, limited only be the network speed and the timer on the server. The tests showed that the system can be used to quickly and effectively trigger operations on remote computers; operations that are determined by the movement and locations of Bluetooth devices.

5.2.5 – Tracking

The main focus of the project, and that that is most complicated is the tracking of devices between clients. The visual representation of the data is through the mapping system, but the paths are complex in that they must use a graphing algorithm (the A* algorithm) to prevent paths plotted in straight lines, that would not accurately reproduce the movements of the device.

An example of the tracking system is shown in the screenshot below.



1.A.C7 MVB
1.A.05 MVB

Name: Loox PDA
Address: (08:00:28:84:7A:05)
Type: <HANDHELD_PC>
Max Range: 3.0 metres
Hosts: [1.A.C7 MVB]

```
>>> 'Loox PDA' (08:00:28:84:7A:05) is in the vicinity of:
[1.A.C7 MVB]

>>> Tracker placed on (08:00:28:84:7A:05) ('Loox PDA')
(08:00:28:84:7A:05) is in the vicinity of: [1.A.C7 MVB]
(08:00:28:84:7A:05) entered the vicinity of: [1.A.05 MVB]
```

Figure 19. The tracking system in use

In this example, the device 'Loox PDA' has moved from room 1.A.C7 to 1.A.05. The device has moved from the vicinity of one client to another, and its movements have been tracked and plotted on the map. There are an infinite number of ways for the system to plot this movement, the simplest being a straight line (which would go straight through the wall), although a more likely option is to go through different doors. The route-planning module has given the shortest path between clients, which is plotted on the map above.

The image shows that the route-planning module and tracker works as required, plotting the most likely route between clients, avoiding obstacle such as the wall. However, the shape of the path after the wall is obviously not what was expected. This strange shape is due to the heuristic used in the A* algorithm. The heuristic attempts to predict the shortest path by using a certain function, in this way the number of graph nodes visited is reduced, but accuracy suffers. This case shows that the heuristic could be improved, but the route gives a good approximation of the device's movement.

5.2.6 – Inferred Client Position

To properly test the inferred client position requires at least four computers. Each of the four computers must act as a client, with one also acting as the server. One of the computers must have its position inferred, in which case the other three clients can be used to plot up to 6 paths (the device can move from each client to any other, in either direction). The paths are used as described in the implementation to find the most likely position of the inferred client.

The project is ready to use client position inference, given sufficient clients. Results of this module cannot be obtained in this project because of the lack of computers available. This module is therefore a research suggestion to the problem of locating clients.

CHAPTER 6 – CONCLUSIONS AND FUTURE WORK

6.1 - Conclusions

The Bluetooth Tracking Network proposed in this project meets the aims set in the introduction. The main aim of the project was to use Bluetooth technology to provide an accurate way of locating a device then tracking its movements. The context of the project was an office or outdoor location, where pinpoint accuracy is unnecessary; this project is intended to find the general position of a client, for example within a room or general vicinity.

Some interesting images and results have been obtained to show the way that the network operates. Due to the nature of any tracking system, it is almost by definition something dynamic. This means it is hard to fully inform the reader of the capabilities and performance of the network simply on paper. Although pictures can show some information, they miss out important transitions in the states of the system.

The network has been shown to be capable of instantly locating a device on the network. The accuracy is dependent on the number of clients and their ranges, but in an office environment the normal accuracy is to within 2 or 3 metres (or accurate to the room).

Tracking devices is performed in real time using updates from clients to plot positions as the device moves. The path finding system works very well but always finds the shortest route so is susceptible to error if only very few clients are used.

Two example programs showed further uses of the system, implementing location aware computing and the possibility for tracking down devices on the move. An extension of the network also showed the opportunity for location-sensitive actions that are performed by the server, using device movements to trigger commands on remote systems.

This project has shown the opportunities available with a Bluetooth locating and tracking network, particularly in an office environment due to the enclosed ranges. The findings have been presented such that others can continue this work in the future.

6.2 – Extensions

As with any project, there is always room for improvement and extension. Some examples of this possible future work are discussed below.

6.2.1 – Client Control Plug-ins

One of the most exciting extensions for this project is the possibility of client plug-ins. The plug-ins could enforce control over a range of hardware of software, for the purposes of extending the possible location sensitive actions. For example, a plug-in could be added to the client that manipulates external electronic relays, to open doors or turn lights on or off when a device enters or leaves an area.

The client is ready for such an extension as it has two event commands (on device enter/on device leave) and any arguments can be passed to plug-ins or programs within the command string from the server.

6.2.2 – Improved Client Position Inference

An interesting extension to the project would be to improve the method of inferring an unknown client's position. This project has adopted a method that projects paths onto an activity matrix, before finding the patch with highest activity. It is assumed that the position with highest activity is the most likely position of the client. This method is dependent on the accuracy of the A* algorithm, which will always give the shortest route between A and B; if there are not enough clients, this path may not be an accurate representation of the true route taken.

A more complex and useful method would be to analyse the probabilities of certain paths being taken rather than using a single path. The terrain could also be analysed to use a restrictive method that blocks off certain parts of the map unless a 'gate' client picks up the device. In this way, the inference relies less on routing algorithms and more on the layout of the building and probabilities.

6.2.3 – Different Clients

A useful extension could be the implementation of a client for other platforms. The client proposed in this project is limited to Windows XP SP2. It may be beneficial to make use of the open source Bluez project (Linux Bluetooth stack) to provide a client for Linux stations. This could also open up the possibility of custom clients with embedded operating systems and tiny form factors, to be placed discreetly around an amusement park or office.

6.2.4 – Tracker & Listener Combination

Two example programs were made for this project; one queries the server for its position while the other receives updates on the location of the tracked device. An interesting extension would be to combine these programs and add a floor plan that can be provided by the server. In this way, the combined program could show the user where he/she is, where the target device is, and the shortest route to get to the target (using the A* algorithm already implemented in this project).

6.2.5 – History

At the moment, the tracking network only knows about devices that are being tracked or are on the network. A nice feature would be to remember device details so that information can be retrieved; such as the last time and place the device was seen.

GLOSSARY

Blocking
The term given to a method or function that waits for a resource before continuing, e.g. a blocking method that reads from a socket will wait until there is a message on the socket.

Broadcasting Station
A synonym for a *client*.

Client
The software that runs on a workstation and continually searches its local vicinity (using a Bluetooth adapter) for Bluetooth *devices*.

Device
Any Bluetooth enabled equipment that is not a part of the network infrastructure (i.e. not a *client*, *server* or *nomad*).

DTD
An acronym for "Distance To Device", meaning the distance separation between a *client* and a *device*.

Feed
A special client that simply receives textual updates on the movement of a tracked *device*, for purposes of tracking the tracked *device* down in the field.

Listener
A Java implementation of a *feed*.

Listening Station
A synonym for a *client*.

Nomad
A special device that is a part of the network infrastructure and can query the *server* for its location (or the location of another device). In this way, the nomad is aware of its location.

RSSI
The Received Signal Strength Indicator, a measure of the signal strength between two *devices* (or a *client* and *device*).

RSSI-Refinement
A crude method of range finding using the RSSI readings of multiple devices to place them in distance order from the client, allowing refinement of predicted distances.

Server
The Java software that collates information from multiple clients and implements the locating and tracking, as well as the extensions.

Speculative Range Finding
A method of range finding using the known device type to guess as its maximum range, for example of mobile phone has a maximum range of ten metres.

Tracker
A Pocket PC implementation of a *nomad*.

REFERENCES

[1] Matthew L. James: Where are you now? Location detection systems and personal privacy (Research Note No. 60, Parliamentary Library: Information and Research Services Group, Commonwealth of Australia, 15 June 2004)

[2] Abhishek Pramod Patil: Performance of Bluetooth Technologies and their Applications to Location Sensing (Masters Degree Thesis, Department of Electrical and Computer Engineering, Michigan State University, 2002)

[3] Openwave: Overview of Location Technologies (November 19, 2002)

[4] Lionel M. Ni, Yunhao Liu, Yiu Cho Lau, Abhishek P. Patil: LANDMARK: Indoor Location Sensing Using Active RFID* (Department of Computer Science Engineering, Michigan State University with the Department of Computer Science, Hong Kong University of Science and Technology)

[5] Specification of the Bluetooth System

[6] Ana Zapater, Kyandoghere Kyamakya, Silke Feldmann, Marc Kruger, Isaac Adusei: Development and Implementation of a Bluetooth Networking Infrastructure for the a Notebook University Scenario (International Conference on Wireless Networks, 2003)

[7] Silke Feldmann, Kyandoghere Kyamakya, Ana Zapater, Zighuo Lue: An indoor Bluetooth-based positioning system: concept, Implementation and experimental evaluation (International Conference on Wireless Networks, 2003)

[8] Bruce Hopkins and Ranjith Antony: Bluetooth for Java, pp. 33-34 (Apress, 2003. ISBN 1-59059-78-3)

[9] Junko Yoshida: Danish zoo to deploy Bluetooth tracking system (EE Times, June 2003)

APPENDIX A – User Manual

Client

About the Client

The client is the part of this project that searches for Bluetooth devices and notifies the server of what it finds. The client can also perform commands on behalf of the server, which instructs the client as certain events occur (for example a device enters a certain room).

The client was produced as a C++ project within the Microsoft .Net developing environment, with reference to the Microsoft Platform SDK, which provides Bluetooth header files. The client requires both these components to compile.

Running the client requires a Bluetooth adapter using native Microsoft drivers, as well as the .Net Framework. The former is available in the Microsoft XP SP2 updates, while the latter is available for free download; without these updates the client cannot run.

Usage

The client takes a maximum of three command line arguments as shown in the example below:

```
lStation /range 3.0 /server TravelMate /port 5432
```

The range argument is optional, if not included the program will provide a list of options and ask for the range. In this example, the range is set on the command line to three metres. The server flag must be followed by the IP address or hostname of the BTN server, which in this example is 'TravelMate'. The final argument specifies the port number on the server to connect to, in this example 5432.

The client will ask for a description of the local area, this should be the name of the room or local vicinity, for example: 'Paul Smith's Office' or 'Candy Floss Stand'.

The client will then enter an infinite loop, looking for devices and communicating with the server. If the connection to the server is lost, the program will give a reason for the problem and exit. To stop the program you must break out of it using CTRL+C.

Server

About the Server

The server is the main part of the project in terms of tracking and other functions of the network. It collates information from all connected clients and processes it in a way useful to the user, providing added functionality such as location-sensitive actions and automatic client positioning.

The server is written in Java and requires no additional frameworks or installations.

Usage

The server is compiled using the following command:

```
javac BTNServer.java
```

The server takes a single argument specifying the port number on which to listen for connections:

```
java BTNServer 5432
```

The server will also open up two more ports for feeds (tracker updates sent to other computers) and nomads (computers that can query their location). These port numbers are as follows:

Feeds	Portnumber + 1000	(6432)
Nomads	Portnumber + 2000	(7432)

If any port is unavailable, the server will automatically choose the next available port above that specified. For example, if port number 6432 is taken, it will attempt to use 6433. The server will notify the user if it is using a port number other than that requested.

Controls in the main program are very easily found and labelled. Instructions are automatically provided for more complex functions in the status box found on the bottom right of the window.

Listener

About the Listener

The listener is a simple Java implementation of a feed that connects to the feed port of the server. The server sends all feeds updates on the position of a tracked device. The listener prints these messages to the standard output.

Usage

The listener is compiled using the following command:

```
javac BTNListener.java
```

The listener takes two arguments specifying the server and the port number to connect to. The server argument can be an IP address or hostname:

```
java BTNListener TravelMate 6432
```

The program will enter an infinite loop, waiting for messages from the server and printing them out. If the server shuts down, the listener will alert the user and exit. To exit the program you must break out of it using CTRL+C.

Tracker

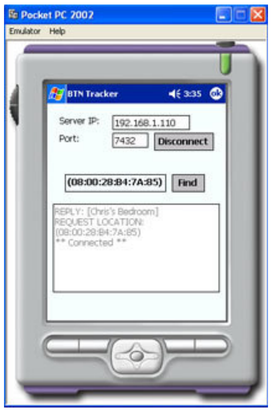
About the Tracker

The tracker is a Pocket PC implementation of a 'Nomad', that is a mobile device that can query the server for its location. To do this, the tracker must be run on a device that has Bluetooth and networking capabilities. Alternatively, the tracker can be used to query the location of another device, in which case only networking capabilities are needed.

The tracker was developed in the Microsoft .Net developing environment, which is required for compilation. The program can be deployed to a Pocket PC using .Net or a standalone installer, at which point it will be installed for use at any time.

Usage

The tracker takes no arguments but requires some parameters to function.



Server IP: 192.168.1.110
Port: 7432
(08:00:28:B4:7A:85)

```
REPLY: [Chris's Bedroom]
REQUEST LOCATION:
(08:00:28:B4:7A:85)
** Connected **
```

Figure 20. The Tracker

The address of the server and the port number to connect to is required. When this information is entered, the user can click 'connect' to establish a connection with the server.

The Bluetooth address of the device to be located (normally this device) is entered in the third text box. Clicking on 'Find' will dispatch a message to the server querying the location of the specified device.

If the server is aware of its location, it will respond with a list of clients that can 'see' the device. If the server is unaware of the device, it will return 'Unknown'.

APPENDIX B – Background Research

One of the main benefits of this project is that it uses commercial equipment that is very cheap and freely available. As a form of background research for this project, I looked at the possible impact and market the system has in a typical office atmosphere, which is the proposed setting of the system.

This research looked at the number of devices 'seen' by a single client, positioned in the atrium of the Merchant Venturers Building (University of Bristol) in one hour. The client recorded the device names, addresses and types. The devices were grouped by device-type, and the results are shown in the table below.

The research was performed at 12.00-1.00 on March 17 2005.

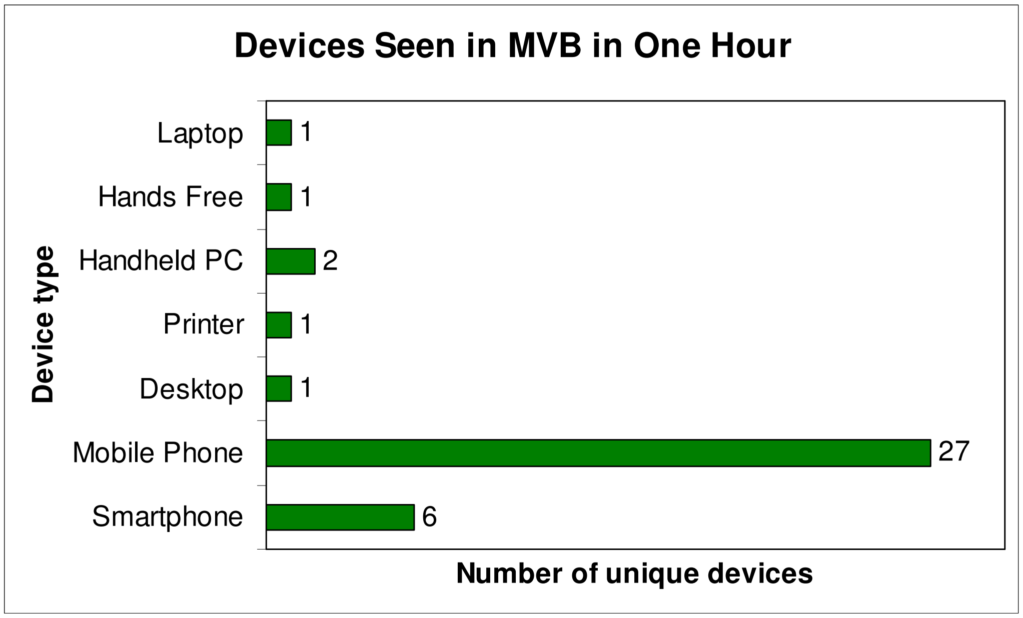


Figure 20. A chart showing the number of devices found in MVB in one hour

An excerpt from the research is shown below. Some results have unknown names; this was described previously in the implementation and is a consequence of rapid device enumeration. It is not necessary to know the name of a device, as the address is the identifier, and is always returned. The name is only for human readability; names can be assigned to device addresses on the network for the ease of the user.

(00:0E:ED:7B:F0:89)	Jimbo g	<CELLULAR_PHONE>
(00:0E:6D:50:D6:15)	Kat	<CELLULAR_PHONE>
(00:12:62:C8:60:FE)	Martyn's Nokia	<CELLULAR_PHONE>
(00:0E:07:6F:7F:26)	Mike Owen (Phone)	<CELLULAR_PHONE>
(00:0E:6D:11:BE:35)	Mu	<CELLULAR_PHONE>
(00:0E:07:8F:BD:30)	Patrick	<CELLULAR_PHONE>

(00:0E:6D:95:11:47)	Timothy's phone	<CELLULAR_PHONE>
(00:11:9F:75:E8:3A)	Tom's Nokia 6260	<SMARTPHONE>
(00:12:62:0B:6F:CE)	Unknown	<CELLULAR_PHONE>
(00:08:F4:00:58:91)	Unknown	<DESKTOP>
(00:05:16:5D:FC:0A)	Unknown	<PRINTER>
(08:00:1F:B7:95:A4)	Unknown	<CELLULAR_PHONE>
(00:02:EE:5F:93:40)	Unknown	<CELLULAR_PHONE>
(00:11:9F:C8:27:2A)	Unknown	<SMARTPHONE>

These results show that not only is the technology widely used in a variety of devices, commonly found in the office environment, but it is left enabled (available for discovery) in many cases. The research shows that in a quiet lunchtime hour shortly before the end of term, thirty-nine unique devices were picked up.

The results illustrate that there is a real possibility of locating and tracking people, with or without their knowledge. It also demonstrates the prevalence of Bluetooth technology in modern consumer equipment, increasing the possibility of implementation and scale of such a tracking project.