USING WIRELESS COMMUNICATIONS FOR LOCATION-BASED SERVICES

by

Timothy J. Dawbarn

B.S.M.E. University of Michigan, Dearborn Michigan 1995

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science

_____
Dr. Joseph Evans
Professor in Charge

_____
Dr. Victor Frost
Committee Member

_____
Dr. Jeremiah James
Committee Member

Date Defended: August 5, 2003

UNIVERSITY OF KANSAS

ABSTRACT

USING WIRELESS COMMUNICATIONS
FOR LOCATION-BASED SERVICES

By Tim Dawbarn

Chairperson of the Supervisory Committee: Professor Dr. Joseph Evans
Department of Electrical Engineering and Computer Science

Providers of location-based services are currently faced with what might be referred to as the "last block" problem. That is, locating users to the nearest city block can be done readily using relatively mature technologies, but offering services to users based on what room they are in, or where they are standing in that room, is still a challenge. The architecture proposed in this document uses wireless communications to solve this problem. The document further describes several implementations of this architecture designed to utilize the finer grained location information to provide users with useful services.

TABLE OF CONTENTS

6. Conclusions and Future Work

LIST OF FIGURES

LIST OF FIGURES

ACKNOWLEDGMENTS

The author wishes to thank the good people at Coca-Cola for providing tasty beverages that are both high in caffeine *and* contain a lot of sugar.

Chapter 1

INTRODUCTION

1.1 What are Location-Based Services?

Location-based applications or services are those in which the location of a person or an object is used to shape or focus the application or service [1]. Some common examples of location-based services are using a mobile telephone to find the nearest coffee shop, or getting driving directions using a handheld GPS unit. The location-based services focused on in this document are more local in nature. Services based on the user's location within a building or room are the concern.

1.2 Motivation

Mature and widely used technologies like GPS, Angle of Arrival, and Time Difference of Arrival are good for locating users with a fairly large margin of error, but finer-grained location technologies are not as prevalent. Users can obtain directions to the nearest shopping mall from their approximate location with relative ease, but locating a particular store in the mall is still left to traditional methods. Furthermore, technologies that offer users services once they have reached their destinations are mostly non-existent. The current state of location-based technologies is analogous to the "last mile" problem in broadband networking: bringing the technology to the neighborhood level had been successful, but continuing on to every doorstep presented a challenge. A new

technology is needed to bridge this gap. The technology needs to be relatively inexpensive, scalable, and easily adopted by end users.

1.3 Project Goals

There are two main goals for this project. The first goal is to develop a technology that bridges the gap in location-based services by providing finer-grained location information to computing devices. The second is to develop a suite of applications which utilize the increase in location accuracy to provide end users with useful services.

1.4 Layout of this Document

This document is organized into the following sections:

- Introduction – This is an overview of the proposed system and why it is needed
- Background – This is information that is necessary for a better understanding of this project.
- Architecture – Detailed description of what the system does and how it is accomplished.
- Implementation – Description of a real system that follows the specifications described in architecture.
- Results – Description of the test environment built to evaluate the implemented system and the results of the testing.

- Conclusions and Future Work – Documentation of the lessons learned from this project and ideas for how to improve it in the future.

Chapter 2

BACKGROUND

2.1 Vision

The goals of this project are to create a system that will enable fine-grained location-based services and to create applications that utilize this system. The user will be able to access services based on the building, store, apartment, or office they happen to be standing in. The services will be offered in formats that users are already familiar with, such as web pages, in order to lessen the learning curve. This document will describe an architecture that enables the achievement of these goals. The architecture will be implemented with minimal cost and will utilize equipment that already has a large installed user base.

2.2 Supporting Technologies

2.2 1 IRDA

"Infrared Data Association is an International Organization that creates and promotes interoperable, low cost infrared data interconnection standards that support a walk-up, point-to-point user model. The Infrared Data Association standards support a broad range of appliances, computing and communication devices" [2]. Among the standards specified are those which define the physical layer (IrPHY), and the data link layer (IrLAP).

IrPHY specifies how the actual infrared light is modulated in order to communicate data. The physical layer is similar to serial communications but with a change to the duty cycle. The actual on-time of the infrared light is much less than the off-time in an effort to save battery power. Each byte is transferred with one start bit (0) and one stop bit (1) with no parity.

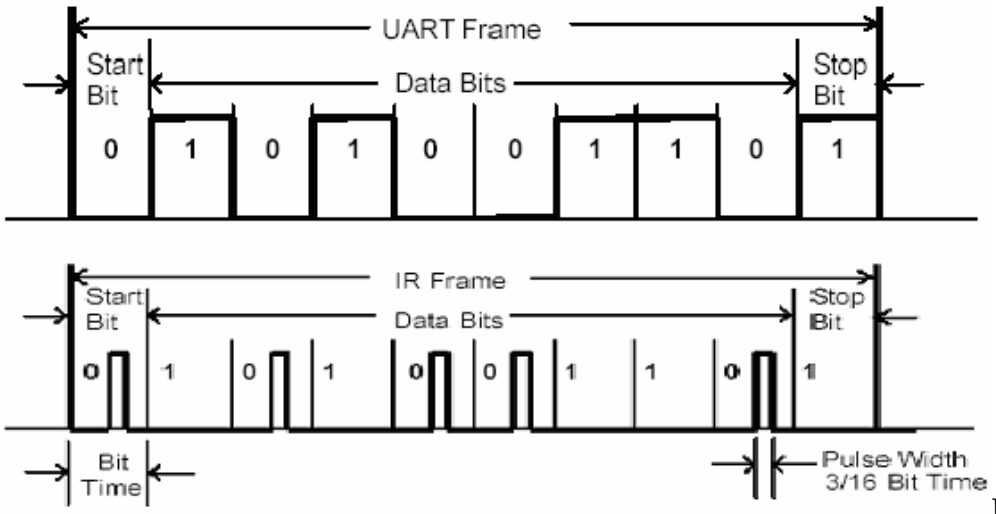

Figure 2.1: IRDA physical layer encoding

IrLAP specifies how a data frame is constructed and transmitted. First a number of (normally 10) beginning of frame bytes (0xFF) are sent followed by one start of frame byte(0xC0). Next the IrLAP payload which consists of address, control, and information fields is sent. The payload is followed by a 16 bit frame check sequence generated using the CRC-CCITT algorithm on the payload data. Finally one end of frame byte (0xC1) is sent.

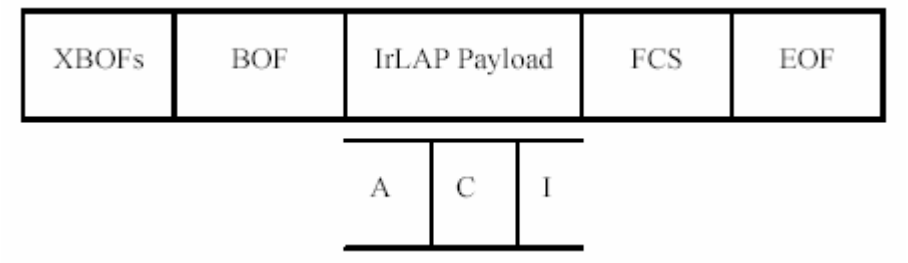

Figure 2.2: IrLAP frame structure

All of the infrared communications used in the implementations of this system conform to the IrPHY and IrLAP standards.

2.2.2 OBEX

Object Exchange Protocol or OBEX is a higher level protocol developed by the IRDA for the exchange of data objects between devices. OBEX will work on top of many lower level protocols including infrared, TCP/IP, and Bluetooth. It has been described as similar to a binary HTTP protocol. The OBEX specification consists of two major parts: a protocol and an application framework. "The OBEX protocol is a session level protocol that specifies the structure of the conversation between devices" [3]. The protocol also defines a method for describing objects. The structure of the description is very similar to

HTTP. Data is organized into various headers which are specified by a header identifier and followed by the associated data which may include length, name, type, and other information about the object.

HI - identifier	header name	Description
0xC0	Count	Number of objects (used by Connect)
0x01	Name	name of the object (often a file name)
0x42	Type	type of object - e.g. text, html, binary, manufacturer specific
0xC3	Length	the length of the object in bytes
0x44 0xC4	Time	date/time stamp – ISO 8601 version - preferred date/time stamp – 4 byte version (for compatibility only)
0x05	Description	text description of the object
0x46	Target	name of service that operation is targeted to
0x47	HTTP	an HTTP 1.x header
0x48 0x49	Body End of Body	a chunk of the object body. the final chunk of the object body
0x4A	Who	identifies the OBEX application, used to tell if talking to a peer
0xCB	Connection Id	an identifier used for OBEX connection multiplexing
0x4C	App. Parameters	extended application request & response information
0x4D	Auth. Challenge	authentication digest-challenge
0x4E	Auth. Response	authentication digest-response
0x4F	Object Class	OBEX Object class of object
0x10to 0x2F	reserved	this range includes all combinations of the upper 2 bits
0x30 to 0x3F	user defined	this range includes all combinations of the upper 2 bits

Figure 2.3: OBEX Header Identifiers

Along with the data headers, the full OBEX standard defines many directives for performing various actions such as transferring a file or establishing a connection. For this system, communications are limited to a subset of the session protocol that deals with connectionless data transfer. This subset is referred to as Ultra OBEX. In Ultra OBEX only two directives are supported: PUT and GET. This system is primarily concerned with the PUT command. The structure of an OBEX packet is typically a directive followed by headers. The following sequence corresponds to the transfer of a file called "myurl.url" using an OBEX PUT.

Bytes	Meaning
0x82	PUT command final bit set
0x0038	Length of OBEX packet
0x01	Name Header ID
0x0017	Length of Name Header
myurl.url	Name Data Unicode Null Term
0xC3	File Length Header ID
0x00000016	File Length
0x49	File Body Header ID Final Segment
0x0019	Length of File Body Header
http://www.ittc.ku.edu	File Body Data

Figure 2.4: Sample OBEX packet

2.2.3 IEEE 802.11b

Mobile devices are important tools for location-based services. The devices and the applications that run on them are even more useful if they are connected to some type of network. While any type of network connection will work, it makes sense for a mobile device to have a wireless connection. Many technologies exist for wireless networking including Bluetooth, GPRS, and IEEE 802.11. An Internet connection via IEEE 802.11b was chosen for this project because of its availability, wide range of support, and flexibility, but it should be noted that other technologies may eventually play a more significant role in these types of applications. The IEEE 802.11b protocol allows for data transfer rates as high as 11 megabits per second from distances up to 160 meters and data transfer at degraded rates from as much as 550 meters. IEEE 802.11b devices can obtain an Internet address automatically from a DHCP server and employ various encryption methods (WEP) to avoid sending clear text through the air. IEEE 802.11b cards are available in

both PCMCIA and compact flash formats and work with a wide variety of mobile computing devices.

2.2.4 Apache HTTP server

Offering location-based services to users in the form of a web page is very convenient. In order to provide web pages over the network, we need a server that is running web server software. The Apache HTTP Server was chosen for the implementations in this project. The Apache HTTP Server is a project of the Apache Software Foundation. It is an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of the Apache HTTP Server project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards [4].

Apache is the most popular web server on the Internet and is the one used for this project. The Apache HTTP Server used includes support for the Common Gateway Interface (CGI), and Secure Socket Layer (SSL) communications both of which are used for this project.

2.2.5 MySQL

MySQL is an open source relational database management system. The MySQL database server is the world's most popular open source database. It is fast and easy to customize. The server can be run with strict transaction control or with transactionless disk access [5].

As the name implies, the MySQL server supports queries and commands written in the Structured Query Language (SQL). Although not every SQL command is supported by MySQL server [6], all capabilities needed by this project are available.

2.2.6 X10

X10[7] is a relatively old but still widely used technology for home automation. Devices that implement the X10 protocol can be controlled via signals that are transported across electrical wiring. The item to be controlled (lamp or small appliance) is plugged into an X10 module. The module is then plugged into a regular wall outlet. A special control module (CM11A) which is connected to a computer's serial port is also plugged into a different wall outlet. The computer can then send X10 commands through the electrical wiring to various modules to turn devices on and off. Cameras, thermostats, and security equipment that communicate using the X10 protocol are also available.

2.2.7 Heyu

Heyu is a Linux program that will operate the CM11A computer to X10 interface. It can control X10 devices, set the CM11A interface clock, and monitor X10 signals that are transmitted over the AC power lines. It is capable of uploading macros and timers to the CM11A for stand-alone execution. It is tailored to Linux, but it should work with other UNIX systems. Heyu has a command line interface for controlling X10 devices. For example, the following command turns the device connected to module A1 on.

```
>heyu turn a1 on
```

2.2.8 vCard

vCard is a standard maintained by the Internet Mail Consortium. "vCard automates the exchange of personal information typically found on a traditional business card. vCard is used in applications such as Internet mail, voice mail, Web browsers, telephony applications, call centers, video conferencing, PIMs (Personal Information Managers), PDAs (Personal Data Assistants), pagers, fax, office equipment, and smart cards. vCard information goes way beyond simple text, and includes elements like pictures, company logos, live Web addresses, and so on" [13]. A simple vCard can be just plain text. The following is an example of a simple vCard with just the name and two telephone numbers.

```
BEGIN:VCARD
VERSION:2.1
N:Friday;Fred
TEL;WORK;VOICE:+1-213-555-1234
TEL;WORK;FAX:+1-213-555-5678
END:VCARD
```

2.3 Related Work

Some projects that employ concepts that are similar to those presented in this document are the Cricket project [8] at the MIT computer science laboratory, a CDPD-based user location system developed by AT&T labs [9], and the 802.11 based system implemented at the Rice University Department of Computer Science [11].

2.3.1 Cricket

The Cricket project is an in-building location support system which utilizes beacons that broadcast via radio frequency and ultrasound as its enabling technologies. Applications on mobile and static nodes determine their location by listening to information sent from various beacons spread throughout a building. There is no central server and the Cricket system does not track individual users' locations. The system instead allows applications to determine their location and the share it with whomever they please.

The Cricket project seems to work well but there are two limitations. First, the implementation requires new unique hardware on both the user and provider ends. Second, since the user applications determine their location based on the distance from a beacon, the beacons themselves need to be placed somewhat scientifically.

2.3.2 CDPD

Researchers at AT&T labs and the University of Rome developed a system which uses Cellular Digital Packet Data (CDPD) to determine the location of users and offer services [9]. Their approach works nationwide on devices that communicate on a cellular network. The user's mobile device first determines the unique identification number of the base station that it is currently communicating with. Next, the base station ID is translated into a geographical location by means of a central server with a database of base station IDs and their locations. The location data is then transferred to information services providers through requests to a location server or by means of a proxy.

A good feature of this system is that it does not require any additional hardware to implement. A disadvantage, however, is that the inaccuracy of the location data is equal to the cell size, which can be quite large.

2.3.2 Rice University, Department of Computer Science

A project done at Rice University Department of Computer Science [11] uses only wireless Ethernet (802.11) to determine location information. The system works by measuring the signal strength of multiple base stations and then comparing the strength signature to a database of measured signatures using a Bayesian model to estimate location.

This system has the advantage of not requiring any specialized hardware but it also has some disadvantages. The system requires a significant amount of training to develop a state space with known signatures. If an access point is moved or added, or the arrangement of objects in the building changes significantly, the system will have to be retrained. Since 802.11 operates at the resonance frequency of water, humans tend to absorb signal. Because of this, the strength signatures will tend to differ as more people enter a room or building.

2.4 This Approach

A goal of the system proposed in this document is to provide very fine-grained location-based services to users in a relatively transparent manner. On the provider end, the system

must be cost effective, flexible, and easy to install. This system will use low-cost beacons that broadcast location information via infrared to mobile computing devices such as personal digital assistants or smart phones. Many standard mobile computing devices can be used without any hardware modifications.

Chapter 3

ARCHITECTURE

3.1 Overview

The architecture described in this chapter provides a sufficient guideline for constructing a location-based services system. Details of sample implementations are given in the next chapter.

The system can be broken down into the following items which are described in this chapter:

- Beacon
- Client Device
- Client Software
- Network
- Server Device
- Server Software

The system operates in a manner consistent with the diagram in figure 3.1. A user, carrying a mobile device that is running the client software, moves within the broadcast range of the beacon. The beacon broadcasts data that is related to its own location. The mobile device receives the broadcast and forwards it to the client software. The client software determines the type of location data received and then calls the appropriate handler application. If the data type that is received refers to a network resource, the

handler application makes a request to a server over the network. The server responds with a service that is based on the location where the user made contact with the beacon.

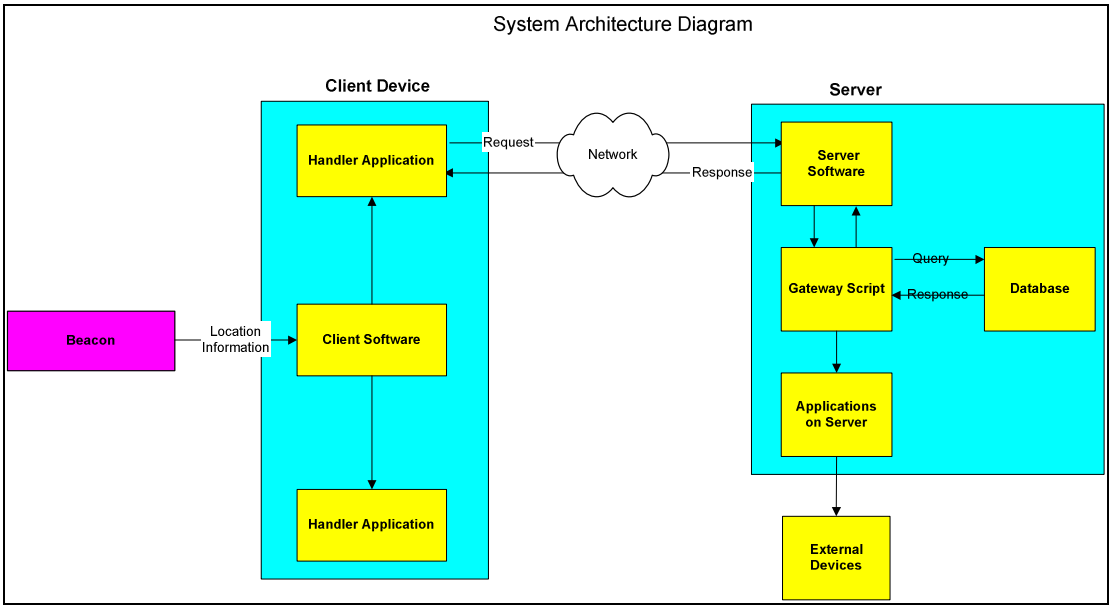

Figure 3.1: LBS System Architecture

3.2 Beacon

A beacon is a device which periodically broadcasts location information. It does not establish or maintain any connections. The information it transmits is not encrypted and is universally available. The beacon's location information is programmed with the location information once by the administrator and it acts autonomously from then on.

Since the beacon broadcasts information specific to its location, the physical method of data transfer is important. For example, the following questions should be answered.

Should the signal pass through walls, or bounce around corners? How far should the signal reach? Is the direction of receipt important?

The location information that is transmitted by the beacon can be one of two types: physical location, or logical location. Examples of physical location are longitude and latitude, or distance and angle from a known point. Logical locations are things like Starbucks, Nichols Hall, or Room 237. Typical location-based systems normally first determine the physical location and then map it to the logical. In this system, the beacon can broadcast either logical or physical location information.

3.3 Client Device

A user who wants to take advantage of location-based services needs to have some kind of mobile computing device that can access the system as a client. The device must have sufficient hardware to receive the location information from the beacon. The device must also be able to access a pool of location-specific resources once it has received the information from the beacon. This is typically done with some type of network access but could also be a local database. Lastly, the client device needs to be programmable in some way. There are many PDA type devices on the market that meet these criteria and there are a few mobile telephones that are perhaps even better candidates.

3.4 Client Software

The client device needs to have custom software installed that listens for beacons, parses the information received, and accesses the appropriate services. The client software needs to be as transparent as possible, but still allow the user to control the level of automation. The user should also be able to stop and start the software as needed. The client software should make an effort to provide some reduced functionality when a network connection is not available. The client software should be fault tolerant. It should be able to handle erroneous beacon broadcasts and network failures without crashing.

The client software should consist of at least two components: a user interface and a beacon data parser.

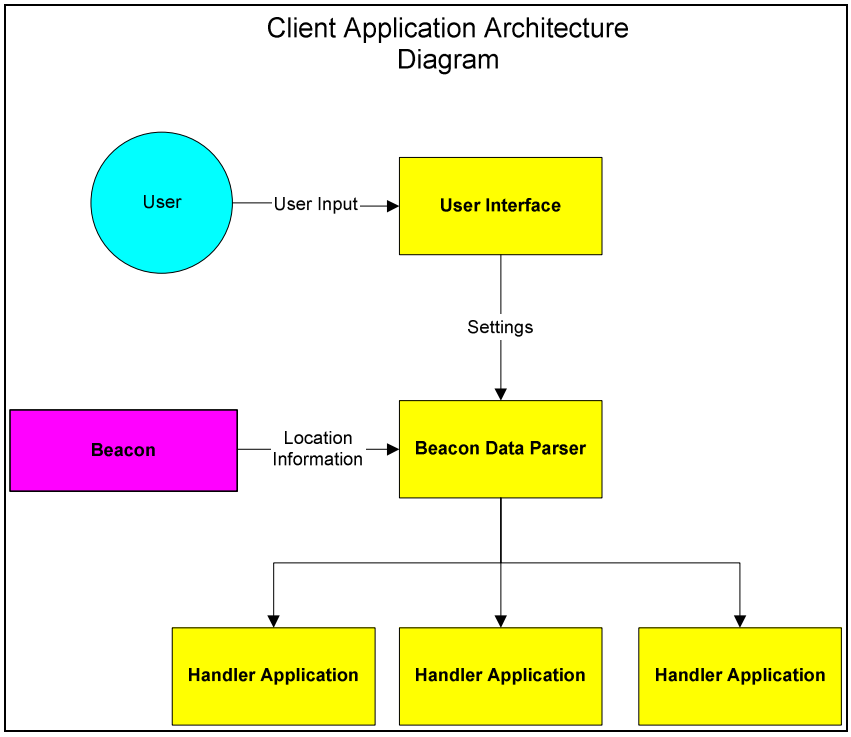

Figure 3.2: Client Software Architecture

The user interface will allow the user to configure some of the application options and will provide a means for prompting the user when interaction is necessary.

The beacon data parser component will listen on some interface for incoming data from beacons. When data is received, the parser will check for errors, and then process the data. The data parser will call the appropriate handler application once the data has been processed.

3.5 Network

As stated before, the client device needs to have access to a pool of location-based resources. Since most mobile devices have somewhat limited memory and storage, it would normally make sense to access these resources via a network. The network accessed could be a LAN, an intranet or the Internet depending on where the resources are located. An "always on" type of connection would provide for a more seamless user experience but is not strictly necessary. Since location-based services are most useful with mobile devices, wireless networking technologies such as 802.11 or GPRS are the most practical type of connectivity.

3.6 Server Device

If the location-based services are to be served from a network, there is a need for a server device that responds to client requests. The server needs to have a physical connection to the network that clients are going to use for requests and it needs to be able to take some

action based on the request. The server could be a desktop, a dedicated server, special purpose hardware, or another PDA as long as it has a connection to the same network as the client.

3.7 Server Software

The server needs to have software that listens for client requests and responds to those requests. The response could be a single transfer of data as in the HTTP protocol or it could establish a connection as might occur with FTP. For some applications, it may be useful for the server software to have some type of gateway interface in order allow client requests to control external software. Lastly it may be useful for the server to have access to a database to maintain information that is specific to users and locations.

Chapter 4

IMPLEMENTATION

4.1 Technologies

4.1.1 Beacons

The beacons used in this implementation communicate information to the client devices using infrared light pulses. The data encoding, error correction, object encapsulation, and pulse timing used is consistent with the specifications prepared by the Infrared Data Association (IRDA). The specific protocols implemented are IrPHY, IrLAP, and a subset of IrOBEX called ultra OBEX.

The beacons are implemented using low-cost electrical components that are commonly available. The central component of the beacon is a programmable microcontroller with built-in EEPROM memory. This memory holds the location specific information and can be reprogrammed by the system administrators with a desktop computer. The end user does not need to be aware of the beacon internals.

The beacons in this implementation are used to broadcast two types of information. The first is a URL (Uniform Resource Locator) or web address. In order to send the URL, the beacon needs to send an OBEX PUT packet with the size, type, and data headers. The rest of the OBEX headers can be omitted. The sequence of bytes that the beacon needs to send in order to broadcast a particular URL is given in figure 4.1.

Bytes	Meaning
FF FF FF FF FF FF FF FF FF FF	Beginning of frame sync
C0	Start of frame indicator
70	Source Address
70	Destination Address
0x82 0x0038 0x01 0x0017 myurl.url 0xC3 0x00000016 0x49 0x0019 http://www.ittc.ku.edu	PUT command final bit set Length of OBEX packet Name Header ID Length of Name Header Name Data Unicode Null Term File Length Header ID File Length File Body Header ID Final Segment Length of File Body Header File Body Data
FCS1 FCS2	16 bit Checksum
C1	End of Frame

Figure 4.1: Sample beacon URL broadcast

The second type of beacon broadcast is vCard data This is very similar to the URL broadcast except now the OBEX type header is text/x-vcard instead of text/x-url.

4.1.2 Client Device

The client device used in this implementation is a Compaq iPAQ 3150. It has a 200 MHz Intel StrongARM processor and 32 Megabytes total internal memory. The internal memory is used for both application data, and system RAM. The operating system used on the device is Microsoft PocketPC OS which is otherwise known as Windows CE 3.0. The iPAQ slides into a backpack which provides the PCMCIA support necessary for adding a network interface card. The iPAQ is equipped with a built in infrared data port which is located near the top end of the device.

4.1.3 Client Software

4.1.3.1 Overview

There are three applications which are used on the client device for this implementation: IrObexClient, VCardStore, and Microsoft Pocket Internet Explorer. IrObexClient listens for beacon broadcasts and then passes the information to the appropriate handler application. VCardStore is a handler application that stores business card information and Internet Explorer handles URLs. In the implementations described in this paper, the beacons are only used to send vCards or URLs, but the system could be modified to send and receive any type of file.

4.1.3.2 IrObexClient

IrObexClient was written in C++ using Microsoft Visual C++ for embedded systems and the Microsoft PocketPC platform SDK. The visual tools in the IDE were used to develop the graphical elements of the user interface.

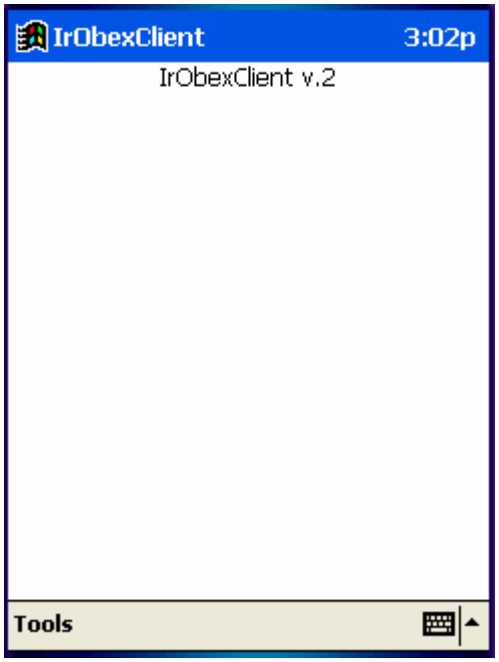

Figure 4.2: IrObexClient

The software behaves in the following manner. When the software is started, a window appears which displays the name "IrObexClient" and the version. At the bottom of the screen is a Tools menu. If no errors occur, this is all that is displayed until a beacon is encountered.

Clicking on Tools brings up the tools menu. It has three selections: about, settings, and addresses.

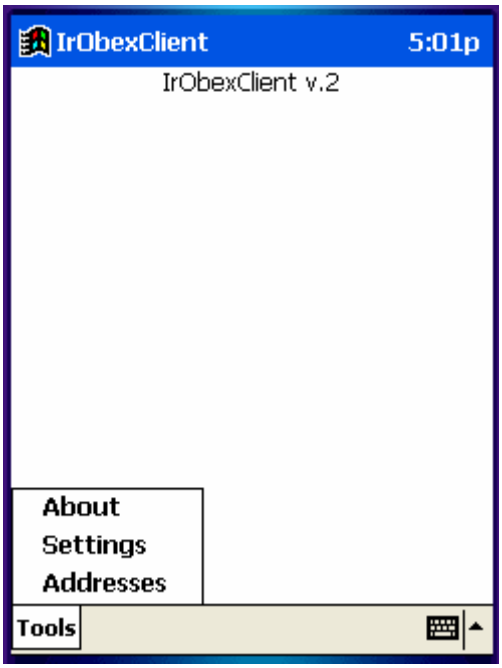

Figure 4.3: Tools Menu

IrObexClient has some user-configurable options that can be set from the "Tools->Settings" menu. The two options available are "Auto open URL" and "Auto Save vCard." The first option sets whether or not the user should be prompted before opening a URL. The second option controls whether the user will be prompted before saving a received vCard to the contacts list.

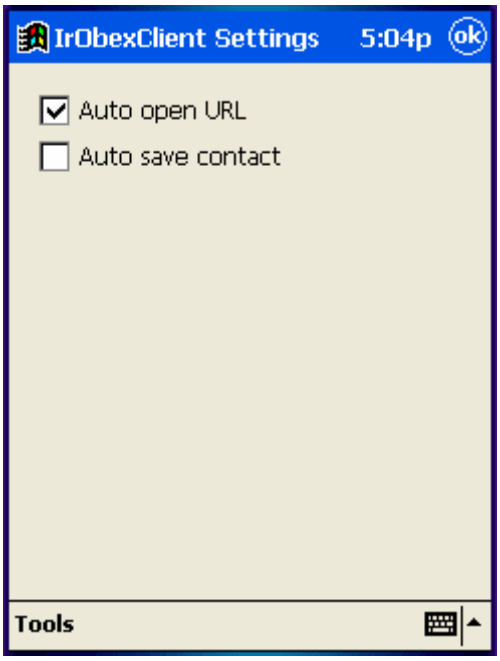

Figure 4.4: Setting Screen

Upon receiving a URL from a beacon, the client software launches Internet Explorer which accesses the web page pointed to by the URL. If the "auto-open URL" option is not enabled, the user is prompted first before it is opened. When a vCard is received, it is either automatically saved to the contacts database or the user is prompted with a "you have received a business card" message if "auto-save contacts" is not enabled. If the "auto-save contacts" option is enabled, the user will not notice the receipt of a business card until he/she looks at the list of contacts.

Before sending a URL to Internet Explorer to be opened, the client software first checks the URL to see if the resource is available. If it is not, the software gives the user the option to save the URL for later review. Saved addresses can be accessed by selecting the addresses option from the tools menu.

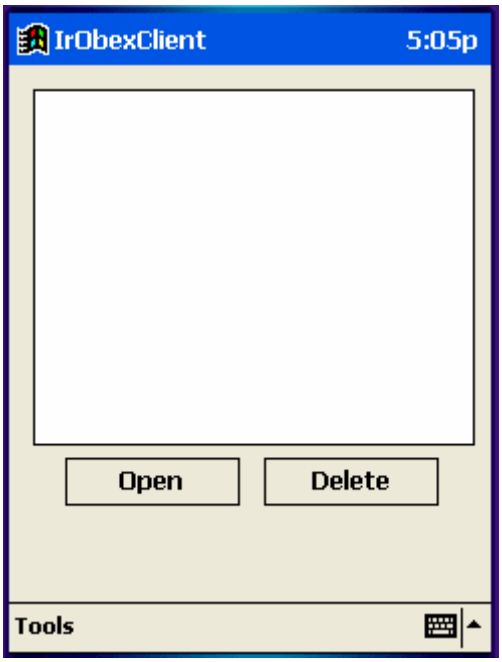

Figure 4.5: Addresses Screen

The following is a description of the functions contained in the client software and the sequence in which they are executed. See Appendix A for the client software source code. The program entry point is the WinMain function. The first function called inside WinMain is InitInstance. This function takes care of initializing the program and creating the windows and dialogs necessary for this application. Next an instance of IrdaParser is created.

IrdaParser is a class whose purpose is to listen for and process objects received via the infrared port. The objects are sent using the OBEX protocol which is in turn encapsulated by the IRLAP protocol. The IrdaParser needs to parse both protocols in order to receive the data.

After creating the instance of IrdaParser, the program calls its public startIR member function. This function first finds the COM port that is associated with the infrared port and opens it for reading. Next it sets some communications parameters before creating the main read thread.

Once created, the read thread continuously attempts to read bytes from the infrared port into a buffer. When some bytes are received, the buffer is passed to getFrame. The getFrame function searches the buffer for a valid IRDA frame. In order for a frame to be valid, it needs to begin with the correct start-of-frame character, have valid address and control bytes, and have a computed checksum which matches the one transmitted. If a valid frame is found, the frame type along with the buffer index where the frame begins is returned. At present ULTRA_OBEX_FRAME is the only frame type that is used. If no frame is found, the INVALID_FRAME type is returned. If the ULTRA_OBEX_FRAME type is returned, the buffer along with the start-of-frame index is passed to parseUltraObex.

The parseUltraObex function processes and stores data contained in the various OBEX headers sent out by the beacon. The headers include length, name, type, and body fields among others. The most important headers for this system are the body and the type. The type identifies the type of information that has been received and dictates which handler application is called. MIME conventions are used to represent the type such as "text/x-url" for a URL or "text/x-vcard" for vCard information. The body header contains the actual data associated with the specified type. If the type is a URL, the body

header contains a web address if it is a vCard, the body includes the name, phone number, and other business card information. After the parsing the OBEX frame, the information gathered from the headers is passed to the processFile function. The processFile function calls either processURL or processVCard depending on the value of the type field.

The processURL function simply converts the URL string to Unicode and then passes it to the openLocation function. openLocation first compares the URL string to the last URL opened before proceeding. This is to avoid reopening the same page repeatedly while standing in front of a beacon. Next the function checks to see if the auto-open URL option is enabled. If is not, the user is prompted before proceeding. After this step, the function uses the checkServer function to check for an active connection to the Internet. If one is found, Internet Explorer is called with the URL string as a command line option. If there is no Internet connection, the user is asked whether they want to save the URL to a list in order to open it later. The program maintains a list of saved URLs that can be accessed at any time from the "Tools" menu.

processVCard stores the card information in a temporary file and then calls the StoreVCard application with the file name as a parameter.

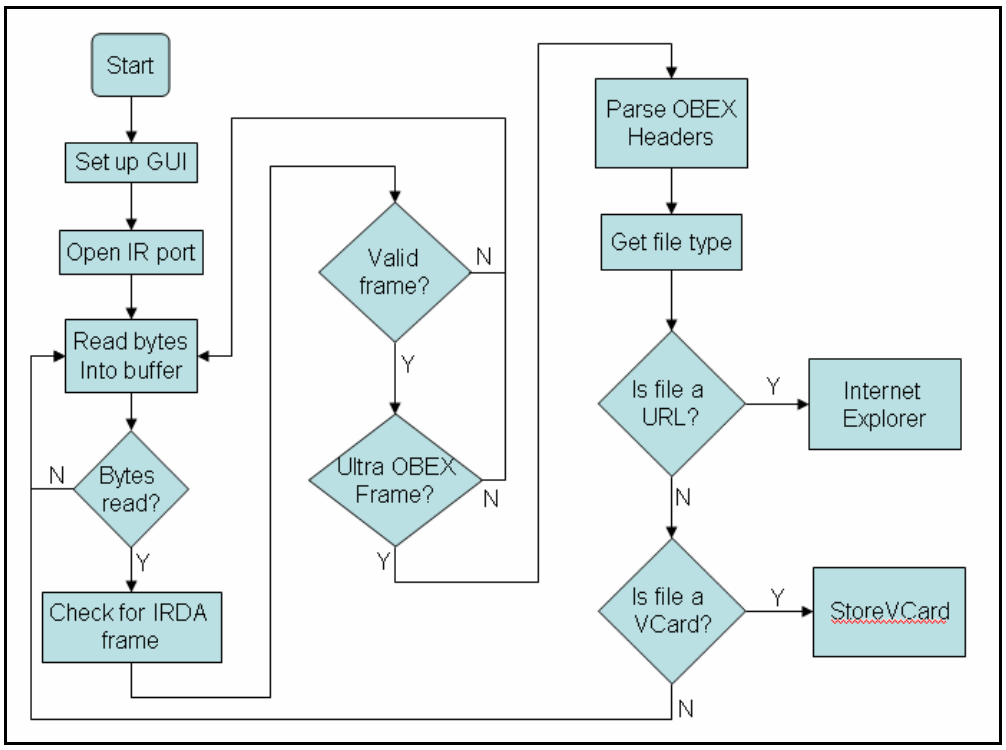

Figure 4.3 IrObexClient Flowchart

4.1.3.3 StoreVCard

The StoreVCard program has two important functions: processVCard, and saveContact. The processVCard function opens the temporary file that was created by IrObexClient and then parses the file to extract the business card data. The extracted data is then passed to saveContact.

The saveContact function uses Microsoft COM (Common Object Model) technology to create an instance of a Pocket Outlook application object. After logging on to the application, the function gets the current list of contacts in the database, and then adds a new one. Next the function sets the name, phone number, email, etc. of the new entry. Finally the new contact is saved and the COM instance is released.

4.1.3.4 Microsoft Pocket Internet Explorer

Microsoft Pocket Internet Explorer is a web browser built specifically for Microsoft PocketPC devices.

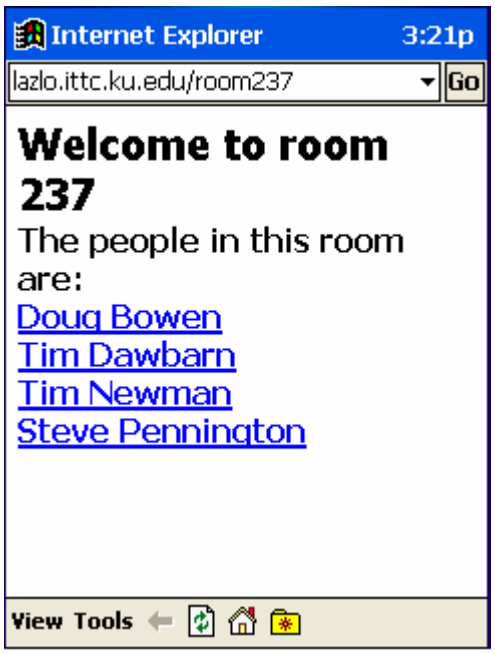

Figure 4.4: Pocket Internet Explorer

It is similar to its desktop equivalent except that it does not include support for scripting, java applets, or plugins. It does however support cookies, and SSL communications. Both of which are utilized by the applications in section 4.2.

4.1.4 Network

An 802.11b wireless network is used in this implementation. The iPAQ is equipped with a Linksys WPC11 802.11b adapter. The adapter receives an IP address, DNS addresses, and

a gateway address automatically via DHCP. 128 bit WEP encryption is used on this network. The client device is able to access resources via the Internet or local area network using this connection.

4.1.5 Server Device

The server device in this implementation is a desktop computer with an Intel 300 MHz Pentium II processor, 192 MB of ram, and a 6.2 GB hard drive. The server is equipped with a both Ethernet and 802.11b connections. The server can be reached from either interface.

4.1.6 Server Software

The server used in this implementation runs the Linux operating system with kernel version 2.4.19. The server runs Apache web server, mySQL database server, Heyu software, and several scripts written in PERL 5.0.

4.1.6.1 Apache

In its simplest usage case, the Apache web server listens for client requests on the network and responds by sending the requested file back to the client if it is available. The Apache server can also respond to requests for CGI (Common Gateway Interface) scripts by running the requested script and transmitting the output of the script back to the client. Lastly, the Apache web server has the ability to store and subsequently access small text files on the client device. These text files are called cookies and are useful for storing user

specific information between multiple visits to the same website. All of these facilities are used in the applications in section 4.2.

4.1.6.2 MySQL

This implementation uses a mySQL database to maintain user information as well as information about specific locations. A typical database is organized into several tables. Each of which have many rows of data. The database can be queried using SQL (Structures Query Language) to retrieve a single or multiple rows from one or several tables within the database. The database can be readily accessed by web clients via the Apache web server and the CGI scripts described in the next section.

4.1.6.3 CGI Scripts

The web server needs a way to make resources other than static files available to the users. In particular, the content returned needs to vary depending on the user's preferences and location. The use of CGI scripts along with the Apache web server makes this possible. The implementation works as follows. The client device requests a URL that includes a CGI script along with some arguments. Such a URL might look like this:

http://lazlo.ittc.ku.edu/cgi-bin/loc_auth.pl?pt=1234&ct=2345

In this example "loc_auth.pl" is the CGI script while "pt=1234" and "ct=2345" are the arguments. When the web server receives the request, it runs loc_auth.pl and passes it the argument string. At this point the script runs with access to as few or as many resources as the administrator permits, including the ability to run other programs on the server. The permission level of the CGI script is not at all related to the user making the request. The

output of the CGI script is then returned to the user in the form of a web page. In the above example, loc_auth.pl simply performs a mathematical operation on the number associated with "ct" and then compares it to the number associated with "pt." If the two match, an authenticated response is returned to the user. CGI scripts also have the ability to query the mySQL database and return the results to the user. This capability is crucial to providing more sophisticated web services that are customized to the individual user.

4.2 Applications

The following are applications that were developed using the technologies and architecture described in this document. All of the applications use the same hardware and software on the user end, while utilizing varying resources on the server end. An individual will be able to use any of the following applications without doing any configuration beyond the initial installation of the client software.

4.2.1 Location Information

Consider the following scenario. Jack is strolling aimlessly through the vastness of Megacon Corporation's world headquarters when he discovers the perfect conference room for his weekly downsizing meeting. To his dismay, there is no administrative assistant nearby to reserve the room for him. But then, in a moment of clarity, he pulls out his PDA, makes contact with a beacon in the room, and is instantly presented with a web-based sign out sheet for the conference room. He reserves the room for every Tuesday at 3:00 pm and continues on his way.

This is the simplest implementation of this system. The user walks in the room carrying a mobile device and makes contact with a beacon. IrObexClient receives a URL that points to a web page, and then passes it to Internet Explorer. Internet Explorer connects over the Internet to the specified server and requests the web page. The Apache web server software running on the server receives the request, and responds by transferring the requested HTML file back to the client. Internet Explorer displays the received web page which contains information pertinent to the room in which the user is currently standing.

The system administrator must do two things in order to provide this application. First he/she must create a web page that contains information that is related to the room in question. In the above application, this is a web-based sign out sheet. Next the administrator must program the web address into a beacon and place it in the room.

4.2.2 Environmental Automation

Sometimes the user does not need to know anything about their location in order to benefit from location based services. Consider the following scenario. Bill, being the fantastic host that he is, would like to set up the rooms in his house in such a way that they automatically configure themselves to the environmental preferences of any visitor. A visitor walks into a room while carrying a mobile device. After their device makes contact with the beacon, the lighting in the room is adjusted to the visitor's preference and the music in the room changes to a genre that he or she prefers.

This application works in the following manner. The administrator places a beacon in the room at an appropriate location. The beacon broadcasts a URL that points to a CGI script and has a parameter that relates to the room. For example:

http://mycompany.com/cgi-bin/enviro_auto.pl?room=148

When the user's mobile device makes contact with the beacon, it loads the URL in Internet Explorer. Internet Explorer requests the web resource from the Apache web server over the network. Upon receiving the request, the web server software runs a script and passes it the room number. The CGI script first retrieves a cookie from the client device which identifies the user. Next, the script looks up the environmental preferences of that user by querying a mySQL database. Then the script queries the mySQL database again using the room number to retrieve the list of controllable devices. After gathering the information from the database, the script sends the appropriate commands out over the X10 network to the necessary X10 modules. When the X10 modules receive the commands, they turn devices on or off as required.

This application is implemented in the following manner. The beacon is programmed with a URL that is specific to the location where it will be placed. For example:

http://www.billshouse.com/cgi-bin/env-auto.pl?room=3

The code for env-auto.pl is given in appendix B. Upon making contact with the beacon, IrObexClient passes the URL to Internet Explorer. Internet Explorer requests the web page from the Internet. When Apache receives the request it runs the env-auto.pl script

with the given argument. The first thing the script does is check for a cookie with user information on the client device. If no cookie is found, it responds with a web page asking the user to log in. After the user logs in with a name and password, a cookie is stored on the device so that this operation does not need to be repeated. At this point, the user has been identified by one of the previous two methods and the system can continue. Next, the script looks up the room in the mySQL database to see what devices can be controlled. The script then looks up the user in the mySQL database to see what his preferences are. If the user has preferences that correspond to X10 devices in the particular room, the script makes calls to the Heyu software to control those devices.

4.2.3 Location Authentication

An organization may want to offer a service only if they can know with some level of certainty that the user was at a specific location. Consider the following scenario. Megaplex theatres wants to entice people into visiting their theatres by offering a chance to win one million dollars (or seven movie tickets, whichever is more) to patrons who visit their theatres. They do not, however, want people who do not visit the theatre to be able to enter. The solution is to place beacons that broadcast a location-authenticated address of the sweepstakes entry form within the movie theatre complex. The valid address of the entry form changes with every beacon broadcast. No address can be accessed twice, and the next valid address cannot be easily guessed.

This application is essentially very similar to the system described in section 4.2.1. The main difference is that the beacon and the web server share a common secret key. The beacon generates a random number called a nonce, and then encrypts it using the secret key. Next the beacon transmits the location URL along with the plaintext nonce and the encrypted nonce as arguments. The broadcasted URL looks as follows:

http://lazlo.ittc.ku.edu/cgi-bin/loca_auth.pl?pt=1234&ct=5342

Where "pt" is the plaintext nonce, and "ct" is the encrypted nonce. When the web server receives the URL request, it decrypts the encrypted nonce using its secret key and then compares it to the plaintext nonce. If the two are equal, the web server has verified that this requested originated at a beacon. To protect against replay attacks, the web server can keep a running log of the last N nonces and deny any that are already in the list. A larger N means a longer time before a replay attack will succeed but more overhead on the web server. The chosen N must be significantly smaller than the nonce in order to ensure that valid requests are not denied.

The system administrator needs to do a few things in advance to set this application up. First the beacon needs to be programmed with the correct URL and the secret key. Next the CGI script loc_auth.pl needs to be stored on the web server in a place where Apache can access and run it. Finally loc_auth.pl must be configured with the secret key and the proper authenticated and denied responses.

4.2.4 Zero-Click Purchase

This is the most complicated but perhaps the most intriguing application in this paper. Consider the following scenario. John realizes that, at least for the foreseeable future, he will order the same drink every time he visits a Starbucks retailer. So he logs on to Starbucks' website, creates an account, and selects his favorite beverage, a grandé, non-fat, toffee-nut latte. The next time John visits a Starbucks, he pulls out his smart phone and makes contact with a beacon. His drink is automatically ordered, paid for, and placed onto the barista's orders list. After a minute, the barista calls John's name and he picks up his drink. John never waits in line and his beverage is always correct.

This application is implemented in the following manner. The beacon is programmed with a URL that is specific to the establishment where it will be placed. For example:

http://www.starbucks.com/cgi-bin/zeroclick.pl?storeid=148

The code for zeroclick.pl is given in appendix C. Upon making contact with the beacon, IrObexClient passes the URL to Internet Explorer. Internet Explorer requests the web page from the Internet. When Apache receives the request it runs the zeroclick.pl script with the given argument. The first thing the script does is check for a cookie with user information on the client device. If no cookie is found, it responds with a web page asking the user to log in. After the user logs in with a name and password, a cookie is stored on the device so that this operation does not need to be repeated. At this point, the user has been identified by one of the previous two methods and the system can continue. Next, the script looks up the user's favorite beverage and associated cost in the mySQL database.

The script then charges the user's account for the drink and places a new entry into the orders table in the mySQL database. The new entry contains the user's actual name, the description of the order, the location information (store ID) retrieved from the beacon, the time, and a status flag. The script then returns a web page to the user confirming their purchase.

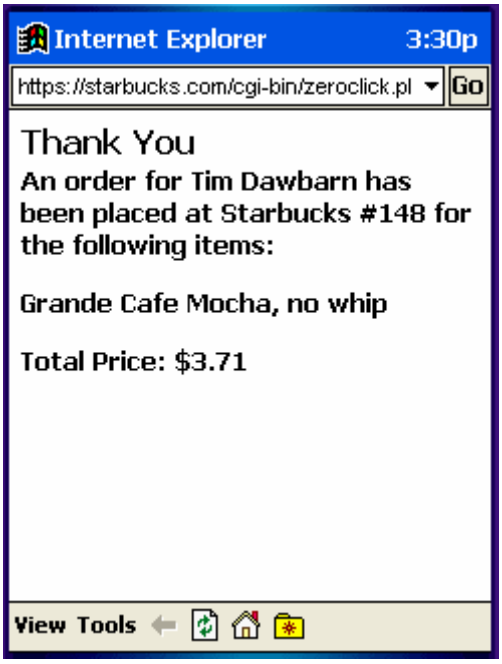

Figure 4.5: Purchase confirmation from zeroclick.pl

In order to fulfill the orders, a barista simply needs to navigate to the following address using a desktop computer or a web enabled register.

http://www.starbucks.com/cgi-bin/getorders.pl?storeid=148

The code for getorders.pl is given in appendix D. The getorders.pl script queries the orders table in the mySQL database for orders that both match the given store ID and have an active status flag. Some HTML formatting is applied to the resultant list, and it is

returned to the barista's browser. When an order has been filled, the barista can click the done button next to the order and its status is changed to "completed", which removes it from the list.

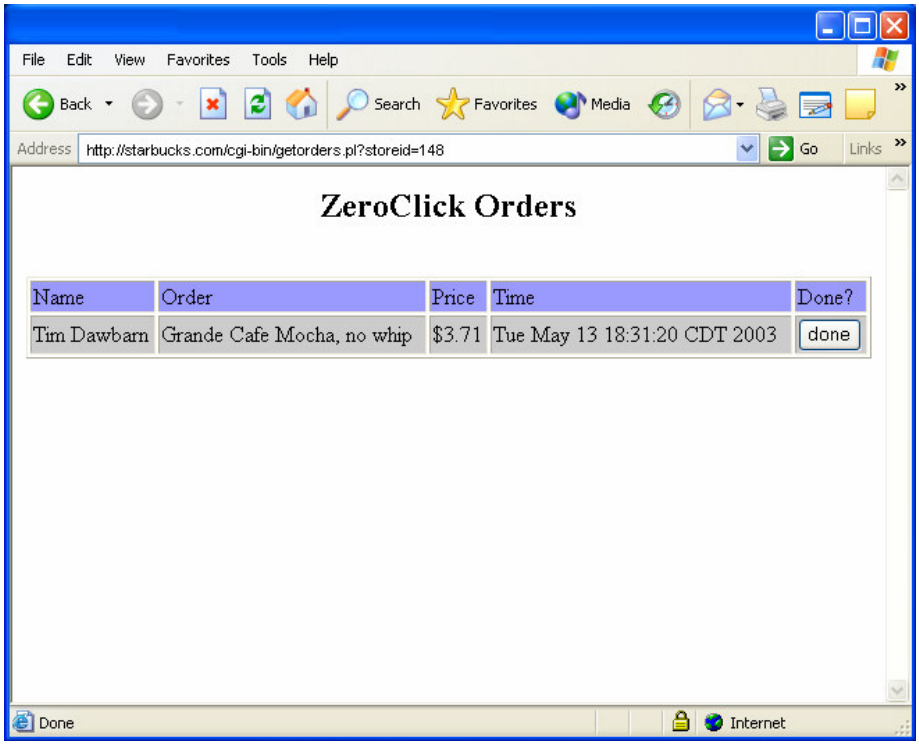

Figure 4.6: The results of the getorders.pl CGI script

4.2.5 Business Card

In some cases the user may not need any further information beyond what is broadcasted by the beacon. Distributing business cards is such an application. Consider the following scenario. Sally is representing her company's hot new product at an ultra high-tech trade show on the west coast. She anticipates talking to between seven and eight million potential clients but she only has a thousand business cards. No problem, she sets up a

beacon in her booth that automatically broadcasts her business card once every second to anyone with a mobile device that is in range. Everyone receives her card and no one has to go to the trouble of programming it into a device.

In this application, a beacon is first programmed with the business card information and then placed into the environment for users to access it. When a user's device makes contact with the beacon, IrObexClient receives the OBEX object being sent. IrObexClient identifies the object type as a vCard and saves the payload to a temporary file.

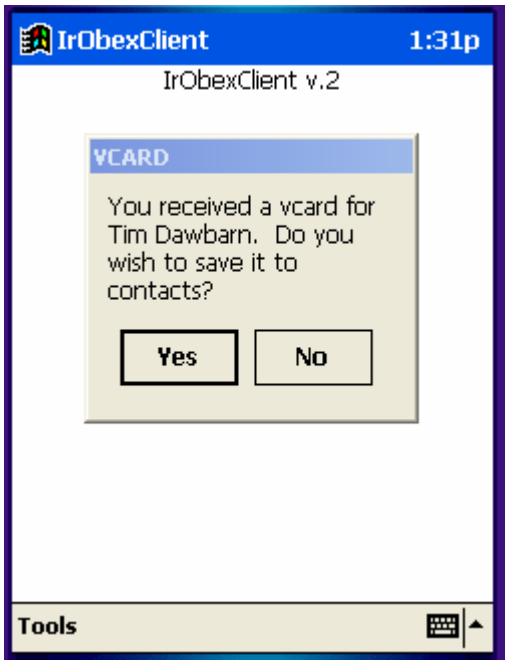

Figure 4.7: Receiving a business card

IrObexClient then calls StoreVCard with the temporary filename as an argument. StoreVCard opens the temporary file and stores the business card information to the contacts list.

Chapter 5

TESTING AND RESULTS

5.1 Hardware and Software Requirements

This is the specification of the system used for testing. Other systems may work but have not been tested.

- **Client Device**
  - **Compaq iPAQ 3150**
    - **200 MHz StrongARM processor**
    - **16 MB RAM**
    - **16 MB ROM**
    - **Linksys WPC11 802.11 Card**
  - **PocketPC OS Version 3.0**
  - **Microsoft Pocket Internet Explorer**
- **Server Device**
  - **Pentium II 300 MHz**
    - **192 MB RAM**
    - **6.2 GB Hard drive**
    - **Orinoco gold 802.11 card**
    - **3com Ethernet adapter**
  - **Linux 2.4.19**
  - **mySQL server version X**

- **Apache httpd version X**
- **Apache SSL version X**

- **Beacon**

5.2 Test Plan

After building the system and installing the software, a variety of tests were run to determine if the system is compliant with the specified architecture and if the applications behave as expected.

5.2.1 System Tests

Beacon range

The client device was held at multiple distances from an active beacon and the distances where successful receipts occurred were recorded. The client device is able to receive broadcasts from distances of six inches to about twelve feet.

Multiple beacons broadcasting the same URL

Multiple unsynchronized beacons broadcasting the same URL were placed in various locations within the same room. The behavior of the client device was recorded at several locations for each of the configurations. If the client device is within range of only one beacon, the device behaves normally. If the device is within the range of both beacons, the device will receive the correct URL provided that the two broadcasts do not overlap. If the broadcasts do overlap, the client device acts as though no URL was received.

Multiple beacons broadcasting different URLs

Multiple unsynchronized beacons broadcasting different URLs were placed in various locations within the same room. The behavior of the client device was recorded at several locations for each of the configurations. If the client device is within range of only one beacon, the device behaves normally. If the device is within the range of both beacons, the device will alternate receiving the two different URLs provided that the two broadcasts do not overlap. If the broadcasts do overlap, the client device acts as though no URL was received.

No Internet connection

The network card was removed from a client device before making contact with a beacon. The behavior of the client device was observed. Upon receiving the broadcast from the beacon, the client software informs the user that a URL was received but the resource is not available. The software then asks the user whether or not to save the URL for later opening.

Stored addresses functionality

After replacing the network card and configuring the network connection, an attempt was made to load a previously stored web address using the client software. The web site was successfully loaded in Internet Explorer.

5.2.2 Location Information

A beacon was programmed with a URL that pointed to a web page with information specific to room 213 at Nichols Hall. The beacon was then activated and placed on a shelf in the room. When the user carrying a client device made contact with the beacon, the device received the URL and was successful in loading the room 213 web page in Internet Explorer.

5.2.3 Location Authentication

CGI script retreival

A beacon was programmed with a URL that pointed to a CGI script. The beacon was then activated and placed on a shelf in a room. When the user carrying a client device made contact with the beacon, the device received the URL and was successful in loading the page. The CGI script executed and the results were displayed.

Nonce encryption

The plaintext nonce was manually encrypted using the secret key and the encryption algorithm used by the system. The obtained cipher text was then compared to that which was broadcasted by the beacon. The two were a match

Replay attacks

A client device was placed within range of a beacon broadcasting the location authentication format. After successfully loading the URL, a second attempt was made using the same address. While the first attempt was successfully authorized, the second was denied.

5.2.4 Zero-Click Purchase

SSL communications

The client device was used to access a website using SSL. The data packets sent to the client device were viewed using tcpdump. There was no clear text observed in the packets.

```
16:35:18.079804 xxx.xxx.xxx.xxx.3528 > xxx.xxx.xxx.xxx.https: P 1:121(120) ack 1
win 17520 (DF)
0x0000   4500 00a0 4dff 4000 6b06 3e19 81ed 7b30      E...M.@.k.>...{0
0x0010   411e 4504 0dc8 01bb cef1 0ebb b045 3ab1      A.E.........E:.
0x0020   5018 4470 d4df 0000 1603 0100 7301 0000      P.Dp........s...
0x0030   6f03 0100 008c 089b 76eb 7839 be08 9a28      o.......v.x9...(
0x0040   fdc3 30b6 fd6b 2ba8 2751 8da4 b67a 6c9f      ..0..k+.'Q...zl.
0x0050   a6fa 8220 1c71 18c2 3959 df0d 8b11 6411      .....q..9Y....d.
0x0060   11ae 41a8 4ac1 d957 4789 5992 a1b4 a23e      ..A.J..WG.Y....>
0x0070   b577 4710 0028 0039 0038 0035 0033 0032      .wG..(.9.8.5.3.2
0x0080   0004 0005 002f 0016 0013 feff 000a 0015      ...../.........
0x0090   0012 fefe 0009 0064 0062 0003 0006 0100      .......d.b......
```

Figure 5.1: Encrypted login data packet captured with tcpdump

Cookies

The zeroclick.pl script requires a cookie which identifies the user to be stored on the client device. The script redirects first time users to a one-time login screen which stores this cookie on the client device. After the completing a login successfully, the cookies directory on the client device was examined. The cookie file was present and the information it contained was consistent with the requirements of the application.

The system requires, for security reasons, that cookies only be retrieved from the client devices over SSL. A CGI script was written to attempt to retrieve the user identification cookie over a non-secure channel. The script was unsuccessful in retrieving the cookie.

5.2.5 Business Card

Business card receipt

A client device was placed within range of a beacon broadcasting business cards. The behavior of the client device was observed. If the auto save option was enabled, the IrObexClient software did not have any visible reaction, but upon examining the contacts database, one could see a new entry. If the auto save option was not enabled, the user is immediately prompted after making contact with the beacon.

Temporary file storage

After making contact with a beacon that is broadcasting business cards, the contents of the temporary file were examined. All of the information in the file was consistent with the requirements of the application and the information stored in the beacon.

Contacts database modification

The new entry in the contacts database was examined to verify that the contents matched the information broadcasted by the beacon. All of the fields matched correctly.

5.2.6 Environmental Automation

Communication with Heyu program

The CGI script was modified to call the heyu program using the –v (verbose) option and redirecting the output to a file. After running the script, the file was reviewed and the appropriate log was found.

```
Version:1.33
Using the config file /etc/x10.conf
Trying to lock (/var/lock/LCK..heyu.write)
/var/lock/LCK..heyu.write is locked
Sending house code  : 6
Sending unit code  : 5
Sending header 4 : 65
xwrite() called, count=2
xread() called, count=1, timeout = 10
xread() returning 1 byte(s). The first is 69
xwrite() called, count=1
xread() called, count=1, timeout = 10
xread() returning 1 byte(s). The first is 55
Sending function 6 : 62
xwrite() called, count=2
xread() called, count=1, timeout = 10
xread() returning 1 byte(s). The first is 68
xwrite() called, count=1
xread() called, count=1, timeout = 10
xread() returning 1 byte(s). The first is 55
all ok with transmit
```

Figure 5.2: Heyu log file

Communication with X10 modules

The CGI script was requested and a light connected to an X10 module was observed. The light turned on indicating that X10 communications are working correctly.

5.3 Reliability

There are several possible points of failure for this system. It is worth examining these modes of failure and how the system will behave.

5.3.1 Network failure

There two ways in which network failure can affect the applications in this implementation. The first is if the server loses its network connection. In this scenario, one of two things will happen. If server connection goes down before making contact with the beacon, IrObexClient will detect that the server is unavailable and prompt the user to save the URL for later access. If the server connection is interrupted after the client makes contact with the beacon, Internet Explorer will execute but will not be able to load the requested resource. At this point, it will be up to the user to reload the web page when the server comes back online.

The second case is if the client device loses its network connection. In this case the timing of the interruption also changes the behavior. If client connection goes down before making contact with the beacon, IrObexClient will detect that the network is unavailable and prompt the user to save the URL for later access. If the client connection is interrupted after the client makes contact with the beacon, Internet Explorer will execute but will not be able to load the requested resource. At this point, it will be up to the user to reload the web page when the server comes back online.

In both cases there is the possibility that a client request might make it to the server but the response might no make back to the client. In purely informational applications this behavior is benign, but in the case of the zeroclick purchasing application, the behavior could be disastrous. Some care must be taken when programming the server-side applications that this scenario is accounted for.

5.3.2 Server Failure

This scenario is essentially the same as if the server loses its network connection. The behavior of the client device again depends on the timing of the server failure.

5.3.3 Client Device Failure

If the client fails in such a way that it is unable to access the network, the behavior is the same as in the network failure section. If the infrared port is unavailable, an error message is displayed and the client software exits. All other errors are handled by the client devices operating system.

Figure 5.3: Infrared port error screen

5.3.4 Beacon Failure

There are several ways in which the beacon could fail. First, if the battery is disconnected or the beacon is severely damaged, it may quit broadcasting completely. If this happens, the client devices will obviously not be able to receive any location information. The beacon is built with a red LED that flashes with every broadcast. If the LED is not flashing at regular intervals, the beacon should be checked.

The beacon could fail due to corrupted memory. If this happens, it is unlikely that it would continue to broadcast at all. If it was able to continue broadcasting, it is even more unlikely that the data transmitted would be valid IRDA frames

5.4 Scalability

Scalability in a location-based information system is important. This system is essentially as scalable as the Internet itself. For an individual organization, the number of unique beacons that can be used is only limited by the number of web pages the company's web server can provide. If pages are generated dynamically using a script, this number is nearly infinite. The only limitation of the beacons is that due to memory constraints, the maximum URL length is 236 characters. Considering all of the valid characters, this provides more than 36^{236} possible URLs.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusions

The goals of this project were successfully met. The system is able to provide users with location information with a level of resolution that is smaller than a single room. The applications created using this system utilize the additional location resolution to provide users with valuable services. In addition, the system was implemented at low cost and requires very little adaptation by the user.

Some important lessons were learned in the development of this project. The first is that the logical location broadcasting scheme works very well. It is a different approach but provides significant benefits. Using logical location data provides a lot of location detail without requiring a scientific setup procedure. This scheme is also very scalable since almost an infinite number of unique URLs can be used.

Another lesson learned is that Beacons can be produced from relatively cheap off-the-shelf components. The cost can also be reduced further by implementing functions that are traditionally done by hardware in software.

Last but not least is that not all IRDA compatible devices implement the same parts of the IRDA protocol specification. It became evident after some time that the iPAQ and its

PocketPC 3.0 operating system do not support the ULTRA (connectionless) portion of the IRDA specification. Even if it did, it would not matter because there is no API exposed for handling raw IRDA packets on the PPC; only socket communications. Although this matter required a significant and unexpected effort, much was learned about IRDA communications protocols in the process.

6.2 Future Work

It seems that some of the newer mobile devices may have a more mature IRDA stack that might already include some of the functionality implemented in IrObexClient. It would be beneficial to try to develop a beacon protocol that works without installing client software on these newer devices.

The encryption algorithm used in the location authentication application is trivial. For a more robust application, a real-world encryption scheme such as DES or RSA public key should be used.

The system could be improved by developing solutions for other mobile platforms. Specifically, system which works with mobile phones and PalmOS devices would be a nice improvement.

The beacon URLs are currently set by directly programming the microcontroller. A user friendly desktop application is needed for configuring beacons.

More applications that use this system could be developed. The applications described in this document are only a small sampling of the possibilities with this architecture.

BIBLIOGRAPHY

[1] *An Approach to Providing a Seamless End-User Experience for Location-Aware Applications* Sastry Duri, Alan Cole, Jonathan Munson, Jim Christensen IBM Thomas J Watson Research Center 30 Saw Mill River Rd., Hawthorne, NY 10532 {sastry, colea, jpmunson, ibmjim}@us.ibm.com, In **Proceedings of the first international workshop on Mobile commerce** July 2001

[2] Infrared Data Association. http://www.irda.org/about/index.asp

[3] IrOBEX specification, Infrared Data Association. http://www.irda.org

[4] The Apache HTTP Server Project, http://httpd.apache.org/

[5] MySQL Database Server, http://www.mysql.com/products/mysql/index.html

[6] MySQL & mSQL, Randy Jay Yarger, George Reese & Tim King, Orielly Press 1999, page 7

[7] X10 Knowlegebase, http://www.x10.com/support/tech_index.htm

[8] The Cricket Location-Support System, Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan, MIT Laboratory for Computer Science, Cambridge, MA 02139{bodhi, achakra, hari} @lcs.mit.edu, In **Proceedings of the sixth annual international conference on Mobile computing and networking** August 2000

[9] Location Based Services in a Wireless WAN Using Cellular Digital Packet Data (CDPD); Rittwik Jana, Theodore Johnson, S. Muthukrishnan, AT&T Research Labs, 180 Park Avenue Florham Park, NJ (07932), USA. {rjana,johnsont,muthu}@research.att.com; Andrea Vitaletti Dipartimento di Informatica Sistemistica Universita di Roma, 00198-Roma, Italia vitale@dis.uniroma1.it

[10] Composable ad hoc location-based services for heterogeneous mobile clients Todd D. Hodes and Randy H. Katz Computer Science Division, University of California, Berkeley, CA, 94720, USA

[11] Using Wireless Ethernet for Localization Andrew M. Ladd, Kostas E. Bekris, Guillaume Marceau, Algis Rudys, Dan S. Wallach, and Lydia E. Kavraki, Department of Computer Science, Rice University, Houston TX, 77005

[12] Heyu software, http://heyu.tanj.com/heyu/

BIBLIOGRAPHY

[13] vCardspecification, Internet Mail Consortium, http://www.imc.org/pdi/vcardoverview.html