Bus Catcher: a Context Sensitive Prototype System for Public Transportation Users

Michela Bertolotto, Gregory O'Hare, Robin Strahan, Ailish Brophy, Alan Martin, Eoin McLoughlin
Department of Computer Science,
University College Dublin,
Ireland.
michela.bertolotto@ucd.ie, gregory.ohare@ucd.ie, robin.strahan@ucd.ie, alan.martin@ucd.ie, eoin.a.mcloughlin@ucd.ie

Abstract

In this paper we describe the architectural and functional characteristics of Bus Catcher, a context sensitive prototype system for public transportation users. Bus Catcher assists mobile users in planning their bus rides by providing timely and accurate information about current bus locations and estimated arrival times. A complete report on the implementation together with a preliminary evaluation of the system is provided in this paper.

1. Introduction

This paper introduces the *Bus Catcher* system. *Bus Catcher* represents a *context sensitive service* targeted at the transportation sector. With escalating levels of mobile device penetration, increasingly the citizen will be equipped with some Personal Digital Assistant (PDA). PDA ownership in Europe is set to treble in the 5-year period 1998-2003 (IDC Western Smart handheld Devices Review, 1998-2003). PDA sales lie just below the mass-market penetration threshold of 10%.

Bus Catcher seeks to use such devices as a conduit for communicating public transportation information relative to the needs of the individual traveler. Mobile location services worldwide are set to increase from 2 Million connections in 2001 to 560 million in 2006 [27] and indications are such that services that are personalized and easy to use will maximize their revenue streams. *Bus Catcher* is representative of this new generation of user friendly, personalized and mobile context sensitive services. It embraces ambient computing principles, enabling less technologically skilled users to have casual and tailored access to information. In the case of *Bus Catcher* this information pertains to general issues of transportation and movement of the citizen.

In particular, bus networks can be confusing. To a traveler in an unknown city, information about which bus routes lead to which destinations can be scarce. Even for people who know the city, the sheer number of bus routes can be daunting; for instance Dublin Bus [21] serves over 150 routes. Added to this is the inaccuracy inherent in the timetables; any number of factors can cause disruption, making a timetable little more than a rough guide to the actual time that a bus will arrive at. What is required is a system with up to the minute data about bus stops, routes and timetables. Handheld PDA's can be a valuable aid in this situation, as they are capable of providing real time information on the move. As they are mobile and travel with the user, she would be capable of accessing the data anywhere, rather than just at the bus stop.

The idea to realize the *Bus Catcher* system was inspired by typical real life situations such as those depicted in Figure 1 and by the announcement in May 2001 that Dublin Bus was planning to install GPS receivers on all their buses to improve management and monitoring of their service. This would allow obtaining timely and accurate information about current bus locations and estimated arrival times. This capability differentiates *Bus Catcher* from other mobile context sensitive services targeted at the transportation sector.

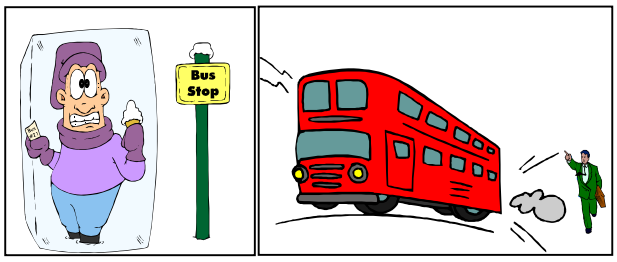

Figure 1. Typical bus users scenarios

The remainder of this paper is organized as follows. Section 2 provides a research context within which *Bus Catcher* was developed. Section 3 discusses issues related to the high-level design of the system including major objectives and architectural characteristics. Section 4 reports on the implementation details while section 5 is dedicated to the system evaluation. Finally, Section 6 presents some concluding remarks.

2. Related research

Bus Catcher is broadly situated in the arena of context sensitive systems. The term *context* refers to any information that can be used to characterize the situation of an entity, where an entity is either a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [3].

Many aspects of the physical and conceptual environment can be included in the notion of context. The user or system's location in space and time is an obvious one. Personal information about a user, such as their preferences, their knowledge and their history of previous interaction with the system could also be considered as context.

The use of context is increasingly important in the fields of handheld and ubiquitous computing, where the user's context changes rapidly. Whilst it would be possible to require users to explicitly provide contextual parameters, most systems accrue such data in an unobtrusive and implicit manner by listening to user system interaction and their general behaviour.

User location and orientation for example, could be detected using one of several position sensing technologies. Specific instances include Global Positioning System (GPS), Galileo, Cellpoint, and True Position. Generally three broad approaches exist: those of satellite based, base cell triangulation or hybrid approaches.

A rich variety of applications have been developed to demonstrate the usefulness of context-awareness in wireless computing. Two seminal systems are those of Cyberguide and Guide.

Cyberguide provides a mobile context-aware tour guide [1]. The contexts Cyberguide is aware of are the location and direction of the user. The application has four distinct parts: Map, Information, Positioning and Communications. The map component displays a map of the user's immediate location and enables the user to navigate the environment. It can be viewed at varying levels of detail and scrolled around. The user's location and direction is displayed as well as locations of interest.

The information component displays information about the area and its objects available to the user. The positioning component in turn provides information on the user's location and direction while the communication component allows the user to communicate with other Cyberguide users. Two versions were developed, an indoor based on Infra Red technologies and the other utilizing GPS.

The Guide project was carried out by the Distributed Multimedia Research Group of Lancaster University. The major aim of the project was to develop a context-aware mobile multi-media visitor guide and to evaluate its use in the city of Lancaster [2]. The contexts that it was aware of were classed into two types, personal and environmental. Environmental contexts included the current time and opening hours of attractions. Personal contexts covered the user's profile (interests and age, etc), current location and their native language.

Guide allowed the user to explore the city anyway they wanted to, without sticking to fixed tours. The user's location was plotted on a map (determined by GPS), which allowed the user to navigate easily. While walking about the city, the user could make queries about the area they were in, nearby attractions, points of interest etc. They could request the latest weather forecast or ask Guide to create a personalized tour for them based on their current location. The user was also able to communicate with other Guide users (to arrange to meet up for example), to book accommodation and to receive alerts about changes in opening times of attractions. The system architecture adopted was that of a client-server model as this supported dynamic real-time update of the content presented to the user. In addition it enabled content stored on the device to be minimized and thus the device could be smaller and less expensive.

In addition to these generic context sensitive services other specific applications have been developed that address the transportation sector.

KMap [22] from JShape Software, is a complete raster image graphics program. It was written in J2ME for the Palm as a demonstration to show how to pan and zoom on a map image. It retrieved its map data from a database that was loaded on to the Palm. This differs from *Bus Catcher* in that *Bus Catcher*'s map image is dynamic and comes from the server. Because it was an experimental project it is very basic.

Train Schedule [23] constitutes a simple J2ME application developed for the Palm OS platform. It allows the users an easy way to view commuter train schedules. Although this application was written for the Chicago Metro train system, it is possible to use it for other train or bus systems by changing the database. Train Schedule works by allowing the user to choose which line they wish to catch a train on. Then by choosing a start and destination station on that line along with the time and day they wish to travel, the program will search the database

loaded onto the Palm for a group of the most suitable trains that the user can get.

In contrast to *Bus Catcher*, Train Schedule works from a stored static database. So the program will only be correct if its database is correct. If the schedules change the database will have to be updated and uploaded to the Palm. It is also not possible to run this program, even though it is written in Java, on a PDA operating an OS other than Palm OS because of its reliance on the Palm Database file.

Bus Catcher gets its schedule from the server so the user will get the most up-to-date information (provided its administrator updates the server) and it does not rely on any platform specific database. Train Schedule does not use GPS or maps to show the user how to get to a train station or other attraction. There is no 'point & query' akin to that offered by *Bus Catcher*.

Train Brain, on the other hand, is a Java applet that simulates the accelerated or real time animation of a transit system, such as a train or bus system. It can display pop-up schedules for various lines or routes, and provides 'point & click' station information.

Train Brain has been used as the basis for a context-aware system for the Virginia Railway Express (VRE). The context of interest was the trains' locations. Train Brain accesses a preexisting Train Information Provider or TRIP system that informs users at a VRE station of the arrival of the next train. The Train Brain applet accesses the TRIP server. When any delays or disruptions to trains are logged on the TRIP server the applet will calculate the train's position according to this information and display it on the VRE website. To find out about a particular train the user moves the mouse over that train and a pop up box informs the user about that train and any associated delay status.

The major disadvantage of this system is that it is web enabled and only accessible via desktop machines utilizing standard HTML. To be of real use to a commuter, the applet would have to be written so that it could be viewed on a WAP or web-enabled phone or PDA.

Numerous commercial systems have been successfully developed some of which are beginning to develop client bases. These include TomTom [24], which provides context aware software for PDAs, such as city maps, route planners and restaurant and attraction guides. CitiKey [25] provides electronic reference guides of capital cities hosted on PDAs. Tourists can rent the system from the local tourist office. Citikey does not feature any context awareness or personalization.

Several recent systems have deployed Multi-Agent Systems such as Impulse, ComMotion and Ad-Me. The Impulse [20] project provides personalized location-based information through the use of agent communication. A User Agent residing on a hand-held device creates a user profile and builds queries for the Wherehoo server and Provider Agents. The results of the queries are displayed to the user by the User Agents in the form of URLs. ComMotion [12, 26] uses a location-learning agent to observe the locations frequently visited by the user via a GPS receiver. It uses both a speech and graphical user interface, which assist in providing location based information, displaying maps and controlling administrative functions. The Ad-me project [7] is a mobile tourist guide that proactively delivers advertisements to users based upon perceived individual user needs together with their location. It adopts a Multi-Agent System (MAS) design philosophy and strives for maximum content diffusion across HTML, WML, HDML and iMode formats.

3. High-level *Bus Catcher* design

3.1. System objectives

The *Bus Catcher* system was designed on the basis of the following requirements:

- Modularity: each component should be developed independently and be of the plug & play type;
- Portability: the application should be designed so that it will run on any Java enabled device regardless of its type, its OS and its available resources;
- Extensibility: the system is targeted to bus users but should allow for easy integration of other type of transportation services information as well as tourist information etc.
- Ease of use: in order to be an effective decision support tool, the system should facilitate human-computer interaction.

The main objectives we considered in the realization of the *Bus Catcher* System (from the point of view of the functionality offered) are:

1. To provide an interface that is intuitive to use. The interface should be of the 'point & click' variety, as the only input modality is that of stylus on the touch sensitive display.
2. To display accurate and timely timetable information for all bus routes.
3. To plot in real time both the user's and bus's location on a map-based display using GPS as the localization technology.

In addition, the information provided should be context sensitive. For instance, maps should only provide data on bus services close to where the user is located, and the timetable information should be related to the real-time location of the buses, taking into account factors such as

the time of day and weather conditions. Further context-sensitive features might include basing data retrieved upon previous requests, if a user frequently requests a particular route, the timetable for that route could come up by default.

Other possible functionality includes: calculation of the correct fare for a specified journey, display of major tourist attractions, possibility for the users to select the language that they wish this interface to be presented in (English, Irish, French, Italian, etc).

3.2. System architecture

The *Bus Catcher* system relies on a multi-tier architecture comprising four main layers, namely the *Client Layer*, the *Application Server Layer*, the *GIS Layer* and the *Database Layer* (see Figure 2).

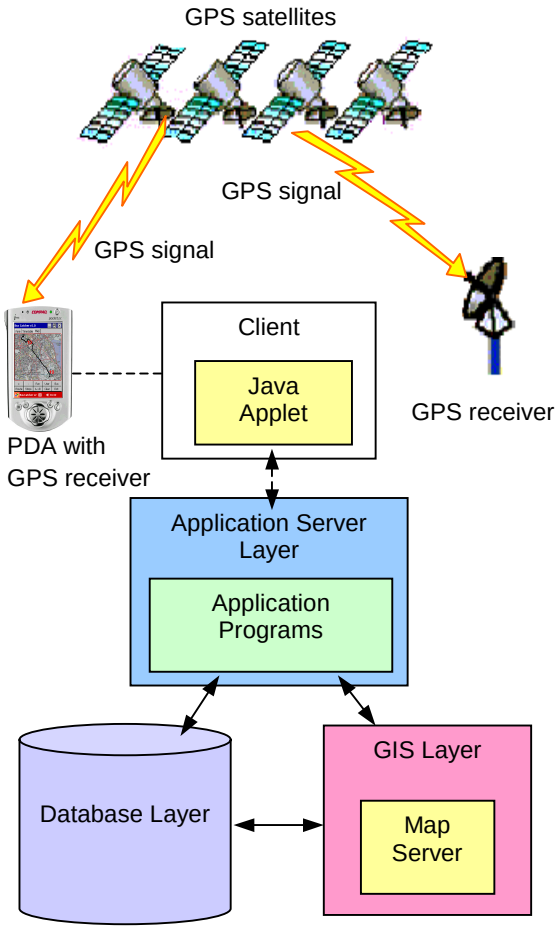

Figure 2. *Bus Catcher* architecture.

In the following we briefly describe each component. The client layer comprises a lightweight client machine running a Java Virtual Machine (JVM). It functions as a thin client with most of the computation handled by the remote server. We strive for a thin client and thick server model. The application is executed on the client using a standard Java applet. All communications between the client layer and the database and GIS layers are conducted through the application server layer. The applet communicates with the application server using the existing TCP networking protocol.

Information is downloaded to the PDA, as it is needed. The client sends requests to the server, which in turn will obtain data from either the map server (residing on the GIS layer) or the database or both. It will then return this information to the client for display to the user. The client also receives data from GPS satellites through the use of a GPS receiver.

The application server layer has been designed according to the following considerations:

1. The server needs to be able to handle many clients simultaneously and efficiently. This is achieved by making the server multithreaded. In addition, each thread should terminate appropriately and use as little memory as possible.
2. Due to the inherently unreliable nature of mobile communications, error control is vital. Timeout handling is required on both sides to avoid waste of CPU time and user's money. If an error occurs, the system should attempt to either retry or gracefully disconnect as appropriate.
3. The protocol should be kept simple and unambiguous. This ensures efficient use of bandwidth and reduces chance of a protocol error occurring.
4. The server should be integrated with the GIS and database.

A dedicated server model has been utilized for the application server. A dedicated server was chosen over a web server because it is simpler, easier to implement, and allows more control over the client-server communication.

A thin client was chosen over a thick client not only to maximise speed but also to provide a higher degree of abstraction and modularity. Indeed, the Database and GIS do not need to be concerned with operations of the client. The Client-Server protocol can be changed without consent of the Database or GIS, increasing programmability and simplicity. The Server can also anticipate what the client needs and request data from the GIS or Database in advance.

Modularity is one of the main characteristics of the *Bus Catcher* system architecture. Each component has been developed independently and can be easily modified without affecting the others. For example, the system is completely independent of the particular GIS system used in the GIS layer. Communications occur via standard protocols and by exchanging data in standard formats.

The GIS layer includes a map server whereby any program can gain access to a range of maps stored remotely, through an API (Application Programming Interface). This component communicates with the database to allow access to, and mapping of, bus related data stored there, such as the bus stops and their locations.

The database layer stores all the information about the bus routes, bus stops, and timetables. SQL statements are issued to retrieve from the database the information to be displayed on the client site. For example, when the user requests to see a given timetable, an event is associated with that particular request and the corresponding SQL statement is called. All prepared statements including the most frequently requested queries are also stored in this layer. These include: route queries, nearest stop queries, arrival time queries, etc. Estimated time of arrival of the next bus is calculated by taking into account parameters such as rush hours, traffic loads of specified areas, weather conditions, etc.

4. Implementation

In this section we describe the implementation of the *Bus Catcher* system. Technologies employed include:

- Java for the application server layer,
- MapInfo for the GIS layer,
- Visual Basic for the Map Server (for its full integration with MapInfo through OLE automation and window reparenting),
- MySQL for the underlying database system,
- ODBC API to access the database with application programs,
- JDBC for access to the Map Server from the application server.

Connections between Java application programs and the Map Server take place through a TCP socket connection across which requests and maps are sent. The TCP protocol was chosen over HTTP as the TCP packets do not have to correspond to HTTP (GET and POST requests). This reduces protocol overhead since the data does not have to be wrapped in HTTP. The server will consequently be smaller and more efficient as the need for servlets does not exist.

TCP was chosen over UDP as the network layer protocol of choice as it is simpler to program (Socket I/O similar to File I/O). It provides a reliable error free channel between the client device and the server with automatic flow control. The simplicity and reliability of TCP far outweighs the modest speed increase provided by UDP.

A preliminary version of *Bus Catcher*, called *Bus Catcher Lite*, was developed to work on the Palm. Due to device limitations, this version only provides textual information related to timetables and fare calculations. These calculations are performed in the Application Server Layer, and take into account an number of different factors, including parameters such as rush hours, traffic loads of specified areas, weather conditions, etc.

Map display resulted in long lead times primarily due to the Palm processor, a mere 33Mhz compared to 203Mhz for the Compaq Ipaq. In addition heap space restrictions within the KVM meant that maps larger than postage stamps were unable to be displayed. For these reasons a restricted functionality was provided for the Palm and the full functionality of *Bus Catcher* was developed on the Compaq Ipaq (Figure 3 depicts the *Bus Catcher Lite* interface).

The Compaq Ipaq connects to the server over an Internet connection provided by the Nokia Card (v2) modem. Although this modem supports GPRS, the mobile network, at the time of testing did not. This meant that the maximum connection speed available was a modest 9600bps. Global positioning was retrieved using a Garmin GPS receiver attached to the serial port of the device, communicating using standard NMEA sentences.

As the fare and timetable parts had been successfully implemented in the *Bus Catcher Lite* version, they were ported to the full version with no changes required.

Within *Bus Catcher* the key additional functionality is a map-based interface. Based around this are several key components of the GUI: map display, standard zoom in/out and pan operations. Maps are geocoded on the map server component of the map layer and are sent to the client through the application server layer. Prior to this, information about the actual PDA display size must be sent from the client to the server. This allows the map server to send an image customized to a particular PDA's display size.

A map centring function was provided which involves requesting a map centred at a particular GPS co-ordinate. The GPS co-ordinate is calculated from the pixel on the current map image that the user clicks on. The zooming in and out is achieved by setting a zoom level. Seven levels of zoom were provided. If the user wishes to pan the map, a new map image is generated at the same zoom level as the current one, just centred at new GPS co-ordinates.

Once the basic manipulation of the map was completed, bus route overlays (Figure 5) needed to be delivered from user-designated start and end points. This was achieved via series of buttons on the map frame.

These bus routes could be annotated with bus stage information. When the user clicks on the appropriate button, a dialog box appears that asks the user to select what route and direction they wish to plot. The user can select as many routes or stops then want to plot by clicking the 'Plot' button and then click the 'Done' button when completed. The GUI then uses a function from the Client

class to request a map image with the requested items drawn on it. Again the Client functions abstract from the GUI how this information is retrieved from the map server. The 'Clear' button removes any previous selected routes.

User location was similarly overlaid on the map-based interface (Figure 4). The user's location was determined using a GPS receiver. The GPS receiver outputs its data in NMEA GPGGA format. This is a worldwide standard for Global Positioning System Fix Data. It contains information such as time, latitude, longitude, altitude, number of satellites used to generate position and error estimation.

A thread is created when the map feature is first constructed that calls the ImageCanvas paint method every 10 seconds. The paint method was further updated to read the latitude and longitude received from a GPS receiver. This latitude and longitude is then checked to see if the location is viewable on the current map canvas. If they are viewable, they are coerced into pixels and an image that represents the user is drawn at that pixel location.

A similar approach was used to plot the bus location. Except that the co-ordinates of the bus are simulated and come from a text file on the server. However, such live GPS coordinates can be received from the GPS receiver on Dublin buses.

Finally an A to B function was implemented. This feature allows the user to select a start and end location on the map and ask which route or combination of routes will allow them to make that journey.

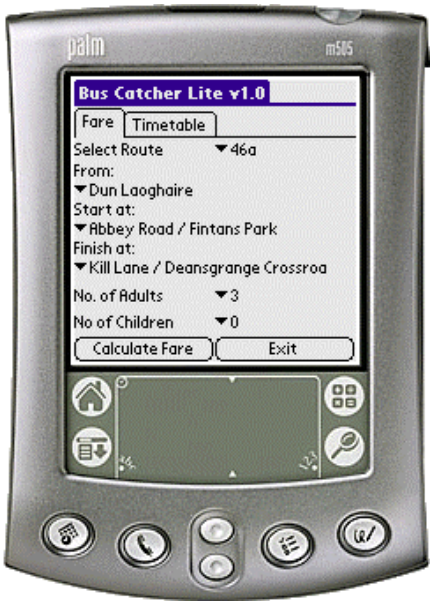

Figure 3. Bus Catcher Lite

Figure 4. Display of user and bus location

When the user selects this feature, they are prompted by a dialog to select the starting point of their journey. Similar to when a user requests to manipulate a map image, the pixel that the user clicks on is converted into a GPS co-ordinate. They can of course zoom and pan the map before selecting any point.

Figure 5. Route overlay

Once the first point is chosen they are prompted to select their second point. Once the second point has been

selected, the two co-ordinates are passed to the appropriate method in the Client class. This client will then in turn send this to the server that will in turn query the database. The information about which route or routes are suitable, along with where to board and get off on the route, are then extracted from the database. These results are returned to the client and then passed to the GUI to display in a dialog box. The dialog allows the user to view which routes they can take and to cycle through them using the 'Prev' and 'Next' buttons. The dialog also allows the user to select up to three routes from the results given and to plot them on the map.

All the code for *Bus Catcher* and *Bus Catcher Lite* was written in Java, and therefore not platform specific. As a consequence little, if any, changes are required in order to run *Bus Catcher* on other platforms. However, device-specific constraints may restrict portability. Features such as processor speed, memory and display size vary from mobile device to mobile device. Ultimately *Bus Catcher* will detect device-specific features and dynamically adapt the GUI format accordingly.

5. System evaluation

In order to evaluate the *Bus Catcher* system user trials were undertaken. These trials were conducted in the month of April 2002 over a two-day period. The sample size was statistically significant. A sample of subjects, roughly thirty-five members of the public, were asked to use the Bus Catcher system briefly in order to achieve familiarization. Thereafter they were asked to complete a questionnaire.

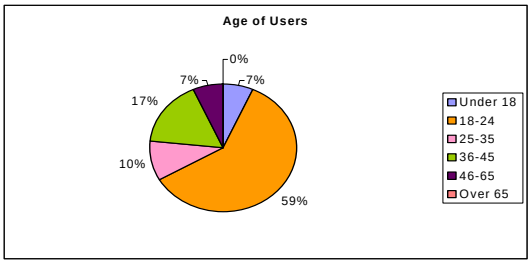

Figure 6: User demographics

Two contrasting sites were chosen for the trial. The first was a busy boarding point on the Belfield campus of University College Dublin, while the second was a busy Dublin city centre location, Grafton Street, located adjacent to the central shopping district. Of the two trial sites subjects were more receptive to participation at the University bus stop. Consequently the demographics of our sample are somewhat skewed (Figure 6). This explains the large number of users in the 18-24 age group (59%). Among the users who participated in the Grafton Street trials, roughly half were tourists or visitors to Dublin, one of the possible target markets for *Bus Catcher*.

The questionnaire was designed using a seven point Likert scale, which sought to assess user opinion on all aspects of the system.

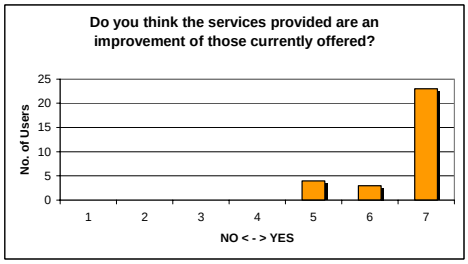

Figure 7. User responses to service improvement

The questions were expressed in an unbiased manner. Score 4 represents a response that is neither positive nor negative. Therefore for a successful test we would require that the majority of answers were for options 5, 6 and 7. As can be seen from Figures 7, 8, 9, this was indeed the case with the vast majority of the user's being very positive.

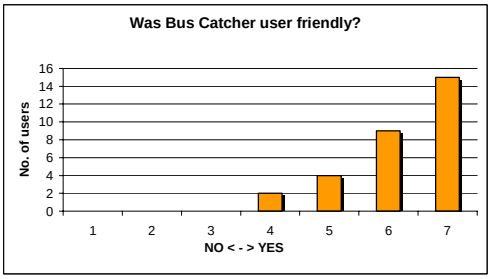

Figure 8. User responses to user friendliness

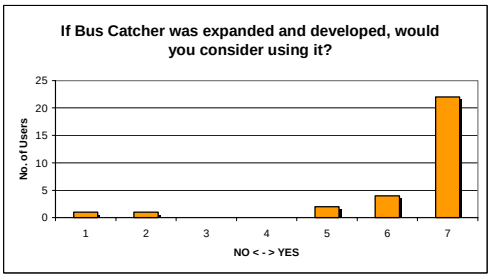

Figure 9. User responses to potential usage

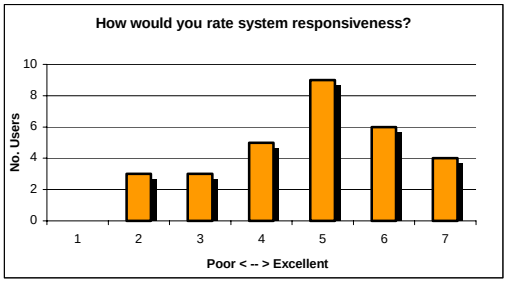

Figure 10. User responses to system responsiveness

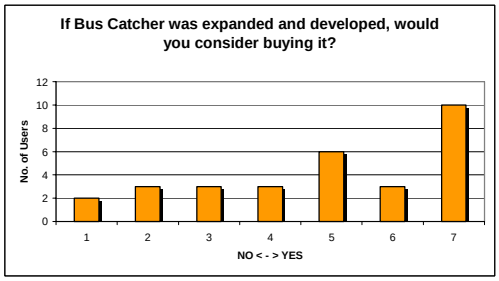

Figure 11. User responses to potential purchase

Results relating to functional benefits derived from the system (Figure 7) were very supportive as were results relating to the user friendliness (Figure 8) and disposition of users to use such a system were it to be available (Figure 9).

Results relating to the system responsiveness were not as positive, (Figure 10). However, this is due to the connection speed being limited to 9600 bps, which resuled in a delay, for example, of 15 to 20 seconds for map retrieval. If the network had supported GPRS, we would assume these results would be much more promising.

As with most products the ultimate test is in the willingness of potential customers to purchase *Bus Catcher* (Figure 11). These results were less favourable but given that some 49% of subjects were students this result may not be significant as the impoverished nature of the student globally is well documented.

6. Concluding remarks

This paper presents the *Bus Catcher* system, a context sensitive GIS/GPS integrated system targeted at the transportation sector. The main functionality provided include: display of maps, with overlaid route plotting, user and bus location, and display of bus timetables and arrival times. The system has been designed on the basis of modularity, portability and extensibility requirements.

We are currently extending the system to include information on places of interest to tourists, such as museums, galleries etc, and which buses link them. *Bus Catcher* could also provide details of bus tours throughout the city.

Alerting mechanisms are also being included: if a tourist is taking a bus that passes close to an attraction, perhaps a pop up ad alerting the tourist to the attraction including opening times, etc.

We are also investigating the possibility of integrating *Bus Catcher* with voice recognition and synthesis techniques in order, for example, to facilitate impaired users (e.g. blind people).

A further development relates to the application of personalization techniques with utilization of user profiles.

Finally, optimisation techniques could be applied to improve performance. For example, if the user asks what route would take them from one location to another and they express an interest in one of the suggested routes, the fare and timetable parts should automatically update themselves with the correct settings for that route to speed up the information retrieval process for the user.

7. References

[1] G. Abowd, C. Atkeson, .J Hong, S. Long, R. Kooper, and M. Pinkerton, "Cyberguide: A mobile context-aware tour guide" *CHI'96 Short paper*, 1997.

[2] K. Cheverst, N. Davies, N. Mitchell and A. Friday, "Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project", *Proceedings of MobiCom 2000, Boston, August 2000, pp 20-31*, 2000.

[3] A. Dey, and G. Abowd, "Towards a Better Understanding of Context and Context-Awareness", *Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness*, 2000.

[4] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction*, Prentice Hall, 1998.

[5] G. Djuknic, and R. Richton, "Geolocation and Assisted GPS", Lucent Technologies White Paper, 2001.

[6] Garmin, "Beginner's Guide to GPS", http://www.garmin.com/aboutGPS/manual.html, 2000.

[7] N. Hristova, and G.M.P. O'Hare, "Ad-me: A Context-Sensitive Advertising System", *Proc of the 3rd Int'l Conf. on Information Integration and Web-based Applications & Services (II-WAS)*, Austrian Computer Society, Linz Austria, 2000.

[8] W.S. Humphrey, *Managing the software process*, Addison-Wesley, 1989.

[10] S. Jones, and S. Gould, "*J2ME Step by Step*", IBM Developer Works, 2001.

[11] H. Lieberman, and T. Selker, "Out of Context: Computer Systems That Adapt To, and Learn from, Context", *CHI 2000, ACM Conference on Human Factors in Computing Systems*, The Hague, The Netherlands, 2000.

[12] N. Marmasse, "Location-aware information delivery with commotion". *Proc. of the 2nd Int'l Symposium on Handheld and Ubiquitous Computing (HUC)*, Bristol, UK, September 25-27, 2000.

[13] G. Pomberger and G, Blaschek, *Object Orientation and Prototyping in Software Engineering*, Prentice Hall, 1996.

[14] R. Pooley and P. Stevens, *Using UML: Software Engineering with objects and components*, Addison-Wesley,1999.

[15] B. Shneiderman, *Designing the User Interface: strategies for effective human-computer interaction*, Addison-Wesley, 1997.

[16] I. Sommerville, *Software Engineering 5th Ed*, Addison-Wesley, 2000.

[17] N. Shafer, and A. Steven, *Ubiquitous Computing and the EasyLiving Project*, Microsoft Research, 2001.

[18] S. Venkateswaran, *Java Programming for Wireless devices using J2ME/CLDC/MIDP*, California Software Labs, 1998.

[19] M. Weiser, "Some computer science issues in ubiquitous computing", *Communications of the ACM 1993*, 1993.

[20] J. Youll, J. Morris, R. Krikorian, and P. Maes, "Impulse: Location-based Agent Assistance", *Software Demos, Proc. of the Fourth Int'l Conf. on Autonomous Agents (Agents 2000)*, Barcelona, Spain, 2000.

[21] Dublin Bus Website http://www.dublinbus.ie

[22] KMap Project http://www.jshape.com/kvm/kmap.htm

[23] Java Palm OS Train Schedule http://www.ericdaugherty.com/java/palm/trainschedule/index.html

[24] The TomTom website. http://www.TomTom.com

[25] The citiKey website. http://www.e-street.com/

[26] The comMotion website. http://www.media.mit.edu/~nmarmas/comMotion.html

[27] The OVUM website. http://www.ovum.com/