<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>An agent-based architecture for wireless bus travel assistants</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authors(s)</strong></td>
<td>Strahan, Robin; Muldoon, Conor; O'Hare, G. M. P. (Greg M. P.); Bertolotto, Michela; Collier</td>
</tr>
<tr>
<td><strong>Publication date</strong></td>
<td>2003-04-22</td>
</tr>
<tr>
<td><strong>Conference details</strong></td>
<td>The Second International Workshop on Wireless Information Systems (WIS 2003), April 22-23, 2003, Angers, France</td>
</tr>
<tr>
<td><strong>Item record/more information</strong></td>
<td><a href="http://hdl.handle.net/10197/4440">http://hdl.handle.net/10197/4440</a></td>
</tr>
</tbody>
</table>
Abstract. Multi-agent systems are open and extensible systems that allow for the deployment of autonomous and proactive software components. This paper describes how a multi-agent architecture has been developed to extend a previously implemented bus travel assistant prototype called Bus Catcher [2]. Such a system was developed to provide accurate information about bus locations and arrival times to Dublin Bus users. The new version of the system is more flexible and easily extensible as it relies on generic agents responsible for channeling context sensitive services. New features have also been added to the system, including user profiling and monitoring of available hardware/service characteristics.

1 Introduction

In this paper, we describe an agent-based architecture for wireless bus travel assistants. The system we are developing delivers up to the minute bus network information that is relevant to the transportation and movement of the user. In doing so, we are able to work proactively, monitoring events and informing the user of any issues that will affect their travel (e.g. the bus they are waiting for breaks down), and where possible, offer alternative solutions.

The work described in this paper represents an evolution of an earlier prototype, called Bus Catcher, presented in [2]. Bus Catcher was developed as a travel assistant to Dublin Bus users and assumed that each bus was equipped with a GPS receiver. Such an assumption was a consequence of the announcement that the Dublin Bus Company was planning to install GPS receivers on all buses to improve monitoring and service. Currently their pilot project is running on only two routes in Dublin city.

Although the first prototype of the Bus Catcher system was fully functioning and obtained positive feedback from user trials, we decided to recast it using a multi-agent paradigm. The motivation for this was threefold. Agent Technologies: (1) hold the key to the design and delivery of autonomous and proactive software systems; (2) they are inherently extensibly suited to modular and managed development with agents introduced incrementally into an evolving community; (3) agent mobility facilitates opportunistic migration and mobility of software.
The service provided by Bus Catcher is delivered using the Agents Conveying ContExS Sensitive Services (ACCESS) architecture, a lightweight mobile agent architecture that adheres to a strong notion of agenthood [4].

Agent-based Computing [9][14] is a broad area of research that covers a diverse range of technologies and techniques. The approach taken in fabricating the agent-based version of the Bus Catcher system is through the extension and revision of Agent Factory [4], a lightweight Java application that supports the deployment of agents that are autonomous, reactive, proactive, social, and mobile. In particular, Agent Factory delivers a programming language that is derived from a temporal modal logic of belief and commitment. This programming language is augmented with a cohesive tool set and an associated software engineering methodology that supports the rapid prototyping of agents.

The remainder of this paper is organized as follows. Section 2 discusses various related systems. Sections 3 and 4 describe the ACCESS architecture and the Bus Catcher service respectively. Finally, section 5 presents various concluding remarks.

2 Related Research

The work presented in this paper 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 [5].

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 [3]. 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.

Numerous commercial systems have been successfully developed some of which are beginning to develop client bases. These include TomTom [10], which provides context aware software for PDAs, such as city maps, route planners and restaurant and attraction guides. CitiKey [11] 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 [8] 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 [7, 12] 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 [6] 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.

Our system allows users to obtain timely and accurate information about current bus locations and estimated arrival times. The version of Bus Catcher we present in this paper is also based on multi-agent technology. This capability differentiates Bus Catcher from other mobile context sensitive services targeted at the transportation sector. The use of agents also makes it more flexible, autonomous and extensible.
3 The ACCESS Architecture

The Agents Channeling ContExt Sensitive Services (ACCESS) architecture concerns itself with the duality of mobile users and mobile software agents, which support the needs of the mobile user. ACCESS is an evolution and adaptation of the Agent Factory Lite system, described elsewhere in the literature [4][5]. As can be seen in Fig.1, agents within the ACCESS system can be broadly divided into two categories, System Agents and ACCESS Agents.

The System Agents manage the platform as described by the FIPA specifications. The Agent Management System manages the creation and deletion of agents and provides a white pages directory service for agents that reside on that platform. The Directory Facilitator Agent provides a yellow pages service for agents, it has information about agents and the services they provide.

The ACCESS Agents are the generic agents responsible for channeling context sensitive services. The Device Aware Content Delivery Agent manages the local cache and the user interface. Its primary role is to ensure real time content delivery to the user. The Position Agent has the role of determining the users physical position and informing specified agents of the user’s location and direction. At present the position agent uses GPS but it is envisaged that it will incorporate other position sensing technologies in the future. The Location Aware Agent receives location information from the position agent. It acts as a hotspot manager, where a hotspot is an area of physical space limited by specific boundaries. It can inform specified agents of when the user enters or leaves a hotspot and provides functionality to allow
agents register and delete hotspots. The User Profiling Agent tries to determine what type of user it believes is using the system. It monitors the user’s location preferences, available hardware and services used. The Map Agent is responsible for dynamically generating maps merged with service specific overlays.

Due to the fact that the ACCESS architecture is generic, application specific Service Agents, which use the ACCESS architecture must be added to the system. The Service Agent functionality is dependent on the particular application being developed. Service agents for the Bus Catcher application are covered in the following section.

In the ACCESS architecture a number of reusable social behavior patterns have been encoded into the system. These interaction protocols include location-based map request, service setup, service update, service remove and profile update. Fig. 2 gives an example of how ACCESS agents interact in the location-based map request behavior pattern. In this case we are assuming that the system is tracking the user and that the map is updated when a hotspot is breached. Initially the Position Agent informs the Location Aware Agent of a position update. The Location Aware Agent detects that a hotspot has been breached. It informs the Device Aware Content Delivery agent of the breach. The Device Aware Content Delivery Agent identifies that the breach relates to a map update and subsequently requests the Map Agent to generate a new map. The Map Agent obtains the content overlay from the appropriate service agent(s) and proceeds to generate a new map. It then informs the Device Aware Content Delivery Agent of the maps URL. The Device Aware Content Delivery Agent retrieves and displays the map.

The ACCESS architecture provides a mechanism to enable agent migration. The following description describes how the interactions between fixed hosts and mobile agents are handled in the case of a disconnection from a source host and a connection to a destination host. An agent with a commitment to migrate makes a socket connection to the destination host. The source agent then transfers the agent design to the destination host. The destination host checks the local cache for the required preceptors and actuators. It then instructs the source agent to transfer its mental state and the actuators and preceptors not present at the destination. Once all of this
information has been transferred from source to destination the source host kills the migrating agent and the destination creates the migrating agent.

The ACCESS architecture exhibits social, responsive and proactive characteristics in the delivery of context sensitive services. The system is open and extensible, many context sensitive applications may be developed using the same generic core architecture. In addition to these features the decoupling of agents in general leads to greater code reuse and also makes the system more easily upgradeable.

4 The Bus Catcher Service

The Bus Catcher service is a customized application specific component that is deployed on top of the generic ACCESS system. The typical features one would associate with travel information systems motivated the functionality of Bus Catcher. These features include: fare calculation, timetable information and route planning. In addition to this Bus Catcher has the ability to act autonomously and proactively in the delivery of real time travel information.

4.1 The Bus Catcher Service Application

The Bus Catcher Application (see Fig. 3) consists of six Service Agents: the Bus Catcher Service Manager, the Bus Catcher User Agent, the Fare Agent, the Timetable Agent, the Route Agent and the Bus Agent. It also includes a database, which stores information relating to fares, bus times etc.

The Fare Agent calculates the fare for a specified journey. The TimeTable Agent handles timetable information requests, either for a route or between two points. It also modifies static timetable information to reflect the real time state of the environment. The Route Agent identifies potential routes, returns route information and the nearest ‘k’ bus stops to a specified location. Finally the Bus Agent connects to
GPS on the bus and monitors its location. It manages subscription request form other agents (such as the TimeTable Agent) and informs the subscribed agents of bus delays/progress. These four agents connect to the database and issue SQL statements to retrieve the required data.

To illustrate how the Bus Catcher Service interoperates with the ACCESS Architecture, steps (5) and (6) in the previous example (Fig. 4) are expanded to show how Bus Catcher responds to a request for Service Overlay data by the ACCESS Map Agent.

![Fig. 4. Route Information request sequence](image)

The Map Agent makes a request for service overlay data to the Bus Catcher User Agent (BCUA). In this scenario the service overlay is made up of the route path and the location of bus stops along that route. Therefore the BCUA makes a request to the Route Agent for the current trip of its user. The Route Agent responds with the data of the user’s current route and the bus stops associated with that route. The BCUA then replies to the Map Agent with the data that it requires to create the Bus Catcher Service overlay.

The most important advantage of using agents in this application over a non-agent capable version [2], is in the area of autonomy. The agents are capable of anticipating the user’s request for information and obtain it without any interaction on the users part. The agents can also query the User Profile Agent of the ACCESS architecture and thus tailor their information to suit the user, whether they are a frequent or first time user of the system. Additional advantages include the ability of the agents to migrate to different hosts in order to survive or to aid load balancing.

4.2 Typical User Scenario

It is expected that the principal use of Bus Catcher will be to help users to travel to desired destinations. Consider, for example, a user called Joe. Joe is attending a conference at University College Dublin (UCD), and wishes to go into town to do some site seeing. Joe starts the Bus Catcher application on his PDA, and requests help getting to town.

Bus Catcher uses GPS to locate Joe, and generates a list of travel routes ordered based upon his User Profile (e.g., the profile may indicate that Joe tends to choose routes that involve the least amount of walking). The information returned to Joe
includes the fare, duration, and waiting and walking times. Joe selects the most appropriate option, which is to catch the 46 bus. In response, Bus Catcher displays a custom map that is centered on Joe’s GPS coordinates, and displays his current position, the route (including bus stops) that the bus takes, and the bus Joe is to catch. This map is illustrated below in figure 5.

![Bus Catcher screenshot](image)

**Fig. 5.** An example Bus Catcher screenshot

Based upon his selection, Joe realizes that he has 15 minutes to wait and the chosen bus stop is only 5 minutes walk. This allows him to pop into the local Coffee Shop and have a cup of coffee. After 10 minutes of waiting, Joe leaves the coffee shop and walks to the bus stop in time to catch his bus.

If the bus is delayed for some reason (e.g. heavy traffic), then Bus Catcher will inform Joe of the delay, and where possible present Joe with alternative routes (e.g. a delay may result in another bus from the same stop offering a quicker travel time). Conversely, Bus Catcher will also inform Joe if the bus is ahead of schedule, ensuring that Joe knows he has less time to get to the bus stop.

## 5 Conclusion

This paper has described a recasting of the Bus Catcher Application [2] as an agent-based service that is delivered through the ACCESS architecture, a lightweight mobile agent architecture that supports the delivery of location-aware services.

Adopting an agent-oriented approach to the development of Bus Catcher has resulted in a variety of advantages: increased flexibility and extensibility of the application (adding new functionality/services is primarily a matter of creating and deploying additional agents); improved robustness and adaptivity (the application agents can dynamically re-organize themselves at run-time); and proactive delivery of real-time information that is relevant to the user (agents are able to make decisions about when and what information to send to the user).
The current approach we adopted in order to identify user specific maps involves the dynamic generation of JPEG images server side, which are subsequently transmitted to the PDA. We plan to investigate the use and integration of the Geography Markup Language (GML) language in our system. GML is an XML-based language developed by the OpenGIS Consortium [13] to enhance GIS interoperability. This will support the display of personalised vector map information and therefore reduce the amount of transmitted spatial data.

Acknowledgements

We gratefully acknowledge the kind support of Enterprise Ireland through their grant ATRP/01/209 to the E=mC^2 - Empowering the mobile Citizen Creatively project.

References