ACCESS: An Agent Based Architecture for the Rapid Prototyping of Location Aware Services

Robin Strahan, Gregory O'Hare, Conor Muldoon, Donnacha Phelan,
and Rem Collier

Department of Computer Science, University College Dublin, Dublin 4, Ireland
{Robin.Strahan, Gregory.OHare, Conor.Muldoon,
Donnacha.Phelan, Rem.Collier}@ucd.ie
<http://emc2.ucd.ie>

Abstract. We describe the Agents Channelling ContExt Sensitive Services (ACCESS) architecture, an open agent-based architecture that supports the development and deployment of multiple heterogeneous context sensitive services. We detail the ACCESS architecture and describe the scenario of an individual arriving in a city and using his ACCESS enabled PDA to secure lodgings.

1 Introduction

This paper describes the Agents Channelling ContExt Sensitive Services (ACCESS) architecture, an open agent-based architecture that supports the development and deployment of multiple heterogeneous context sensitive services. A key motivation for the development of ACCESS was the recognition that most location-aware context sensitive applications exhibit a common core of functionality. That is, such systems commonly employ: location-sensing technologies, dynamic generation of maps that are customised to the user, support for the management of the users context, and service brokerage.

In this paper we review some related systems, detail the ACCESS architecture paying particular attention to three facets: map caching, user profiling, and agent tuning and finally, we demonstrate the interaction between ACCESS and a mobile user by presenting the scenario of an individual arriving in a city and seeking a hotel room.

2 Related Work

There are a number of existing systems that aim to provide a basis for the development of agent-based context aware systems.

MyCampus is an agent-based environment for context-aware mobile services developed at Carnegie Mellon University [1]. The system aims to aid a PDA equipped user in carrying out different tasks (planning events, send messages, find other users, etc) by accessing Intranet and Internet services. The information from these services is filtered by use of context, such as the user's location, their class schedule, the location of their friends and the weather. Personal preferences are also used to tailor the information provided to the user.

The SALSA framework [2] allows developers to implement autonomous agents for ubiquitous computing systems. It uses WiFi to communicate and estimates the user's position by triangulation of at least three 802.11 access points. It uses an open source Instant Messaging Server (Jabber) to notify the state of people and agents, and to handle the interaction between people, agents, and devices through XML messages. Agents abstract the complexities associated with the collaboration of users and the opportunistic interaction with services of ubiquitous computing environments.

The COBRA (COntext BRoker Architecture) [3] is a broker-centric agent architecture that aims to reduce the cost and difficulty of building context-aware systems. At its centre is an autonomous agent called the Domain Context Broker that controls a context model of a specific domain or location. It updates this model, using data from heterogeneous sources (physical sensors, Web, pattern behaviours, etc) and attempts to resolve any detected context conflicts. It shares this information with independent agents within its domain. The use of broker-centric allows computational intensive activities such as gathering and interpreting data to be preformed by a suitable device rather than on resource limited devices.

3 The ACCESS Architecture

As an agent-based architecture intended to support the rapid development and deployment of multiple heterogeneous context sensitive mobile services, the Agents Channelling COntExt Sensitive Services (ACCESS) architecture seeks to provide support for multi-user environments by offering personalization of content, through user profiling and context, as well as support for mobile lightweight intentional agents and intelligent prediction of user service needs.

The ACCESS architecture has been designed to extend Agent Factory (AF), a well-established framework [6][7] for the development and deployment of multi-agent systems. The ACCESS architecture enhances the basic infrastructure delivered by the AF run-time environment through the implementation of additional agents, which deliver functionality that is common to many context-sensitive applications. These ACCESS Management Agents form a cohesive layer into which multiple heterogeneous context-sensitive services may be plugged. In addition, the existing AF development methodology has been augmented to aid the developer in producing ACCESS-compliant services. The resulting ACCESS toolkit facilitates the rapid prototyping of context-sensitive services by allowing service developers to focus on the service business logic, and not the infrastructure required to deliver it.

4 ACCESS Management Agents

The ACCESS Management agents implement the core functionality of the ACCESS architecture, which includes context management, user profiling, map generation, content delivery, account management, and location sensing. The User Profiling Agent, acting in tandem with the Activity Analyzer Agent and the Context Agent, undertakes personalization. The User Profiling Agent provides a mechanism to allow agents to request a user's preferences, adopting a neighbourhood-based approach to

identify this set of similar users. Implicit profile information is obtained using data mining techniques on recorded user activity, for instance when examining service usage, what was used, when it was used, and where it was used is noted.

The notion of user context is the concept that ACCESS employs to determine not only what services the user may require, but also where and when to offer them. A user's context is considered a combination of their location, preferences and previous activities. It is the responsibility of the Context Agent to create and manage user specific hotspots. A hotspot is a region of space-time that is limited by specified bounds (e.g. an area of 100 square meters centred around a particular shop, during business hours). The Context Agent informs the Profiling Agent when a user encounters a hotspot, which then decides whether the hotspot breach is relevant to the user.

Fig. 1. Access Management Agents

The Map Server element is made up of the Map Broker and one or more Map Agents. The Map Broker manages the dynamic generation of maps for a specified location. When the Map Broker is requested to generate a map, it communicates with a number of Map Agents to determine which of them can produce the most suitable map. The selected Map Agent then generates the map segment which can subsequently be merged with service specific content overlay. This peer community of Map Agents allows for distribution of load and for a degree of failure protection.

The Position Agent is responsible for periodically informing subscribed agents of updates in the user or device's position. The Position Agent currently determines position by use of a GPS receiver.

To provide service specific functionality, abstract Service Manager and Service Delivery roles are provided that a service developer may extend. These roles encapsulate the requisite agent-based functionality for interacting with the core infrastructure. The Service Manager role is responsible for advertisement of the service by registering with a Service Broker Agent and for registering the service with other ACCESS compliant services that it may wish to use. The Service Delivery role acts as the users representative within a service and collaborates with the Service Manager Role and an

Interface Agent. The service developer must extend this role with additional behaviour patterns to interact with service specific agents.

The Interface Agent manages the user's interaction with the Graphical User Interface within the system. Additionally it provides a means by which it may dynamically extend its functionality at run-time through the use of Java introspection for the provision of service specific functionality.

4.1 Map Caching

A key feature of any application or service built using the ACCESS architecture is the ability to provide dynamically generated maps. But with ACCESS being primarily aimed at resource-limited devices such as PDAs or smart phones, it is desirable to minimise the detriment in making use of this feature.

There are two methods used to minimise the cost of using the map feature; the first focuses on minimising the delay between a device requesting a map and rendering it, the second attempts to minimise both the time delay and the communication or bandwidth cost in displaying a map.

The first method consists of pre-emptive or active caching. This is accomplished by attempting to determine the user's future direction of travel. The Cache Agent tracks the user's heading and calculates the average heading. When the user approaches the edge of the current map, this average heading is used to establish which map segments are requested by the Cache Agent from the Map Server. So, for example, if the average heading was 90° , then the Cache Agent would request three map segments covering the area to the east of the user's current location. In addition, knowledge communicated to the Cache Agent from other agents, such as user location, history from the Activity Analyzer Agent etc, can influence which map segments are requested. This pre-fetching of the segments results in the map segment being immediately available for rendering, thus minimising the delay in changing the currently displayed map segment to the next segment.

The second method attempts to remove the bandwidth or communication cost in sending and receiving data between the Cache and Map Agents. This is done through the use of passive caching. Each map segment that is received from the Map Server, including segments used in active caching, is stored locally (cached) on the device by the Cache Agent. When a new map segment is required by the user, the Cache Agent will attempt to retrieve a suitable map segment from this cache and will only request a map segment from the Map Server if there are no segments in the cache that meets the user's requirements. If several suitable segments from the cache are found, the Cache Agent selects the segment whose centre is closest to the user's current location, thereby attempting to maximise the amount of time before another segment is required.

4.2 User Profiling

The fact that ACCESS is a tool kit for the deployment of context sensitive services places some unique constraints on the design of the User Profiling Agent, such as the fact that anticipating the profiling requirements of individual service developers is difficult, if not impossible. In an attempt to overcome this obstacle ACCESS users are

provided with a ‘generic’ ACCESS profile containing personal data such as gender, address, age, job description, education etc., which service developers can augment with a service specific profile. Using this generic profile, the User Profiling Agent can identify a set of similar users to use as recommendation partners for the active user. Users *must* explicitly complete their generic profile when registering with ACCESS, whether a user is required to supply a set of service specific preferences is a matter for the individual service developer. Should a user not enter a service profile, some personalisation is possible from the ACCESS profile. Items being recommended to the user may also have their own profiles listing their distinguishing features (for instance in a hotel finder application each hotel might have a profile listing its amenities). Again this is a matter for the service developer.

The ACCESS User Profiler Agent adopts a neighbourhood-based approach in identifying a set of similar users to use as recommendation partners for the active user. The measure of similarity adopted to identify this set is neither a correlation measure nor a vector-based cosine similarity metric. Instead, a simple overlap measure, where, for each user, the number of preferences rated identically to the active user, constitutes the overlap similarity value.

$$\text{sim}(x,y) = \frac{|\text{preferences}(x) \cap \text{preferences}(y)|}{|\text{preferences}(x) \cup \text{preferences}(y)|}$$

The number of overlapping items in the active user (x)’s profile and a test user (y)’s profile is simply counted. The union of all their preferences, as a normalising factor, divides this value. This method, known as the Jaccard similarity metric, is quite a simple way of measuring user similarity, but we can take the similarity value obtained from this method as indicative of the ACCESS user profiles true similarity, because the ratings in each user’s profile are obtained by explicit means (that is, users consciously elected to give a certain value to a given question), rather than implicitly-required ratings, which can sometimes be misconstrued or inaccurate reflections of users profiles.

4.3 Agent Tuning

Collaborative Performance Tuning is an autonomic procedure by which ACCESS management agents collectively alter their response times so as to adapt to the dynamic utilities and requirements of the software systems that they represent. The tuning framework enables inactive agents to go into a temporary hibernation state without affecting their ability to react to their environment in a timely manner. The hibernation subsystem monitors an agent’s commitments and reduces the number of redundant CPU cycles wasted on agents whose mental state does not change between iterations of their control algorithm. Once an agent goes into hibernation, other agents on the platform may opportunistically reduce their response times to take advantage of the additional computational resources made available.

Other systems [9][10] have been developed for the collective negotiation of CPU resources. Collaborative Performance Tuning differs from these systems in that a BDI approach is used that utilises the notions of joint intentions and mutual beliefs thus coercing agents into forming coalitions and to act as a team. This prevents agents getting into prisoner dilemma type scenarios in which agents making locally optimal

decisions create socially unacceptable behaviour patterns in which all agents concerned are worse off as an inevitable consequence of rational choice.

As the inference engine executes, it examines the number of commitments an agent has on its current deliberative cycle. If the agent does not have any commitments it is said to be inactive on that cycle. The rationale for this is that given an agent's current mental state, if an agent is inactive on a particular cycle and its mental state will not change on the next cycle, the agent need not deliberate on the next cycle as it knows that its mental state will not be altered.

At various stages throughout execution ACCESS agents may collectively alter their response times to adapt to the evolving system requirements. Joint intentions are utilized whereby rather than unilaterally reducing its response time an agent must adopt a commitment to make its intentions mutually believable to all active agents on the platform. This implies that an agent must communicate with and receive a response from all other active agents before taking action. Thus ensuring that agents act in a manner that is beneficial to the team as a whole. The problem with collective decision-making however is that decisions can only be made as fast as the slowest decision maker. Within the ACCESS architecture the Position Agent often operates with quite a large response time, agents cannot self-tune or pre-empt the Position Agent because to do so they would have to act unilaterally. This prevents Collaborative Performance Tuning being used for tasks with a low level of granularity such as rendering maps, which with modern just-in-time compilers would be completed prior to the termination of the negotiation process. Therefore Collaborative Performance Tuning is primarily used to improve efficiency in medium to high-level quality of service problems or to improve the general performance of the system over the entire course of its execution as the system adapts to dynamic environment conditions.

5 A User Scenario

To illustrate the usefulness and appropriateness of ACCESS we consider a fictitious character Joe arriving at an airport equipped with a PDA hosting ACCESS. Initially when Joe loads ACCESS at the airport he receives an advertisement for Sos, an accommodation finding application, which his Interface Agent received from a Service Broker. Joe clicks the advertisement to download Sos and to start the Service. The Interface Agent contacts an appropriate Service Manager whose name it received from the Service Broker to begin using Sos. When the Interface Agent registers with the Sos Service Manager it is delegated to a Service Delivery Agent. The Sos Service Delivery Agent then checks to see if Joe has been registered with an additional Meeting Service. The Meeting Service is used to keep track of the user's diary. If Joe has been registered the Service Delivery Agent of Sos requests the Service Delivery Agent of the Meeting Service to inform it of meetings relevant to Joe's current spatial context. The Meeting Service is aware of the locations of meetings that Joe will be attending and thus returns an appropriate list.

Joe is now presented with a screen (Fig. 3) allowing him to select his room preferences and a drop down list containing points of interest that has been populated with

relevant meetings and popular tourist attractions. Selecting an option from this list indicates that Joe would like his hotel to be located within the vicinity of the chosen item. Once the room details have been selected, the Profile Agent is contacted to obtain an appropriate list of hotels. The Profile Agent factors in implicit and explicit preferences, tariffs charged for the advertisement of hotels in addition to proximity of user's location and points of interest when generating this list. Once Joe receives the list and selects a hotel, the Sos Service Delivery Agent pushes the advertisement for a Bus Service. When Joe loads the Bus Service it receives the destination hotel location, which it uses to work out an appropriate route. The Caching Agent is informed of this route, which it uses to pre-emptively request the generation of maps for the impending journey. The Bus Service Delivery Agent collaborates with an additional Location Service, which operates in the background without the Joe's knowledge. The Location Service works with the Caching Agent in obtaining maps from the map server centred on Joe's position for tracking purposes.

Fig. 3. Room Select Screen and Position Agent tuning Mental State

When Joe gets onto the bus and starts travelling the Position Agent realizes that Joe's average velocity has increased and that it needs to reduce its response time. Fig. 3 illustrates the Position Agent's mental state. The Position adopts a commitment to reduce its response time and thus informs the other team members of the situation. On receiving this information the Interface Agent adopts a commitment to have its response time increased whereas the Cache Agent adopts a commitment to reduce its response time. Once all agents receive replies from their teammates their response times are altered. The Agents will maintain these altered response times so long as Joe's average velocity remains within a certain range.

6 Conclusions

This paper has described ACCESS, a generic agent based architecture for the rapid development and role out of location aware services. The key characteristics and differentiators of this architecture are the provision of lightweight intentional mobile agents, which offer an agent tuning ability, support dynamic profile updates, dynamic map generation and a rich concept of context. ACCESS context awareness enables and underpins degradation or enhancement of content to suit the device context and user needs.

Acknowledgements

We gratefully acknowledge the support of Enterprise Ireland (grant ATRP/01/209) and Science Foundation Ireland through Modelling Collaborative Reasoners (SFI Investigator Award).

References

1. Sadeh N., Chan T., Van L., Kwon O. and Takizawa K., Creating an Open Agent Environment for Context-aware M-Commerce, in *Agentcities: Challenges in Open Agent Environments*, LNAI, Springer Verlag (2003).
2. Rodríguez M. and Favela J., A Framework for Supporting Autonomous Agents in Ubiquitous Computing Environments, *Proceedings of System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp 2003)*, Seattle, Washington, (2003).
3. Chen M, An Intelligent Broker Architecture for Context-Aware Systems, PhD Dissertation Proposal, UMBC (2002).
4. Pearce J., IBM: Pervasive Computing is the future, ZD Net article, Jan 30 2003 (2003).
5. Creese S., Future Challenges in Pervasive Computing Environments, SC Infosec article, Mar 5 2003 (2003).
6. Collier, R., Agent Factory: A Framework for the Engineering of Agent-Oriented Applications, *Ph.D. Thesis*, Computer Science Dept., University College Dublin, Ireland (2001).
7. Collier, R.W., O'Hare G.M.P., Lowen, T., Rooney, C.F.B., Beyond Prototyping in the Factory of the Agents, 3rd Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'03), Prague, Czech Republic (2003).
8. 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).
9. Soh, L.-K., Tsatsoulis, C., Agent-Based Argumentative Ne-gotiations with Case-Based Reasoning. AAAI Fall Symposium Series on Negotiation Methods for Autonomous Cooperative Systems, North Falmouth, Mass. (2001)
10. Walsh, W.E. and Wellman, M.P., A market protocol for decentralized task allocation: Extended version. In *The Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)* (1998).