

Agents as Catalysts for Mobile Computing

G.M.P. O'Hare¹, M.J. O'Grady², R.W. Collier², and S. Keegan²

¹ Adaptive Information Cluster (AIC),
Department of Computer Science, University College Dublin (UCD),
Belfield, Dublin 4, Ireland
gregory.ohare@ucd.ie

² Practice & Research in Intelligent Systems & Media (PRISM) Laboratory,
Department of Computer Science, University College Dublin (UCD),
Belfield, Dublin 4, Ireland
{michael.j.ogrady, rem.collier, stephen.keegan}@ucd.ie

Abstract. Agent-Oriented Programming (AOP) offers an alternative and radical approach to the development of information systems in various domains. However, one domain that AOP has only minimally affected, at least up until now, is that of mobile computing. Until recently, the use of strong intentional agents in such a domain has been considered impractical and, indeed, computationally intractable. In this paper, Agent Factory, a system for the fabrication of strong intelligent agents, is reviewed in the light of agent deployment on mobile devices. As an illustration of the potential of agents in mobile applications, two archetypical mobile computing applications, realised through Agent Factory, are described.

1 Introduction

This paper explores one particular genre of Agent-Oriented Information Systems (AOIS), namely mobile and ubiquitous computing. Such systems are typified by devices that are computationally challenged in terms of screen, memory and processor real estate, as well as networks that are resource-bounded and bandwidth restricted. Users expect content relevance, timeliness and a degree of personalization to their individual needs. These demands are significant and demanding, and necessitate systems that exhibit the ability to anticipate the content and service needs of users.

Until recently, the use of strong intentional agents in such a domain would have been considered impractical, and, indeed, computationally intractable. In this paper, Agent Factory, a system for the fabrication of strong, intelligent, mobile and agile agents is utilized. In particular, its strategies for realising such agents in the computationally-constrained world of mobile computing are outlined. We illustrate the successful deployment of agent technology generally and Agent Factory specifically via two archetypical mobile computing applications. The first, EasiShop, a ubiquitous commerce (uCommerce) application, enables shoppers to seek out good deals while wandering an arbitrary shopping mall or high street. The second, Gulliver's Genie, is a mobile context-sensitive tourist guide that focuses on the delivery of personalised multimedia content in a just-in-time basis.

2 Agent Factory

Agent Factory (AF) [1] [2] [3] is a cohesive framework, illustrated in Figure 1., for the development and deployment of agent-oriented applications that has been developed by the authors. Central to this framework is the Agent Factory Agent Programming Language (AF-APL), an Agent-Oriented Programming (AOP) language that supports the fabrication of agents that are autonomous, situated, socially able, intentional, rational and mobile. However, Agent Factory differs from other AOP offerings in that AF-APL has been embedded within a distributed FIPA-compliant [4] Run-Time Environment, and supports the development and deployment of agents through an integrated development environment, and an associated software engineering methodology. Details of these layers are presented in the following sections.

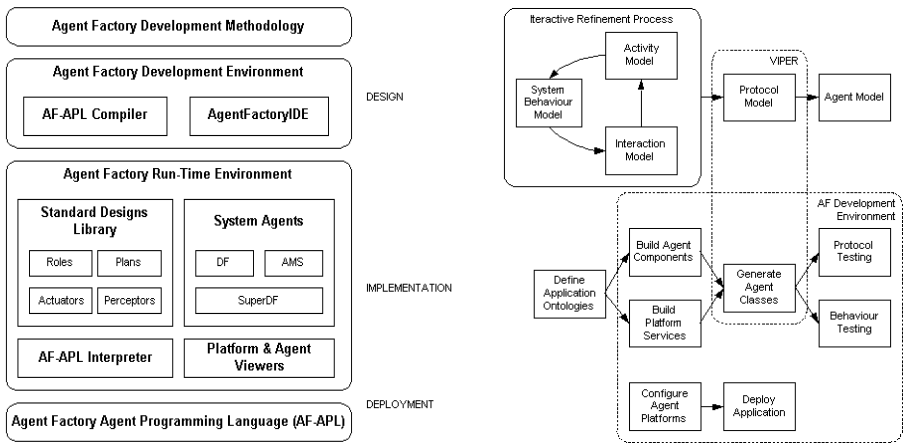


Fig. 1. The Agent Factory Framework and its associated Development Methodology

A key concern in the design of AF has been to ensure that AF-APL agents can be deployed on Personal Digital Assistants (PDAs). This has been achieved by ensuring that the Run-Time Environment, which includes the AF-APL Interpreter, is compliant with version 1.1.8 of the Java SDK (a.k.a. Personal Java for Mobile Devices). To check compatibility with future versions of Java, J2ME-compliant versions of the Run-Time Environment have also been developed. However, due to incompatibilities between Personal Java and J2ME, and as a result of our wish to ensure that AF can be deployed on the most prevalent operating system / JVM configuration for PDAs (e.g. MS PocketPC and Jeode), AF is currently not J2ME-compliant.

2.1 AF-APL

AF-APL is a declarative Agent-Oriented Programming (AOP) language that supports the programming of agent behaviours. The basic premise behind AF-APL is the view that complex agent behaviours can be more naturally modelled by viewing agents to

be mental entities that maintain an internal *mental state* that is comprised of mental attitudes, in this case: *beliefs* and *commitments*. Beliefs describe, using a first-order logic representation language, the current state of the agent and its environment, and commitments describe the current (and future) activities that the agent has decided to perform. These will be illustrated further in Sections 3.3 and 4.4 respectively. Finally, decisions are modelled through a set of commitment rules that map situations (a conjunction of positive and negative beliefs) onto commitments. These rules are checked repeatedly within a sense-deliberate-act cycle.

As with other similar offerings, such as Goal-Directed 3APL [5] and AgentSpeak(L) [6], the syntax and semantics of AF-APL have been formally specified. In particular, AF-APL is based upon a logical model of reasoning that is centred about the notion of commitment. This contrasts with the more traditional Belief-Desire-Intention (BDI) architecture [7] on which these other approaches are based. In our model, intentions are synonymous with commitments, while goals are represented implicitly as the situations in which the agent must commit to a given course of action. Details of both the formal model and the syntax and semantics of AF-APL can be found in [2]. Finally, [8] presents a recently proposed extension to AF-APL that supports both goals and means-end reasoning.

2.2 The Run-Time Environment

The AF-APL interpreter is embedded within a distributed FIPA-compliant Run-Time Environment (RTE). Specifically, AF adheres to the following FIPA specifications:

- The FIPA Abstract Architecture Specification (0001);
- The FIPA Agent Management Specification (00023);
- The FIPA ACL Message Structure Specification (00061);
- The FIPA Agent Message Transport Service Specification (00067);
- The FIPA ACL Message Representation in String Specification (00070);
- The FIPA Message Transport Protocol for HTTP Specification (00084);
- The FIPA Agent Message Transport Envelope Representation in XML Specification (00085).

Within the context of these specifications, the RTE is organised as a collection of *agent platforms*. An Agent Platform (AP) provides the basic infrastructure that is necessary to deploy agents. Specifically, each AP implements a number of *platform services*, which provide various mandatory and optional infrastructure services. One platform service is the Agent Management System (AMS) service. This service is mandatory and is responsible for the creation, termination, suspension, resumption, registration, deregistration and execution of agents that are residing on the AP. A second service is the optional HTTP Message Transport Service, which provides a HTTP-based message-passing infrastructure for the Run-Time Environment. Other services include directory facilitator services (i.e. yellow pages services), persistence services, migration services and cloning services. During the development of agent-oriented applications, developers are required to identify and implement an appropriate set of platform services.

In addition to the platform services, the RTE also implements a number of System Agents, the most important being the AMS and DF agents. In essence, both these

agents are agent wrappers that envelop the associated AMS and DF platform services respectively. Both agents control access to the relevant service on the AP.

2.3 The Development Methodology

Methodological support for the fabrication of agent-oriented applications using Agent Factory is provided through a software engineering process that supports the design, implementation, testing and deployment phases of the software engineering lifecycle and utilizes the UML as its modelling language. A diagrammatical overview of this process, details of which can be found in [9], is presented in Figure 1. It can be seen that the design stage of the process focuses around the development of five models:

1. The **System Behaviour Model (SBM)** is used to identify the main roles that agents will play within the system, and to associate those roles with the key system behaviours. Visually, this model is formalised using a customised UML Use Case Diagram where actors are stereotyped as roles, and use-cases are stereotyped as system behaviours.
2. The **Interaction Model (IM)** expands on the SBM through the modeling of the interactions that occur within each of the system behaviors. Visually, this model is formalized using a customized UML Collaboration Diagram.
3. The **Activity Model (ActM)** complements the IM, in that it expands on the SBM through the modeling of the set of activities that occur within each of the system behaviors. Visually, this model is formalized using a customized UML Activity Diagram where swimlanes are employed to represent roles.
4. The **Protocol Model (PM)** represents the transition point where the focus turns from understanding the system behaviors to the formalization of those behaviors as a set of protocols and agent-classes. Visually, these protocols are represented using Agent UML Sequence Diagrams.
5. Finally, the **Agent Model (AgtM)** completes the design by switching the focus from roles, interactions and activities to a more agent-centric view of the target system in which roles and agent classes become the constituent components.

2.4 The Development Environment

Support for the development of agents is realized through a number of toolsets. The Agent Factory Integrated Development Environment (IDE) provides a standard programming environment in the vein of NetBeans and JBuilder. Specifically, the editor includes features such as syntax highlighting, code compilation and application execution.

In addition, VIPER [10] is a graphical tool suite that allows the user to compose the Agent UML Sequence Diagrams that sit at the heart of the Protocol Model. VIPER is comprised of two tools: a Protocol Editor that provides a visual tool for generating Agent UML Sequence Diagrams and a Rule Editor that further supports the user by guiding them through the step of implementing the protocols in AF-APL.

In addition to the tools that have been provided to support the development of AF-APL agents, the Agent Factory Development Environment also includes a suite of tools that facilitate the testing and debugging of agent-oriented applications. These tools are associated with the Agent Platform component of the Run-Time Environment and include:

- the **Agent Viewer Tool**, which allows the developer to monitor and modify the agents internal state;
- the **Message Sender**, which allows the developer to interact with other agents as if they were themselves an agent; and
- the **Community Monitor**, which allows the developer to monitor interactions between a specified set of agents.

3 EasiShop

Delivering a real-time shopping solution is regarded as a litmus test for intelligent mobile agent technologies. Attempts to deliver such a mechanism on the web are well documented. Some insights revealed by these attempts hold a certain relevance to our

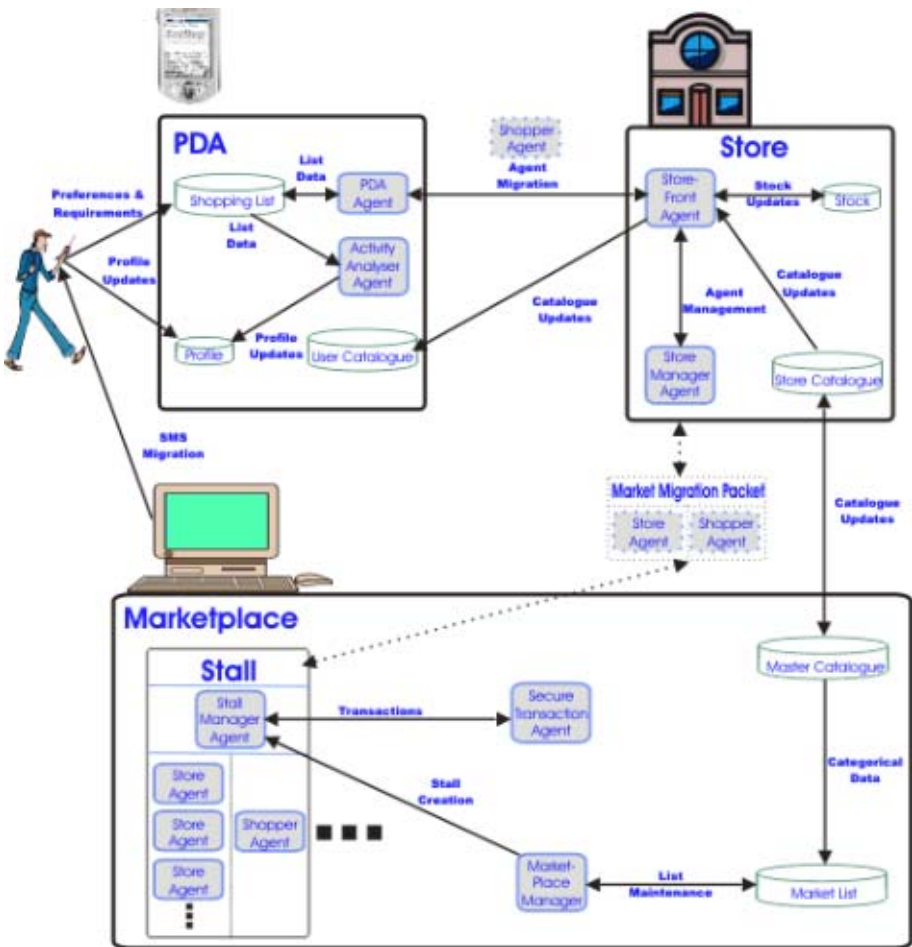


Fig. 2. Overview of EasiShop

domain. Ringo [11] for example, incorporated a collaborative filtering mechanism. This approach has since been adopted in the commercial realm by vendors such as Amazon. ShopBot [12] was an agent that could learn how to submit queries to e-commerce sites and interpret the resulting dataset to identify lowest-priced items. ShopBot automated the process of building wrappers to parse semi-structured HTML documents and extract features such as product descriptions and prices. Tete@Tete agent technology [13] strives to deliver integrated product brokering, merchant brokering and negotiation, thus encompassing three stages of the consumer buying behavior model as documented by Howard and Sheth [14].

Research into the role of intelligent mobile shopping systems is less mature. The Impulse project [15] developed at MIT augments GPS with agent technologies to provide context-sensitive information to the user. The main objective of the MyGrocer [16] project is to enable interactivity, personalisation and automation of home replenishment activities for products in the grocery retail sector. The Shopper's Eye experiment [17] introduces the concept of location-based filtering to assist the shopper. In contrast to all of these, however, EasiShop [18] [19] adopts a user-centric approach, realised through a suite of strong intentional agents. Its principal objective is the delivery of a scalable architecture that permits the partial automation of the shopping process while maximizing the opportunity for a prospective shopper to avail of optimum consumer conditions, for example, price, geographical proximity, after-sales service and so on.

3.1 The EasiShop Scenario

To envisage the modus operandi of EasiShop, let's assume the situation whereby the shopper is in possession of a PDA with the onboard client-side EasiShop system. To initiate the system, the shopper provides a composite set of preferences, profile and shopping list information. This is accomplished using the standard input of the PDA – stylus or virtual keyboard. The EasiShop architecture is complemented by another aspect of the system – the server-side components. It is envisaged that each participating retailer will make provision for an EasiShop Hotspots (EH). The EH is an active bluetooth broadcast area, strategically positioned at the foyer of the store. This zone provides a channel by which negotiation and trade can occur between the PDA Agent and the representative Store Agent. Figure 2 illustrates the main components of EasiShop.

The architecture is completed by the EasiShop Marketplace. This is a remote server containing software implementing an agent-based auction protocol. Using a specially adapted auction protocol (based on the Vickrey model), the system permits expedient and Pareto-efficient negotiation in real-time. The Marketplace also contains a secure datasink in which user profile information is retained. This information can be utilized to provide the shopper with more appropriate product offerings as well as providing Store Agents with valuable information as to what type of potential customer is in the proximity. It can be seen that the EH acts as a conduit between the shopper, represented by a personal PDA Agent, and the Marketplace.

3.2 Implementation

The EasiShop system was realised as a seven layer architecture, as illustrated in Figure 3. These layers are now considered briefly:

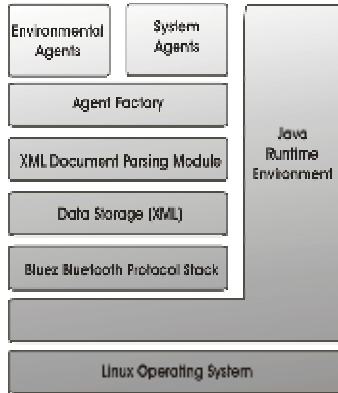


Fig. 3. The EasiShop Architecture

Agent Factory. System intelligence is delivered through Agent Factory.

Data Structures (XML). An adaptation of the Contract Net Protocol as proposed by Davis and Smith [20] has been proven to be the most appropriate mechanism for EasiShop. Agents utilise the protocol to make contracts which are binding for sales and purchases. For each bidding cycle, the communication involves four message types: Task announcement, Bidding, Awarding and Report Messages. In implementing the Contract Net Protocol, a suitable product description ontology is necessitated. For EasiShop, the UNSPSC [21] has been proven to be the most appropriate. XML was selected to represent the product information because of its inherent portability and for the fact that it becoming a de facto standard. It is extensible and separates content from presentation. Each time the shopper passes an EasiShop Hotspot, the difference (if any) between the product database on the EasiShop server and the PDA is sent to the PDA. In essence, all that is sent is the *diff* of the two files.

Auction Protocol. There are several existing protocols for multiple seller/single buyer auctions. English, Dutch and Vickrey are some examples. Each has a set of inherent advantages and drawbacks. When compared to other auction protocols, Vickrey auctions have the advantage that their duration is known prior to the auction taking place (each interested party bids only once). This factor holds particular relevance to the EasiShop domain, where, due to the fact that users are mobile and typically moving at a walking pace, real-time auction resolution is imperative. Furthermore, since the dominant bidding strategy of the Vickrey protocol is to bid to one's true evaluation, counter-speculation is avoided and a realistic pseudo-marketplace can be realised. Dutch auctions, on the other hand, have been shown to

provide more revenue for the seller than Vickrey auctions in situations with three or more bidders, while both English and Vickrey auctions provide higher revenue for the auctioneer than their Dutch and first-price sealed bid counterparts.

BlueZ Bluetooth Protocol Stack. BlueZ is the official Linux Bluetooth protocol stack. It is an Open Source project distributed under GNU General Public License (GPL). To facilitate EasiShop, a module that accesses and utilises the BlueZ core was developed. As part of this process, it was necessary to construct an interface to permit information flow between the BlueZ module and the JRE.

Java Runtime Environment (JRE). The Blackdown JRE 1.3.1 RC1 for the Linux/ARM architecture is utilised on the PDA. Kaffe, a clean room implementation of the Java virtual machine, plus the associated class libraries needed to provide a JRE is used at the Store and Marketplace hosts. Blackdown is the JRE of choice for implementation on the PDA since it offers sophisticated GUI functionality. This includes the Swing components like JTree and JEditPanel, which allow for complex XML document display and manipulation techniques.

Linux Operating System. The Server-side components reside on a Linux 2.4.18-3 server. The PDA (an IPAQ 3870) runs the familiar v0.6.1 Linux build (2.4.18-3) from handhelds.org. Many of the inherent benefits of Linux are applicable to the EasiShop implementation. These are widely documented and encompass stability, security, open source, network orientation, speed and efficiency.

3.3 Agents @ Work in EasiShop

To demonstrate how agents support the EasiShop vision, consider the following series of events. In the typical EasiShop scenario, the shopper, equipped with a PDA, is situated in a busy shopping street. The shopper enters an EasiShop hotspot when passing within bluetooth broadcast range of a participating retailing outlet. Upon entering the hotspot, agent collaboration ensues. Activity is initiated by the PDA Agent (housed on the user's PDA) when it (having briefly liaised with the StoreFront Agent) uploads a shopping list to the shop. At this point, the Store Manager agent determines that one of the products on the list is currently in stock here. The PDA Agent is informed of this and may choose at this point to initiate the migration of the Shopper Agent from the PDA to the shop. The commitment rule that specifies this behavior is as follows and is further illustrated in Figure 4.

```
BELIEF(ShopHasProduct(?product)) &
BELIEF(inHotspot(?thystore)) =>
COMMIT(Self, Now, BELIEF(true), Migrate2Store(?thystore));
```

In general, the Agent Factory IDE proved to be invaluable during the delivery of EasiShop. The Agent Viewer Tool, the Message Sender and the Community Monitor were seen to be essential in debugging and optimizing agent states and interactions. The Agent Viewer Tool in particular enables a convenient means of visualizing real-time inter-agent communications. At present, the VIPER tool has been integrated to a lesser degree with the overall system when compared with the other tools. However, the potential of this tool, particularly in terms of delivering efficient inter-agent activity protocols, became apparent during the implementation and full integration would undoubtedly be of great benefit.

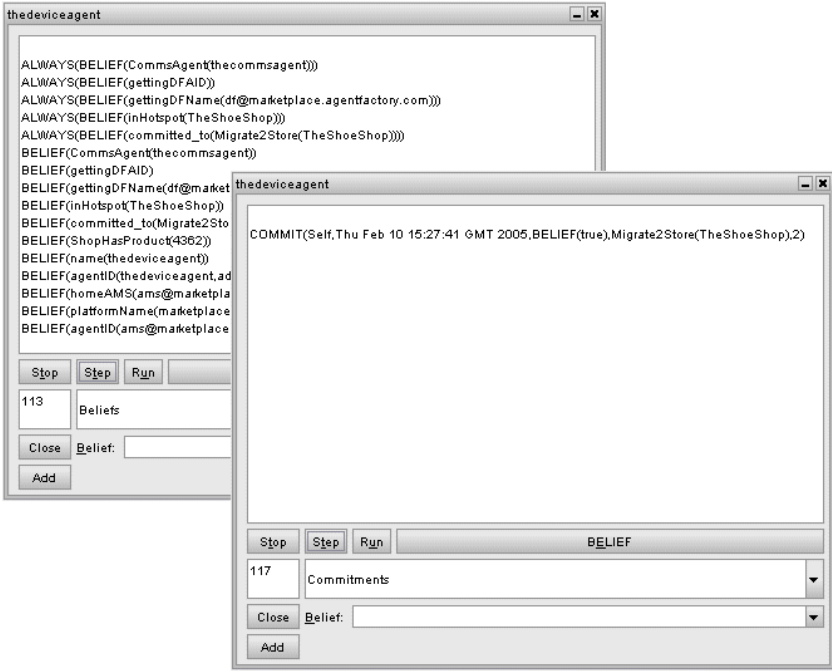


Fig. 4. A commitment is triggered in EasiShop

4 Gulliver's Genie

Gulliver's Genie [22] [23] is an application developed with the primary intention of delivering services to mobile users. Though it currently concentrates on the tourist domain, we expect it to evolve into a generic and customisable application that will deliver various services to mobile users. In addressing the needs of tourists, it is not unique, as several research disciplines have found the tourist domain a fertile testbed for theories and applications. Projects that come closest to the Genie in objectives and scope include GUIDE [24], a context-sensitive tourist guide for the city of Lancaster, and CRUMPET [25], developed for the city of Heidelberg. Examples of commercial products include Vindigo [26] and Portable Guide [27]. However, it is its use of BDI agents that differentiates the Genie from all other efforts in this area. Indeed, it may be regarded as a Multi-Agent System (MAS), the context of which envelops the Internet, a wireless data network and PDAs, and comprising a suite of agents all collaborating to obtain the Genie's goal: the timely distribution of information to roaming tourists.

4.1 Genie Services

At present, services provided by the Genie fall into two categories:

1. Navigation: Navigation support is a service that mobile users, and tourists in particular, find useful. Such a service can range from simple to sophisticated

with the Genie lying somewhere in between. An electronic map, scaled to street level and with all the relevant attractions highlighted, constitutes the initial component of the Genie’s navigation support service. However, this is augmented with a real-time position determination facility that ensures that the tourist’s position and orientation are always highlighted on the map. In this way, the tourist can see their location at any given instance.

2. **Cultural Information:** Though the motivation of individual tourists may vary considerably, experiencing the culture of a new environment is a common goal. The Genie seeks to facilitate this experience by delivering concise multimedia presentations on the various tourist attractions within the region in question. The presentations are dynamically assembled to account for the tourist’s position, orientation and, particularly, their individual cultural interests. As tourists come within a predefined range of an attraction, a presentation is automatically activated.

4.2 Genie Architecture

The Genie architecture is illustrated in Figure 5, and comprises the following agents:

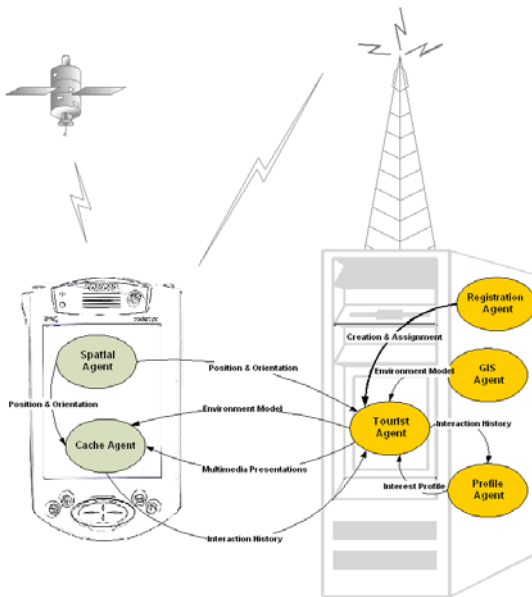


Fig. 5. The Architecture of the Genie

Spatial Agent. As its name suggests, the Spatial Agent monitors the tourist’s movement and draws some inferences about what their activity is at any point in time. It continuously monitors the position sensor, in this case GPS, and extracts both position and orientation. After verifying that the readings are accurate and consistent (a history of the tourist’s position readings is also maintained), it updates the display

and notifies the Cache Agent. It then proceeds to review this new information in the light of its goals or objectives. Obviously, one of its primary goals is to keep the agents on the server up-to-date. Therefore, if the tourist has moved a significant distance since the last update, the server agents may be notified. If not, then the agent may decide not to dispatch any messages thus conserving bandwidth. Alternatively, if the server agents have not been notified of a new position reading within a certain time frame, the agent may decide that an update is appropriate.

Cache Agent. This agent manages the multimedia cache on the PDA, a critical task given the bandwidth limitations of wireless networks as well as the memory restrictions on current PDA models. Again, the Cache Agent works in close co-operation with the Spatial Agent and the agents on the server. It relies on the server-side agents for updates to its model of the tourist's environment as well as the actual multimedia files. It also relies on regular updates from the Spatial Agent concerning the tourist's current position and orientation. By comparing this reading with its model of the attractions in the tourist's vicinity, it can fulfil its fundamental objective of displaying information that has been customised to the tourist's interest profile, at the appropriate time and place. It is assured that the individual tourist's cultural interests have been catered for when server agents dynamically assemble the presentation.

Registration Agent. The Registration Agent is responsible for administrating the agent community on the Agent Server. It allocates individual agents (Tourist Agents) to tourists seeking to register for Genie services. It also performs standard maintenance tasks and reallocates system resources upon tourists exiting the system.

GIS Agent. Providing what we term GIS - related services to Tourist Agents is the function of the GIS Agent. It is equipped with a model of the environment in which the tourist is currently roaming. Using this, it can advise on what tourist attractions, or indeed any other desired facilities, exist within the tourist's immediate vicinity.

Profile Agent. The Profile Agent maintains the tourist's cultural interest profile and advises on what content should be considered for inclusion in any multimedia presentation sent to the tourist. In particular, it dynamically updates the model in response to tourists' selections when interacting with the Genie. In this way, the model evolves over time and all information presented to the tourist is assured to be compliant with the most recent deductions concerning the tourist's interests. The user's profile, when augmented with their location and orientation, provide a rich set of filters for adapting information prior to presenting it to the tourist.

Tourist Agent. All tourists who register with the Genie are assigned their own individual agents, termed Tourist Agents. The Tourist Agent is essentially the tourist's gateway to the services provided by the Genie and maintains a snapshot of the tourist's activities at any given time. This agent, acting on information received from the Spatial Agent, collaborates with both the GIS Agent and the Profile Agent to ensure that the Cache Agent's model of the tourist's immediate environment is valid. Secondly, it ensures that all cultural presentations that may be required as the tourist continues to roam are pre-cached on the server, awaiting a download request from the Cache Agent. This content has been adapted in light of input from the Profile Agent

concerning the tourist's interests. Adaptivity of information content ensures that different users may well be presented with very different content even though they are in the same vicinity! In the case of the Genie, adaptivity manifests itself in a variety of forms. For example, a presentation could display different images dependent upon an individual tourist's direction of approach.

4.3 Implementation

The initial version of Gulliver's Genie has been realised on an IPAQ H3660 running Pocket PC. The IPAQ is equipped with a dual slot expansion sleeve that hosts the PCMCIA cards for GPS (position recovery) and GPRS (data communications) respectively. GPRS (General Packet Radio Service) is the first step in the evolution of GSM data services to 3G. In contrast to its predecessor, it is a packet-switched system and supports the IP protocol. It also supports dynamic bandwidth allocation thus making the prediction of download times impossible. While it supports data speeds of up to 30 kb/s on average, our experience indicates that these can vary quite considerably and even drop down to 9.6 kb/s, the standard rate supported by GSM. All data communications with the server use the standard Internet protocol HTTP and the Jakarta Servlet engine is used to interface with the server. In each case, Java is the programming language of choice. At present, the Java client is implemented using Jeode, a commercial implementation of a JVM for devices running Pocket PC. Sound playback is achieved using a customised version of the Java Media Framework (JMF).

4.4 Agents @ Work in Gulliver's Genie

Before discussing some examples of commitment rules that the Genie uses, it is instructive to reflect on some of the pertinent issues concerning the deployment of strong intentional agents of mobile devices. The first issue of note is that such agents are computationally expensive when considered in the light of the limited resources on PDAs. Therefore, the task(s) assigned to the agent needs careful consideration. However, at some point a trade-off will occur. Ideally, each task would be assigned a different agent. From a design perspective, this is appealing and intuitive for all the obvious reasons. In practice, this could well mean assigning what might be considered relatively trivial tasks to agents. However, as the number of agents increases, system performance will decrease in something that would be quite noticeable on a PDA. Therefore the designer must, early in the design process, make a judicious decision concerning the number of agents that can be deployed without degrading performance.

Two agents have been incorporated into the Genie MAS for deployment on the PDA. Given the importance of position to the successful operation of the Genie, the task of interfacing with the GPS device and interpreting the GPS signals has been assigned to the Spatial Agent. While the autonomous nature of the agent makes it ideal for this task, its reasoning engine is not unduly stressed. In essence, it periodically polls the device to assess the health of the signal and determine the user's new position and orientation. By recording the most recent positions, it can of course make certain deductions concerning the user's movement, for example, whether they

are walking or standing. Although another approach could have been adopted, the use of an agent ensured that this vital task could be performed in an integrated manner without affecting system performance adversely. A secondary consideration was the possibility that an alternative position determination mechanism might be used at some future point, either one based on wireless telecommunications network or one that augmented the basic GPS signal via some Satellite-Based Augmentation System (SBAS). In which case, the basic agent functionality would be the same but its logic for interfacing with the appropriate device would be different.

In the case of the Cache Agent, it has responsibility for implementing the Genie's intelligent precaching mechanism, the details of which may be found elsewhere [22]. However, as an example, consider the scenario where the agent must trigger a presentation for the user. To do this, it must first precache a presentation for the exhibit in question. Then, if the exhibit is the nearest exhibit, which can be determined from its model of the local environment, and the tourist is with in a certain predefined distance of the exhibit, calculated from the current position as indicated by the Spatial Agent, then the Cache Agent is in a position to trigger a presentation. An example of a commitment rule that achieves this is as follows:

```
BELIEF(NearestExhibit(?exhibit)) &  
BELIEF(ExhibitDistance(?delta) < (activation_radius)) &  
BELIEF(CachedPresentation(?exhibit)) => COMMIT(Self,  
Now, BELIEF(true), DisplayPresentation(?exhibit));
```

An illustration of an agent's mental state that would give rise to the activation of this commitment rule is illustrated in Figure 6.

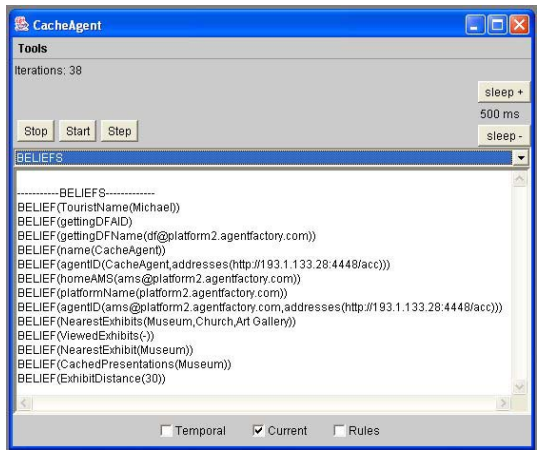


Fig. 6. Example of the mental state of a Cache Agent that triggers the display of a presentation

Finally, it's useful to reflect on the use of the Agent Factory IDE when developing the Genie, in particular those tools provided for testing and debugging. The Agent Viewer Tool proved indispensable for ensuring that the agents' mental state remained consistent. In particular, its ability to track the adoption of commitments facilitated

the quick identification of those parts of the system that were functioning incorrectly. Though Message Sender Tool was used less frequently, it proved useful for interactively testing various scenarios whenever an agent's mental state was inconsistent and the immediate cause of this was unclear. The Community Monitor was not used to any great extent as the number of agents did not warrant its extensive use.

5 Conclusion

This paper has explored one particular genre of Agent-Oriented Information Systems (AOIS), that of mobile and ubiquitous computing. Specifically, we have commissioned Agent Factory, a system for the fabrication of strong, intelligent, mobile and agile agents, in the realization of two archetypical mobile computing applications: EasiShop and Gulliver's Genie. Our experiences have shown the feasibility of supporting mobile intentional agents within a mobile computing context. Both EasiShop and Gulliver's Genie have successfully harnessed intentional agents in the delivery and deployment of location-aware and context-aware services. These systems have served as case studies vindicating the efficacy of the approach, the methodology utilised and the Agent Factory development environment.

Mobile computing applications present some particular problems from which agent based systems are typically shielded. The first is that of the heterogeneity of mobile devices, ranging from processor specification, memory availability, screen real estate, peripherals, operating system, available battery power and many more. All present specific decisions ranging from the use of J2ME or Personal Java, to the feasibility of a single or multi-threaded approach. Further to this are the restrictions and limitations of network bandwidth and latency. Where multimedia content is dynamically assembled and fetched from a remote server, these are critical performance issues. Within Gulliver's Genie, agents can and have been used in the creation of the illusion of limitless bandwidth through intelligent precaching. Within EasiShop the migration of agents to the market place gives rise to further network demands.

In conclusion, agents offer intriguing possibilities in the new ubiquitous and mobile application space. Part and parcel of this are the associated challenges that this sector presents necessitating agile, mobile agents that can potentially exhibit autonomic and self adaptive capabilities.

Acknowledgements

Michael O'Grady gratefully acknowledges the support of the Irish Research Council for Science, Engineering & Technology (IRCSET) though the Embark Initiative postdoctoral fellowship programme. Gregory O'Hare gratefully acknowledges the support of Science Foundation Ireland under Grant No. 03/IN.3/1361.

References

1. O'Hare G.M.P., Agent Factory: An Environment for the Fabrication of Multi-Agent Systems, in Foundations of Distributed Artificial Intelligence (G.M.P. O'Hare and N. Jennings eds) pp449-484, John Wiley and Sons, Inc., 1996

2. Collier, R., Agent Factory: A Framework for the Engineering of Agent-Oriented Applications, PhD Thesis, Dept. Computer Science, University College Dublin, 2001.
3. Collier, R., O'Hare, G. M. P. Lowen, T. D., and Rooney, C. F. B., Beyond Prototyping in the Factory of Agents, In Proc. 3rd Int. Central and Eastern European Conference on Multi-Agent Systems (CEEMAS), Prague, Czech Republic, 2003.
4. FIPA, The FIPA 2000 Specifications, FIPA Website URL: <http://www.fipa.org>
5. Dastani, M., van Riensdijk, B., Dignum, F., Meyer, J.J., A programming language for cognitive agents: Goal directed 3APL, In: Proc. AAMAS2003, Melbourne, Australia, 2003.
6. Rao, A., *Agentspeak(L)*: BDI agents speak out in a logical computable language, In de Velde, W., Perram, W.J.V., eds.: Proceedings of the 7th International Workshop on Modeling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, 1996
7. Rao, A.S., Georgeff, M.P.: Modelling Rational Agents within a BDI Architecture. In: Principles of Knowledge Representation. & Reasoning, San Mateo, CA. 1991.
8. Ross, R., Collier, R., O'Hare, G.M.P.: AF-APL – Bridging Principles & Practice in Agent-Oriented Languages. In: Proceedings of the 2nd International Workshop on Programming Multi-Agent Systems Languages and Tools (PROMAS-2004), New York, July, 2004.
9. Collier, R., O'Hare, G., Rooney, C.: A UML-based Software Engineering Methodology for Agent Factory. In: Proc. 16th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), Banff, Alberta, Canada, 2004.
10. Rooney, C.F.B., Collier, R.W., O'Hare, G.M.P.: VIPER: Visual protocol editor. In: Proceedings of COORDINATION 2004, Pisa, Italy, 2004.
11. Shardanand, U., Maes, P.: Social information lettering: Algorithms for automating word of mouth. In: Conference on Human Factors in Computing Systems: Mosaic of Creativity (CHI'95), New York, USA, 1995, 210-217.
12. Doorenbos, R. B., Etzioni, O., Weld, D. S.: A Scalable Comparison-Shopping Agent for the World-Wide Web. In: Proceedings of the First International Conference on Autonomous Agents (Agents'97), Marina del Rey, CA, USA, 1997, 39-48.
13. Maes, P., Guttman, R., Moukas, A.: Agents that Buy and Sell: Transforming Commerce as We Know It. In: Communications of the ACM, 42 (93), 1999, 81-91.
14. Howard, J. A., Sheth, J. N.: The Theory of Buyer Behavior. New York, John Wiley & Sons, Inc., 1969.
15. Youll, J., Morris, J., Krikorian, R., Maes, P.: Impulse: Location-based Agent Assistance. In: Software Demos, Proceedings of the Fourth International Conference on Autonomous Agents, Barcelona, Spain, June, 2000.
16. Kourouthanasis, P., Spinellis, D., Roussos, G., Giaglis, G.: Intelligent cokes and diapers: MyGrocer ubiquitous computing environment. In: Proceedings of the First International Mobile Business Conf., 2002, 150-172.
17. Fano, A.: SHOPPER'S EYE: Using Location-based Filtering for a Shopping Agent in the Physical World. In: Proc. 2nd Int. Conf. on Autonomous Agents, Minnesota, 1998, 416-421.
18. Keegan, S., O'Hare, G.M.P.: EasiShop - Agent-Based Cross Merchant Product Comparison Shopping for the Mobile User. In: Proc. of 1st Int. Conf. on Information & Communication Technologies: From Theory to Applications (ICTTA '04), Damascus, Syria, 2004.
19. Keegan, S., O'Hare, G.M.P.: EasiShop: Enabling uCommerce through Intelligent Mobile Agent Technologies. In: Proceedings of 5th International Workshop on Mobile Agents for Telecommunication Applications (MATA'03), Marrakesh, Morocco, 2003.

20. Davis, R., Smith, R.G.: Negotiation as a Metaphor for Distributed Problem Solving. In: Bond, A., Gasser, L. (eds.): *Readings in Distributed Artificial Intelligence*, 1988, 333-356.
21. United Nations Standard Products and Services Code: <http://www.unspsc.org>.
22. O'Grady, M. J., O'Hare, G. M. P.: Just-in-Time Multimedia Distribution in a Mobile Computing Environment, *IEEE Multimedia*, vol. 11, no. 4, pp. 62-74, 2004.
23. O'Hare, G. M. P., O'Grady, M. J.: Gulliver's Genie: A Multi-Agent System for Ubiquitous and Intelligent Content Delivery, *Computer Communications*, 26 (11), 2003, 1177-1187.
24. Cheverst, K., Mitchell, K., Davies, N.: The Role of Adaptive Hypermedia in a Context-Aware Tourist Guide, *Communications of the ACM*. Vol. 45 (5), 2002, 47-51.
25. Poslad, S., Laamanen, H., Malaka, R. Nick, A., Buckle, P., Zipf, A.: CRUMPET: Creation of User-friendly Mobile Services Personalised For Tourism. *Proceedings of the 2nd International Conference on 3G Mobile Communication Technologies*, London, UK, 2001.
26. Vindigo, Inc. New York. <http://www.vindigo.com>.
27. 27.Port@ble Internet, Inc. New Jersey, <http://www.portableinternet.com>.