<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Mobile intelligence: enabling a new class of context-aware services</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authors(s)</strong></td>
<td>O'Hare, G. M. P. (Greg M. P.), O'Grady, Michael J., Collier, Rem, Keegan, Stephen</td>
</tr>
<tr>
<td><strong>Publication date</strong></td>
<td>2004-07-19</td>
</tr>
<tr>
<td><strong>Conference details</strong></td>
<td>Agent Oriented Information Systems Workshop at the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS04), Columbia University, New York, 19th - 23rd July 2004</td>
</tr>
<tr>
<td><strong>Item record/more information</strong></td>
<td><a href="http://hdl.handle.net/10197/4458">http://hdl.handle.net/10197/4458</a></td>
</tr>
</tbody>
</table>

Downloaded 2024-05-24 00:00:21

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)

© Some rights reserved. For more information
Mobile Intelligence: Enabling a New Class of Context-aware Services

G. M. P. O’Hare, M.J. O’Grady, R.W. Collier, S. Keegan

Practice & Research in Intelligent Systems & Media (PRISM), Department of Computer Science, University College Dublin (UCD), Belfield, Dublin 4, Ireland.
{gregory.ohare, 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 introduced. In particular, its strategies for realising such agents in the computationally-constrained world of mobile computing are outlined. Finally, two archetypical mobile computing applications, realised through Agent Factory, are described. 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.

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, which are computationally challenged in terms of screen, memory and processor real estate. Networks that are resource bounded and bandwidth restricted. Users who 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) [16] [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 [3]. However, Agent Factory differs from other AOP offerings in that AF-APL has been embedded within a distributed FIPA-compliant [9] Run-Time Environment, and supports the development and deployment of agents through an integrated development environment, and an associated software engineering methodology (see figure 1). Details of these layers are presented in the following sections.

Fig. 1. The Agent Factory Framework (left) and its associated Development Methodology (right)

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 which 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. 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) [20], 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. Details of both the formal model and the syntax and semantics of AF-APL can be found in [2].

2.2 The Run-Time Environment

The AF-APL interpreter is embedded within a distributed FIPA-compliant [9] 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)

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 of 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 System Agents implement a number of infrastructure services that are needed to support the inter-AP infrastructure. Specifically, AF currently implements three System Agents. The AMS and DF agents are agent wrappers envelop the associated AMS and DF platform services. Both agents control access to the relevant service on the AP. However, in the context of a multi-AP environment, which is expected for most agent-oriented applications, there is a need to support the federation of AMS and DF services. This federated service is realized by a third System Agent, entitled the Super DF. The Super DF provides a common point of interaction between the various AMS and DF agents. Within the configuration of individual APs, developers are able to specify the agent identifier of a Super DF agent that they would like the AMS and DF agents to connect to. Upon start-up, this agent identifier is passed to the AMS and DF agents who register themselves with the specified Super DF. Once registered, the AMS and DF agents are able to perform global (at least within the set of APs that are connected to the Super DF) white and yellow pages searches. Ongoing research is concerned with the development of a more robust infrastructure that exhibits autonomic properties such as self-organisation and self-healing, and which contains no single point of failure.

2.3. The Development Methodology

Methodological support for the fabrication of agent-oriented applications using Agent Factory is provided through a UML-based software engineering process that supports the design, implementation, testing, and deployment phases of the software engineering lifecycle. A diagrammatical overview of this process, details of which can be found in [4], is presented in figure 1. In this diagram, it can be seen that the design stage of the process is focused around the development of 5 models:

- 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. Some typical system behaviours that might apply to a mobile computing system include: user movement updates, hotspot activation, and map generation.

- The **Interaction Model (IM)** expands on the SBM through the modeling of the interactions that occur within each of the system behaviours. Visually, this model is formalized using a customized UML Collaboration Diagram.
Specifically, objects are stereotyped as roles, and messages are restricted to valid FIPA ACL performatives via the “fipa-acl” stereotype.

- 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 behaviours. Visually, this model is formalized using a customized UML Activity Diagram where swimlanes are employed to represent roles.

- The **Protocol Model (PM)** represents a demarcation point within the methodology, in that, it represents the transition point where the focus turns from understanding the system behaviours to the formalization of those behaviours as a set of protocols and agent-classes. Specifically, the PM is derived from the IM where the agent interactions are formalized as protocols. Visually, these protocols are represented using Agent UML Sequence Diagrams.

- 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. Specifically, this model focuses upon two concepts: roles, and agent classes. The roles specify meaningful aggregations of protocols (e.g. in a mobile computing application, roles may include the Map Creator, the Interface Manager, and the PDA Cache Manager). These roles are then implemented through certain agent classes. As such, the agent classes combine one or more roles, with any additional protocols, and finally, a set of activities that are required to realize the protocols. These activities are derived from the ActM.

Further details of the visual notation employed within the AgtM and the implementation and deployment phases of the methodology can be found in [4].

### 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), illustrated in figure 2, 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 [21] 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 (section 2.3). 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. To activate the debugging environment the developer configures the
AP to operate in its debugging mode. Tools provided in the debugging environment 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.

Further details of these tools can be found in [2].

![Fig. 2. The Agent Factory IDE (left) and the VIPER Protocol Editor (right)](image)

### 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 domain. Ringo [22], for example incorporated a collaborative filtering mechanism. This approach has since been adopted in the commercial realm by vendors such as Amazon. ShopBot [7] 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 [14] strives to deliver integrated product brokering, merchant brokering, and negotiation, thus encompassing three stages of the consumer buying behaviour model as documented by Howard and Sheth [10].
Research into the role of intelligent mobile shopping systems is less mature. The Impulse project [25] developed at MIT augments GPS with agent technologies to provide context-sensitive information to the user. The main objective of the MyGrocer [13] project is to enable interactivity, personalisation and automation of home replenishment activities for products in the grocery retail sector. The Shopper’s Eye experiment [8] introduces the concept of location-based filtering to assist the shopper. In contrast to all of these, however, EasiShop [11] [12] adopts a user-centric approach, realized through a suite of strong intentional agents. Its principal objective is the delivery of a scalable architecture that enables 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 wherein the shopper is in possession of a PDA with the onboard client-side EasiShop system. To initiate the system, the user 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 GUI is composed of a series of standard HTML forms and elements. The EasiShop architecture is complimented by another aspect of the system – the server-side components. It is envisaged that participating retailers will make provision for EasiShop Hotspots (EHs). 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 3 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 utilised to provide the user with more appropriate product offerings as well as providing Store Agents with valuable information as to what type of potential customer is in proximity. It can be seen that the EH acts as a conduit between the user, represented by a personal PDA Agent, and the Marketplace.

3.2 Implementation

The EasiShop system was realised as a 7-layer architecture, as illustrated in Fig. 4. These layers are now considered briefly:

**Agent Factory:** System intelligence is delivered through Agent Factory (section 2).

**Data Structures (XML):** An adaptation of the Contract Net Protocol as proposed by Davis and Smith [6] 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 [23] has been proven to be the most appropriate. XML was selected to represent the product information because of its inherent simplicity and for the fact that it has become a de facto standard. It is extensible and separates content from presentation. Each time the user 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.
Agent 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, realtime auction resolution is imperative. Furthermore, since the dominant bidding strategy of the Vickrey protocol is to bid to one's true evaluation, counterspeculation 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 3 or more bidders, while both English and Vickrey auctions provide higher revenue for the auctioneer than their Dutch and first-price sealed bid counterparts.

EasiShop agents provide a level of dynamism in their choice of protocol, in an attempt to provide the best possible terms for the user. While the default protocol is set to Vickrey, it may be decided as the knowledge base increases over time that another protocol is more suitable - either for a particular category of product or for all categories. The system design supports such dynamic alterations.

To allow optimum agent autonomy, the concept of a decommitment penalty has been proposed. This allows the agent, when given sufficient permission by the user, to enter and bid in auctions of interest. Should the user be unsatisfied with any purchase then the user is allowed to cancel the payment. With this process, the user effects a small penalty payment.

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 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.

4  Gulliver’s Genie

Gulliver’s Genie [15] [17] 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 [1], a context-sensitive tourist guide for the city of Lancaster and CRUMPET [19], developed for the city of Heidelberg. Examples of commercial products include Vindigo [24] and Portable Guide [18]. 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 MAS, the context of which envelopes the Internet, a wireless data network and PDAs, and comprising a suite of intelligent agents all collaborating to obtain the Genie’s goal, namely the efficient and 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

![Genie Architecture Diagram](image)

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

**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 cooperation 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. Fig. 6 illustrates an example of a presentation that the Genie might provide to a tourist or indeed, a prospective student.
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 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).

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 ubiquitous computing context. Three key principles underlie the operation of both EasiShop and Gulliver’s Genie, namely: adaptivity, implicit interaction, and intelligent content/service delivery. Adaptivity arises from the customisation and personalisation of content based upon user preferences and device capabilities. Thus information presented to both the tourist and the shopper is adapted based upon their position, orientation, interests, and device. In this way, we maximise content relevance presented to the user while simultaneously minimising the data transmitted to the PDA. Implicit interaction enables non-intrusive interaction modalities to drive system behaviour. For example, a user movement can express subtle and occasionally obvious preferences. Finally, intelligent content/service delivery proactively anticipates user needs thus delivering relevant and timely content. All three principles are realized through the judicious use of an Agent-Oriented Information System.

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