Whitestein Series in Software Agent Technologies, 179–206

Agent Applications in Tourism

Antonio Moreno

Abstract. Agent technology has been applied in recent years to solve different problems that are common to many applications in Tourism, such as dynamic service discovery, automatic management of user profiles, personalisation of cultural information or planning of touristic activities. This chapter shows different contributions of Spanish research groups in the following areas: personalised access to cultural information from mobile devices, planning of complex touristic activities, service discovery in Tourism applications and dynamic location tracking.

Keywords. Tourism, mobile devices, user profile management, personalisation, recommendation, service discovery, planning, dynamic location tracking.

1. Introduction

Agent technology is especially amenable to be applied to domains in which information is physically distributed and a set of autonomous entities have to join their efforts and coordinate their activities to solve a complex task. Tourism is a domain with such characteristics, as a tourist needs to search for information related to the cultural and leisure activities available in a given city (which is usually distributed in different places all around the city, as it depends on different stakeholders), filter those that fit with his personal interests, and try to build a plan in which the selected activities may be performed within a given time span. There are diverse research fields within agent technology that can be directly applied to the provision of cultural information to tourists:

- Implementation of agents in mobile devices.

  Tourists are keen on accessing cultural information directly from their mobile devices, at any point of the city at any time, without having to go to tourist offices or specific places in the city.

Co-authors of this chapter: Javier Bajo, Vicente Botti, Juan M. Corchado, Arantza Ilarramendi, Sergio Ilarri, Vicente Julián, Miguel A. López Carmona, Iván Marsá, Eduardo Mena, Antonio Moreno, Juan Pavón and Aïda Valls.

- Automatic management of user profiles, and personalised proactive recommendations.

  It is very interesting that systems that provide information to tourists are able to automatically learn and maintain a profile with the interests of every particular user, so that they can filter the cultural information that is relevant for each person and can offer that information proactively to the user at the appropriate moment.
- Dynamic discovery and access to tourist e-services.

  Tourists should be able to easily discover the available e-services and be able to access them in a transparent way.
- Planning of touristic activities.

  An intelligent agent-based system with planning capabilities may help a tourist to select those activities that are more relevant for him during his stay in a city and to arrange a temporal sequence of movements within the city in order to optimize the time that the tourist has to enjoy his holidays.

The following sections of this chapter provide some examples of agent-based systems related to the Tourism domain which have been developed by Spanish research groups in recent years:

- `Turist@` is a multi-agent system developed at University Rovira i Virgili that is focused on the intelligent personalised proactive recommendation of cultural activities to tourists visiting Tarragona.
- `TourAgent`, developed at the Technical University of Valencia, allows users to make structured searches of tourist information, make reservations in restaurants, and plan the activities in a given day.
- The University Complutense of Madrid and the University of Salamanca designed, developed and made a trial with real tourists of an agent-based system that uses Case Based Reasoning techniques to provide personalised plans to tourists visiting Salamanca.
- The University of Alcalá has proposed the idea of having a hierarchy of *smart spaces* in order to approach the problem of dynamic service discovery.
- Finally, `Loqomotion` is a system designed at the University of Zaragoza that solves the problem of dynamic location tracking in Tourism applications.

2. `Turist@`: agent-based proactive and personalised recommendation of cultural activities

Big cities attract tourists due to the large amount of activities that they offer. However, despite being an advantage, that huge offer also has a negative side: the tourist must select the activities he wants to do from an immense set.

The multi-agent system `Turist@`, developed at University Rovira i Virgili (URV, Tarragona), tries to remedy this difficulty. The system is context-aware, and the user may receive touristic information at any point of the city by interacting with a *Personal Agent* that is executing in his mobile phone. The system keeps a

dynamic profile of the interests of each user, and uses both fuzzy logic and novel content-based and collaborative recommendation techniques to make personalised and proactive suggestions of cultural events that may interest the user.

The system is composed of a set of agents that have information about different types of cultural activities, such as museums, itineraries, conferences or exhibitions. This idea makes the approach very scalable, since it is very easy to dynamically add agents that manage new kinds of events. Figure 1 depicts the structure of `Turist@` (showing only a subset of the cultural activities). In addition to the agents that manage different types of events, there is a *Broker Agent* that facilitates the mediation between the *Personal Agent* (which belongs to each tourist that logs into the system) and the cultural activities agents, and a *Recommender Agent* responsible for making (on demand or proactively) personalised recommendations to the users as well as maintaining a database with their profiles. As will be described below, both content-based and collaborative techniques are used to make the recommendations.

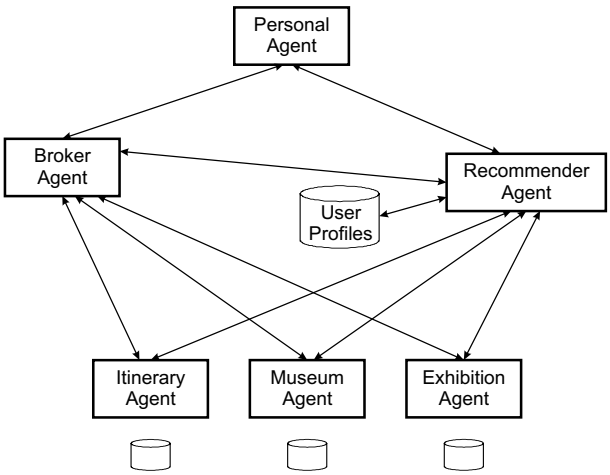

FIGURE 1. `Turist@` multi-agent system architecture.

2.1. Personalised recommendation of cultural events

The main feature of `Turist@` is the existence of a personalized recommendation system. The purpose of this system is to select, from a large set of activities, the ones that are most suitable for a particular person according to his personal characteristics and preferences. Each activity is described by a set of variables that include descriptive information about the activity, such as the place, timetable or price, and information about the features of the activity that can match with the interests of the tourist. To make this matching, the *Recommender Agent* keeps a dynamic profile of the user's interests.

An activity is represented with a vector a_i, in which each component indicates to what extent the activity fits with a specific property (e.g., the artistic or historical relevance of the activity). In the same way, the user's profile u_i is described in terms of numerical preference degrees with respect to the variables that describe the activities.

Any recommender system must consider at least the following three issues: (1) how to initialize the user's profile,(2) how to use the profile to make recommendations and (3) how to adapt the user's profile dynamically (explicitly vs. implicitly).

The user profile in `Turist@` is initialized with the information provided by the user the first time he accesses the system. One of the weaknesses of recommender systems is that people usually do not want to spend much time providing data to computer applications. To ease this task the tourist is allowed to describe his interest in each of the variables considered in the system by selecting a linguistic term from a fixed vocabulary: {*none*, *little*, *medium*, *quite*, *a lot*}. The system translates internally those terms into numbers in the [0,1] interval using a fuzzy logic approach.

In addition to the preferential features, there are demographic variables that can be either *categorical* or *linguistic*. For example, the academic level can be selected from an ordered range of linguistic terms {*none*, *basic*, *graduate*, *university*}, whereas the spoken languages are chosen from a categorical set like {*Catalan*, *Spanish*, *English*, *French*, *German*}. These values are taken from this closed set and stored in the profile.

The methods that make automatic recommendations can be classified in two broad paradigms [26]:

- *Content-based approaches*: the recommended items are the ones that match with the information stored in the personal profile.
- *Collaborative approaches*: the system recommends to a user those items that have been selected by other people with a similar profile. Thus, those approaches focus on the similarity between the users, rather than on the similarity between the items.

A pure content-based system has several shortcomings. Considering only the similarities in the features stored in the personal profile, one can omit other aspects of the items that also influence user decisions. Moreover, the user is restricted to seeing items similar to his profile, without having the opportunity to explore new activities, with different unexplored features. The pure collaborative approach can deal with any kind of content and the user can receive items with content dissimilar to those seen in the past. However, this pure approach has other drawbacks: a large number of users is needed to be able to start making recommendations; when a new item appears, there is no way to recommend it before any user has selected it; and for a user whose tastes are unusual compared with the rest, it will be difficult to find similar users, which will lead to poor recommendations. By combining content-based and collaborative techniques, one can eliminate some of the weaknesses

of each approach. Turist@ applies a content-based and a collaborative method separately and then aggregates the results to obtain the final list of recommended items.

Turist@ makes content-based recommendations using only the user's preferences stored in the profile. A similarity measure is used to compare the user's profile u_i with each activity profile a_i. Several classical similarities were implemented within the system.

When a minimum number of user profiles has been stored in the database, the *Recommender Agent* can start to make collaborative recommendations. Unsupervised clustering techniques are used to generate a partition of the users into clusters of users with similar tastes [27]. However, those clusters must be periodically updated in order to take into account the changes in the user profiles. The clusters are computed considering the users' preferences and also the demographic variables. To make a recommendation to a tourist, the system selects those items that have been positively evaluated by the people that belong to the same cluster as him.

Finally, the automatic adaptation of the user's profile has also been studied in Turist@. Both explicit and implicit adaptation procedures have been incorporated into the system.

- *Explicit profile adaptation*: the user indicates which of the recommended activities he likes and which day he will do it. When the system detects that the activity has been finished, it sends a message to the user asking for an evaluation of this activity. The user must pick up one of the terms {*horrible*, *bad*, *good*, *very-good*, *excellent*}, which is used by the system to modify the vector of user's preferences accordingly (reducing or increasing the preference value stored in the profile).
- *Implicit profile adaptation*: this is done when the tourist makes searches in Turist@ through the *Broker Agent*. The search parameters are used to detect if the user is looking for activities with interests in features different from the ones that he used to be interested in. If that is the case, his profile is updated. The change in the user preferences depends on the actual preference values in the profile and the main characteristics of the activity (e.g., if the user searches for art exhibitions, his interest in the *art* feature will be slightly increased).

2.2. Dynamic location of users

To provide touristic information to users, Turist@ features a *Personal Agent* that is executed in a mobile phone. The main advantages of this technology in the Tourism domain are:

- It is not necessary for tourists to acquire new specific hardware to access the system.
- The tourist can receive at any place information on new activities that are uploaded to the system.

- The system can know the position of the tourist in the city and can provide personalised information about the sites of interest that are near this position.

This agent is implemented using JADE-LEAP. To know the position of the tourist, the GSM technology is used. The reasons for not using GPS are the following: it is not always possible in a city to locate the satellites needed to calculate the position (for example, if the device is in a pocket or inside a building), it requires some start-up time, and not all the phones include this feature.

The GSM network is the natural way of communicating in mobile phones, so no extra device is needed. This network divides the territory into small cells of some hundreds of meters, which is a good approximation to the location of the tourist inside the city. This precision is necessary and sufficient to know what interesting cultural activities are available in the vicinity (inside that cell). So, each time that a new activity is loaded to the system, it is located in one of those cells, which are uniquely identified in the GSM network. To work in this way, the *Personal Agent* must read the configuration of the cells in the city that the user is visiting before starting to use the system. Figure 2 shows the interface of the *Personal Agent* when the user is looking at his position on the city map.

Once the position of the user has been automatically discovered by the agents, the possibility of making proactive recommendations is straightforward. The *Recommender Agent* is notified about the position of the tourist, and it uses a similarity measure to filter which of the activities in the corresponding cell coincide with the preferences of the person. If any activity matches, it is automatically recommended to the tourist.

FIGURE 2. Visualizing the user location.

2.3. Integration of `Turist@` components

The different aspects of the `Turist@` system (the basic multi-agent system, the *Personal Agents* running in mobile devices, the recommendation techniques and the planning capabilities) have been developed in the last four years independently, mainly by Computer Science students at URV. This temporal span has fostered the use of different versions of the underlying development framework (JADE and

JADE-LEAP). Currently all the different parts are being integrated in a single system, and trials of the whole system in the city of Tarragona are scheduled for summer 2008.

3. TourAgent

The main goal of this research is to develop an open system, capable of incorporating as many agents that can provide useful services to the tourist as necessary. To provide personalised services, the system needs to identify tourists in a transparent and ubiquitous manner. In this sense, the growing use of handheld devices is a great opportunity to interact with the clients in a simple way and to manage the tourist profiles.

The `TourAgent` system [25] is a multi-agent system architecture which offers services in the Tourism industry. It gives the possibility to different users (mainly tourists in a city) to obtain up-to-date information about the places they will visit and to plan activities in a specific day. Users can access and employ these services using a Java-enabled mobile phone or PDA.

From a tourist point of view, it would be of great help to have all the updated information, in a dynamic and flexible way, about the different places of interest in the city, such as a cheap restaurant, or a restaurant that serves a particular type of food, giving them the opportunity to make reservations in those restaurants; or if they want to know if there is any art showroom open; or simply know to which cinema they can go and receive information about which movies are playing. Because of all of these aspects, the advantages of having a resource for tourists that allows them to search and plan different activities of interest during a given day are noticeable.

3.1. System architecture

This section presents the developed application in which the applicability of agent technology integrating different devices has been evaluated. In the implemented version of this system, a tourist can find information about places of interest like restaurants, cinemas, museums, theatres and other places of general interest like monuments, churches, beaches, parks, etc., according to his preferences using his mobile phone or PDA. Once a specific place has been selected, the tourist can establish a process to make a reservation in a restaurant, buy tickets for a film, etc, in a given time period proposed by the tourist.

The system is basically formed by four classes of agents: the *BrokerAgent*, the *UserAgent*, the *SightAgent* and the *PlanAgent*. The main functionalities of these agents are:

- The *BrokerAgent* has updated information about the registered *SightAgents*. It is in charge of establishing a communication between the user and the *SightAgents*.

- The *SightAgent* manages all the information about the characteristics and activities of one specific place of interest in the city. It is exclusive to one place of interest and has all the information efficiently up-to-date.
- The *UserAgent* allows the tourist to use the different services by means of a GUI on his mobile device. This agent is exclusive to one interested tourist.
- The *PlanAgent* is in charge of establishing and managing all the planning processes offered by the system, taking into account the preferences previously established by the users, and/or different searches users made previously within the system.

As agents need to communicate with each other, it was indispensable to establish a common conceptual vocabulary as a representation of the information for establishing and controlling tasks. The implemented ontology gives detailed descriptions of touristic places, contains information about scheduling (necessary for the planning service), etc. Besides, the developed ontology has some actions and predicates that help to control and establish tasks in the system.

Figure 3 shows the architecture of the implemented MAS system, illustrating the communication between the different agents commented above.

3.2. Basic functionalities

In the proposed system, if the user (tourist) is anywhere in the city, using the GPRS connection he can take advantage of the tourism information system, sending questions or actions to the *BrokerAgent*. The *BrokerAgent* will interact with the corresponding *SightAgents* trying to find the desired information about the sites that fall into the parameters the user has chosen. Then, the *BrokerAgent* will send the appropriate answer to *UserAgent*. After the user gets an answer to his search, he will establish automatically a communication with the corresponding *SightAgent*. Users can establish the required connection using GPRS. The wireless option can be set up and established (from a PDA or a smartphone supporting this kind of connection) to access the system, if the user has the proper infrastructure to do it. The main functionalities of the system are the following:

Search (see Figure 4). This service is offered by the *BrokerAgent* and it is invoked by the *UserAgent*. In order to be employed, the *UserAgent* must send a *Request* message (according to FIPA-ACL) with the preferences as the message content. The result of this service will be a list formed by all the places that match the user requirements containing the relevant information about each specific site. To check the detailed information of a specific place, the user can perform a search by different types of places like restaurants, cinemas, museums, theaters, and other places of general interest like monuments, churches, beaches, parks, etc. This search can be filtered according to the user's preferences. As a result of this service, the *UserAgent* will receive a list of elements that fulfil its search parameters. By choosing one of the elements of the list, the *UserAgent* will get detailed information about the place.

Reserve (see Figure 5). This service is offered by the *SightAgent* and can be invoked by the *UserAgent*. In order to be used, the *UserAgent* must send a *Propose*

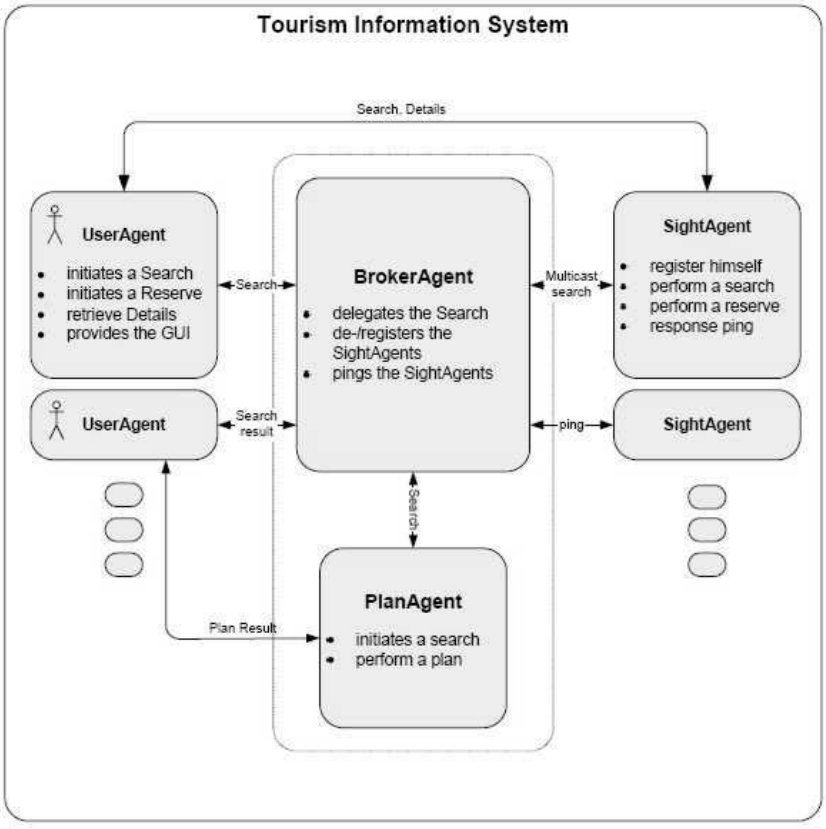

FIGURE 3. `TourAgent` general architecture.

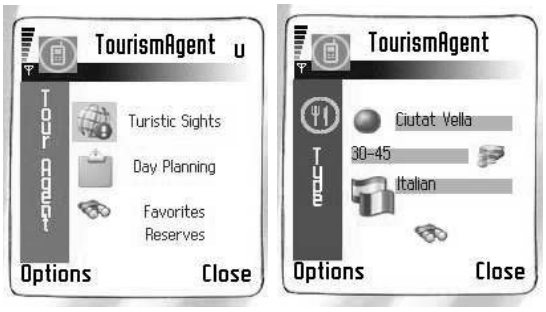

FIGURE 4. Main options view (left) and basic restaurant search options (right).

FIGURE 5. Restaurant information view (left) and reservation options (right).

message to the corresponding *SightAgent* with the terms of the reservation (only restaurant reservations are supported). The result of this service will be a successful reservation process or an error message. If the reservation is not possible, it starts a negotiation process where the *SightAgent* will try to find an alternative time or date where it can accomplish the reservation request of the user. The *UserAgent* can finally accept or refuse the new proposal.

FIGURE 6. Main plan options (left) and tentative plan result (right).

Plan a Specific Day (see Figure 6). This service is offered by the *PlanAgent* and can be invoked by the *UserAgent*. In order to be used, the *UserAgent* must send a *Request* message to the *PlanAgent* with some parameters to take into account for the plan. The result of this service will be a list of places or activities forming the plan. This important service makes it possible for users to plan a specific day with a series of activities where reservations for lunch, dinner or both can be included. The *PlanAgent* tries to arrange the activities in such a way that time can be managed efficiently throughout the day.

3.3. Implementation details

The Tourism Information System has been implemented using JADE. The agents running on the mobile devices were implemented using JADE-LEAP. The graphic interface (GUI) was developed using J2ME, that combines the Mobile Information Device Profile (MIDP) with the Connected Limited Device Configuration (CLDC). Different controlled experiments were conducted to evaluate different parameters in order to assess the proposal. The main set of experiments investigates the performance of the system according to the number of tourists requesting services at the same time. As an example, Figure 7 illustrates how the average response time increases in proportion to the number of agents. However, the system maintains a good performance, taking into account the great number of services requested concurrently in each test.

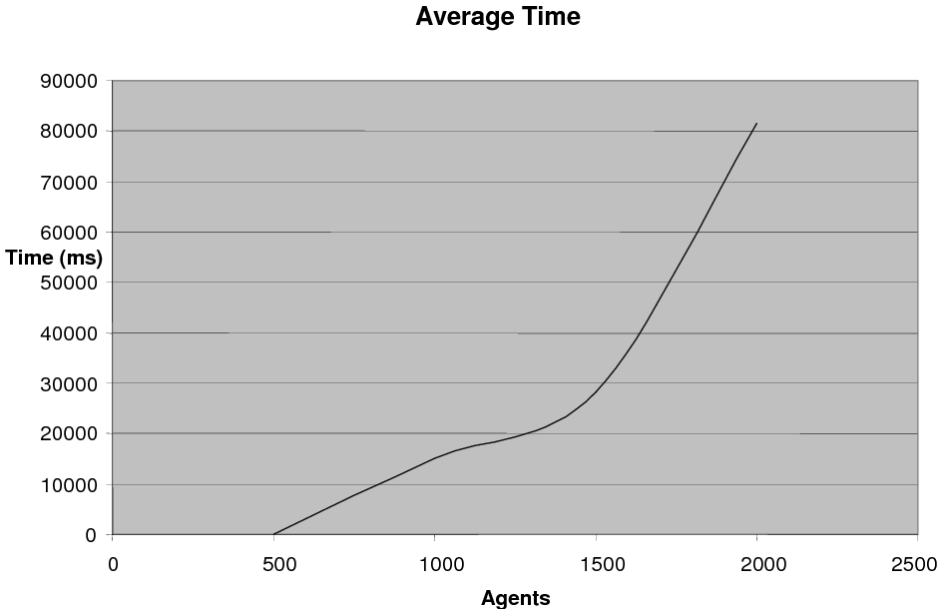

FIGURE 7. Average time test.

4. A CBR-planning based Tourism application

Agents are usually classified depending on the set of capabilities that they support, such as autonomy, reactivity, proactivity, social ability, reasoning, learning, and mobility, among others [11]. One of the possibilities is the development of deliberative agents using *case-based reasoning* (CBR) systems, as a way to implement adaptive systems in open and dynamic environments. Agents in this context

must be able to reply to events, take the initiative according to their goals, communicate with other agents, interact with users, and make use of past experiences to find the best plans to achieve goals.

Deliberative agents are usually based on a BDI model [9], which considers agents as having certain *mental attitudes*: Beliefs, Desires, and Intentions (BDI). Under this model, agents have a *mental state* that consists of informational, motivational, and deliberative states respectively. Case-based reasoning systems solve new problems by adapting solutions that have been used in the past. A classical CBR reasoning cycle consists of four sequential phases: retrieve, reuse, revise, and retain [1]. The structure of the CBR system has been designed around the concept of a *case*. A case is made of three components: the problem, the solution, and the result obtained when the proposed solution is applied [1]. The CBR-BDI proposal defines a direct mapping from the concept of an agent to the reasoning model. In this model, the CBR system is completely integrated into the agent's architecture. The proposal is also concerned with the agent's implementation and presents a "formalism" which is easy to implement, in which the reasoning process is based on the concept of *intention*. In this model, intentions are cases, which have to be retrieved, reused, revised and retained. The CBR-BDI architecture facilitates learning and adaptation, and therefore a greater degree of autonomy than with a pure BDI architecture. This is made by mapping the three mental attitudes of BDI agents into the information manipulated by a CBR system. This direct mapping between the agent's conceptualisation and its implementation is the main difference with respect to other proposals that have also tried to combine BDI and CBR [2, 7, 8, 10].

Planning can be defined as the construction of a course of actions to achieve a specified set of goals in response to a given situation. The classical generative planning process consists mainly of a search through the space of possible operators to solve a given problem, but for most practical problems this search is intractable. Given that typical planning may require a great deal of effort without achieving very good results, several researchers have pursued a more synergistic approach through generative and case-based planning [2]. In this context, the case indexation strategy facilitates and speeds up the planning process substantially. A case in case-based planning consists of a problem (initial situation and set of goals) and its plan. Given a new problem, the objective of the retrieval and reuse phase is to select a case or a number of cases from the case-base whose problem description is most similar to the description of the new problem and to adapt it/them to the new situation. In case-based reasoning, two different approaches to reuse can be distinguished: transformational and derivational adaptation. Transformational adaptation methods usually consist of a set of domain dependent concepts that directly modify the solution that was obtained in the retrieved case. For derivational adaptation, the retrieved solution is not modified directly, but is used to guide the planner to find the solution.

There are different ways to integrate generative and case-based planning. The method chosen in the work described in this section is the *Variational Calculus*

Based Planner (VCBP) [5]. Although VCBP is domain dependent, it introduces a new interesting strategy to efficiently deal with the adaptation stage. Variational Calculus-based Planner guarantees the planning and re-planning of the intentions in execution time. This planning strategy is divided into two steps:

1. identify cases that are similar to the problem case (retrieval stage), and
2. adapt them to the problem case (reuse stage).

Variational calculus automates the reasoning cycle of the BDI agents, and guarantees the identification of an efficient plan, close to the optimum. Although different types of planning mechanisms can be found in the literature, none of them allows replanning in execution time, and agents inhabit changing environments in which replanning in execution time is required if goals are to be achieved successfully in real-time.

The proposed system has been used to improve an agent based wireless system developed for guiding tourists around the city of Salamanca (Spain). The integrated, multi-platform computer system is composed of a guide agent (*Planner Agent*) that assesses the tourists and helps them to identify tourist routes in a city with a given visiting time period and under a number of restrictions related to cost, tourist interest, etc. There is one assistant agent for each user of the system, the *Performer Agents*. Each user willing to use the system has to register and solicit one of these agents. Finally, there is a third type of agent, the *Tracker Agent*, which maintains updated information about the monuments, the restaurants, public transport conditions, etc. This agent maintains horizontally and vertically compiled information on hotel accommodation, restaurants, the commercial sector and transport, in order to meet the needs of the potential visitor on an individually customized basis, and responds to requests for information, reservations and purchases as soon as they are expressed.

The user may decide whether to install the corresponding *Performer Agent* on a mobile phone or PDA, or run it on the server and interact with it via its mobile device. Users may interact either with their *Performer Agents* installed in their wireless devices or in an internet server. The *Performer Agents* interact with the *Planner Agent* looking for plans, and the *Tracker Agent* interacts with the *Planner Agent* to exchange information. The *Planner Agent* is the only CBR-BDI agent in this architecture. The *Performer Agents* can be considered assistant agents and the *Tracker Agent* is a reactive agent. Tourists may use a mobile device to contact their agents and to indicate their preferences (monuments to visit, visits duration, dinner time, amount of money to spend, etc.).

The *Planner Agent* is the only deliberative agent in this system. This agent deals with *cases*. There are different types of cases. The cases store information about the environment, for example the opening and closing times of monuments. This type of information can be seen as an agent belief, for example, the Museum of Contemporary Art opens from 9:00 to 14:00 and from to 16:30 to 20:00. Cases may also be previous successful routes (plans), as shown in Figure 8(a), that includes the monuments to visit, the time to spend visiting each monument, information

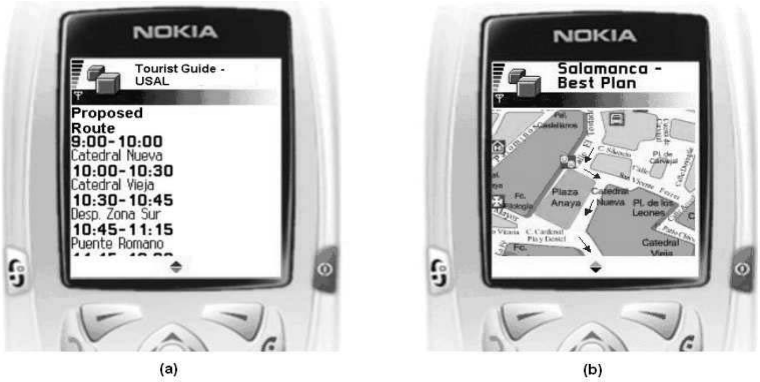

FIGURE 8. Textual and graphical presentations of plans.

about the cost of the visit, the time required for going from one place to another, the characteristics of the route (museum route, family route, university route, roman route, gothic route), etc. Once a tourist contacts the system he has to describe his profile, select the type of visit in which he is interested, determine how much money he wants to spend and for how long, and the type of restaurants he prefers. This information is used to construct the *problem case*. Then the reasoning mechanism of the planning agent generates the plan. This reasoning mechanism is the previously mentioned CBR system using VCBP [5, 6].

The *Planner Agent* generates a plan that fulfils the given conditions. This plan is easy to modify at execution time if the user changes his mind. The retrieval stage must be carried out using a method that guarantees the retrieval of a reasonably small number of cases that are related to the current problem case. A number of different retrieval methods have been analysed, such as Sparse Kernel Principal Component Analysis [3] or a K-nearest neighbour algorithm based strategy [6]. The best results have been obtained with a variational calculus based strategy. Once the most similar cases have been retrieved, VCBP adapts them to the problem case (reuse stage). Basically, for the solutions (plans) corresponding to the similar retrieved cases, the following procedure is executed. The new optimum plan is constructed in such a way that the planner proposes the plan in sections. The optimum plan is the one with a greatest density of plans around it, that is, the one that offers the best alternative for replanning if an interruption happens. Figure 8(b) shows a graphical view of a generated plan. The plan is presented in sections to the user. If the plan is interrupted, the user can choose the replanning option.

As can be seen in [4], the system was tested during 2003. The system needed initial knowledge, so the case base was initially filled with information collected during a recent five month period. Local tourist guides provided the agent with a number of standard routes. Three city hotels offered the option to their 6217

guests to use the help of the agent or a professional tourist guide; 14% of them decided to use the agent based system and 23% of them used the help of a tourist guide. The rest of the tourists visited the city by themselves. On arrival at the hotel the tourists were asked to evaluate their visit and the route. The tourists that used the help of the agent-based tourist guide provided the answer directly to the agent. The system was tested for 135 days and the obtained results were very encouraging [4].

5. A hierarchical approach to service discovery in touristic *Smart Spaces*

One of the technical solutions to the problem of service discovery from mobile devices is defining a Service-Oriented Architecture where physical locations are given a key role. In particular, SETH (Smart EnvironmenT Hierarchy) approaches this problem by defining a hierarchical, modular architecture for smart spaces, which are specific, self-contained locations within the environment able to adapt themselves to the user needs and to provide customized interfaces to the services available at each moment. There are vastly different research lines regarding service provision in smart environments, like i-Room [30] or Gaia [29]. Some of them use multi-agent systems, as they have been revealed as a good technology for developing distributed, autonomous, and intelligent systems [28]. The hierarchical approach devised in SETH allows one to create complex smart environments by combining, for instance, a certain number of smart rooms to create a smart building and a certain number of smart buildings and smart outdoor spaces to create a smart touristic city. Detailed description of the SETH architecture is beyond the scope of this chapter, and can be found in [14]. In this section the most relevant characteristics of the architecture needed to follow the rest of the description are outlined. Then, the section focuses on the mechanisms for service discovery and access to services.

5.1. SETH: a hierarchy of *Smart Spaces*

SETH relies on the concept of *smart spaces* (SSs), which are specific, self-contained locations within the environment. From a functional point of view, a given *smart space* is characterized by a set of devices, a set of available services, and a given context. Smart spaces may be hierarchically arranged if the specific characteristics of the environment require so. This hierarchical approach allows one to provide different layers of services, context information, and security. The demonstration scenario which will be referred to through the rest of the section considers a *city* smart space, which contains four indoor smart spaces, *home*, *restaurant*, *hotel* and *touristoffice* and an outdoor smart space *monument*. The *hotel* smart space contains the *secondFloor* space, which also contains the *hotelRoom* and *meetingRoom* spaces.

Inheritance rules may be established in the hierarchy to govern from which context information, services and devices from higher levels in the hierarchy are available at a specific location. Aggregation rules may also be established, so that a smart space may export context information, services and devices to other spaces located at higher levels in the hierarchy. Inheritance and aggregation rules may be combined to allow, for example, that users within the *home* space have access through inheritance to the reservation service, which is provided at the *restaurant* space, but has been made available to users in the *city* space by means of aggregation.

5.2. SETH devices

To meet its goals, the architecture relies on a set of devices distributed throughout the environment. The *Smart Space Agent Platform* (*SSAP*), mandatory in any SETH smart space, contains the agent platform which supports the existence of all other agents in the smart space, hosting the higher-level agents of the system and also those agents used to control non-intelligent devices. *Devices with Agents* are sensors and effectors with a certain degree of autonomy, usually provided by agents running over an embedded Java Virtual Machine. *Devices without Agents* are sensors and effectors without autonomy or intelligence, controlled from the SSAP. Furthermore, each user must carry a portable *Identification Device*, which is used to identify the user and determine the user location within the smart environment. Finally, users may carry handheld, mobile *Personal Devices* (cell phones, PDAs) which not only may provide the functionality of the identification devices above, but may also host the necessary agents to learn, maintain and try to satisfy user preferences, and to display adequate interfaces to the available services when needed.

5.3. SETH software agents

In a typical SETH smart space, different kinds of software agents may be found. The *Smart Space Coordination Agent* (*SSCA*), residing in the SSAP, provides device, service and context discovery to all other users or agents in the smart space, and to SSCAs of other smart spaces. *Device Agents* provide a common interface to devices, so that other agents in the system may use them regardless of specific hardware issues. *System Agents*, like context and security agents, reside in the SSAP, and add an additional layer of intelligence on top of the devices in the environment through control and coordination mechanisms. *Personal Agents* (*PAs*), usually residing in the user personal device, are the very representatives of users to the environment, since they are in charge of enforcing user preferences. Finally, *Service Agents* are intended to provide services directly to the user, and they may be *persistent*, if they are always active at a given SSAP, and *non-persistent or mobile*, if they are created by the SSCA for each request of the service, move from one SSAP to another when user location changes, and are destroyed once the use of the service has concluded. For the *Service Agents* to be able to provide their services to the

users, the *Personal Agents* need to be aware of the services available at every moment. This is where the problem of service discovery appears.

5.4. Service discovery

Since *Personal Agents* are transported within the user personal devices, they move through the different SSAPs following the movements of their associated users to provide personalized services at each location. But in order to provide this personalization, PAs need to know which are the available services, and how to access them. Service discovery functionality is provided by the *Smart Space Coordination Agents* (SSCAs). As the coordinator of a given smart space, an SSCA must be aware of all agents present in the space, and all the services they may provide. This may be accomplished through any registration process, such as the ones provided by CORBA, the IEEE FIPA DFs [15] or by architectures based on Web Services. In this way, the SSCA knows all devices, agents and services available at its associated smart space. The address of the SSCA of a given smart space is provided by the context agents when the *Personal Agent* enters the associated SSAP, so the PA may know the addresses of all relevant SSCAs in the hierarchy.

At any given time, a PA can query the SSCA at its SSAP for a list of available services. Queries may be general or specific (i.e., request a list of all available services which meet certain characteristics). The returned list of services includes the name of the service, the service agents that provide them, and the service description, which in turn contains the information needed by the personal agent to know how to access the service.

As stated above, services may be inherited from higher levels in the hierarchy or aggregated from lower levels. Services may be inherited or aggregated at the SSCA level or at the personal agent level, and thus service discovery may be *SSCA-driven* or *PA-driven*. Inheritance or aggregation at the SSCA level occurs when an SSCA is interested in providing a service available at another SSAP. In this case, the SSCA adds the service to its list of available services, providing the address of the agent which provides the service at the remote SSAP. Service inheritance at the SSCA level is provided automatically, that is, all SSCAs query regularly their upper level SSCAs to see which services are available for inheritance. Service aggregation is provided by a subscription mechanism. Lower-level service agents subscribe to higher-level SSCAs to have their services made available to users at the upper levels of the hierarchy.

In scenarios with many levels in the hierarchy, automatic inheritance and aggregation may result in huge lists of available services. This is solved by limiting automatic inheritance and aggregation to a small subset of services, and providing extended search services at the SSCAs only when specific service discovery queries yield void results (i.e., a given service is not found within the list of available services, and the search is propagated to selected SSCAs at higher and lower levels to see if the service may be provided at another SSAP). In `SETH`, this is called *SSCA-driven service discovery*. *PA-driven service discovery* can be performed at any time by issuing direct queries from the PA to the corresponding SSCAs.

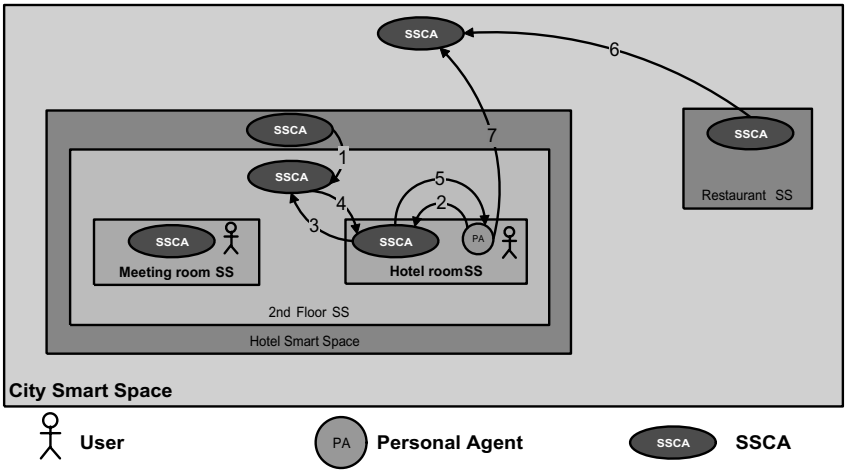

FIGURE 9. Inheritance, aggregation and service discovery.

To illustrate the mechanisms described above, we can refer to a typical use case shown in Figure 9. User *Alice* is in her hotel room, and her associated PA knows she has an appointment to have lunch with user *Bob* at the restaurant in an hour. The *Personal Agent* decides to notify *Bob*, but it does not know where he is. We can see examples of service aggregation, service inheritance and service discovery in the following process:

- *Service Inheritance:* there is a user location service at the *hotel* smart space which is provided at the floor level (1).
- *SSCA-driven Service Discovery:* to locate *Bob,* Alice's personal agent queries $SSCA_{hotelroom}$ for a location service (2). It has no such service in its list, but it propagates the query up in the hierarchy (3), receives from $SSCA_{secondfloor}$ (4) the requested information, and finally forwards it to the PA (5).
- *Service Aggregation:* the *SSCA* at the restaurant has eventually advertised its reservation service to $SSCA_{city}$, which has aggregated it to its list of available services (6).
- *PA-driven Service Discovery:* after being able to locate *Bob* and remind him of the meeting, *Alice*'s personal agent decides to make a reservation in a restaurant. None of the SSCAs of the building has inherited that service, nor are they willing to propagate service discovery requests for this kind of services (e.g., it may be against their business policy). So, perhaps after a series of unsuccessful searches at floor and hotel level, *Alice*'s personal agent will have to send its query directly to $SSCA_{city}$ (7) to find the agents who are providing restaurant reservation services.

5.5. Access to services

To access a given service, a PA only needs to send a request message to the agent providing that service. The *Service Agent* will then attempt to provide the service, usually by requesting actions to other service, system or device agents. However, the process may be slightly more complex depending on the kind of service requested. As we mentioned in Section 5.3, there are services directly related to each specific smart space, such as climatization, lighting, or interfaces which are required to be available at any moment to every user in the space, and so they are provided by *persistent* service agents. These agents are always active in the SSAP, and their addresses are specified in the service lists returned by the SSCA, so that any PA can request services directly to these agents at any time.

However, there are other services, such as content access or unified messaging, which are more directly related to the user who requests them, and they are provided by *non-persistent* service agents, which are created for each specific request for the service and destroyed once the service has been provided. The SSCA uses a special address value in the list of available services to indicate which services are provided by non-persistent agents. If a PA wants to access a service provided by a non-persistent agent, it must first request the *SSCA* to create the agent. The non-persistent agent is created and its address returned to the PA, so that it can issue the service request as in the previous case.

Figure 10 illustrates a typical service access case in the `SETH` system. User *Alice* enters the *hotelRoom* smart space. The personal agent arrives at the hotel room SSAP through one of the processes described in [12]. Context agents notify both the PA and $SSCA_{hotelRoom}$ of this event (1). The personal agent, knowing that its user is planning to go sightseeing for a while, and after checking Alice's user preferences, concludes that she would like to watch a presentation about the city's main monument. According to that, Alice's PA asks $SSCA_{hotelRoom}$ for an agent providing a presentation service (2). No such agent is persistently active in the SSAP, so $SSCA_{hotelRoom}$ creates an instance of a Non-persistent Presentation Service Agent (NPPSA) (3) and returns its address to the PA (4). The personal agent then contacts the NPPSA and requests it to present a slideshow of some touristic contents known to be stored in the tourist office (5). Then, the NPPSA contacts $SSCA_{touristoffice}$ to learn who to ask for the document (6). He is given the address of a persistent *File Transfer Service Agent* (FTSA) (7), which the NPPSA contacts to request the file (8). The FTSA obtains the file from the *Device Agent* (DA) associated to the content server in the tourist office (9) and transfers it to the NPPSA (10). Finally, the NPPSA requests the *device agent* of the TV in the hotel room to display the presentation (11).

Any typical interaction such as the one described above may involve service discovery requests to different SSAPs, requests to service and device agents, and even access to resources within the user's desktop computer. This flexibility of the interaction mechanism provided by agents greatly improves the functionality

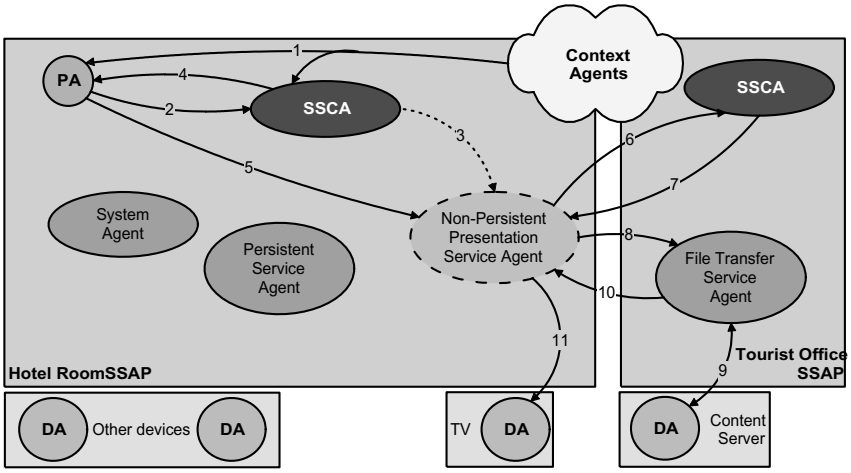

FIGURE 10. Access to services.

of the smart space, but also raises some relevant security concerns that must be addressed in order to ensure there is no misuse of the provided infrastructure [13].

5.6. Experimental results

For the evaluation and validation of the developed system, three services were implemented on top of the SETH architecture. The first was an interactive tourist guide that used GPS over a PDA to locate the user in the city map and was able to suggest touristic routes according to user preferences and user previous movements throughout the city, leading the user to the most suitable points of interest. Integrated with this service was a content service, which presented to the users personalised contents on their PDAs regarding the different points of interest in their vicinity at each moment. The contents were hosted in a centralised touristic content server. Finally, the negotiation system described in [24] was adapted and integrated with the SETH architecture to provide a recommendation service for restaurant reservations. When lunchtime approached, the personal agents residing on users' PDAs contacted the reservation agents at the different restaurant smart spaces and used the information received to make weighted recommendations to the users. The city smart spaces were simulated in our campus, using the wireless network infrastructure available to provide connectivity for the user PDAs and the servers representing the different smart spaces (points of interest, tourist office, and restaurants). The experiment yielded promising results, and we expect to be able to test the system on a real-world scenario, making a limited deployment of the SETH architecture in the touristic city of Alcalá de Henares by 2008.

6. Dynamic location tracking for Tourism applications

When considering data services for tourism, an important aspect is to provide the mobile user with information relevant to his current location (e.g., information about nearby attractions or the locations of his travel guides). In this section, the challenges that this goal presents are first analyzed, from the point of view of location tracking, through the concept of location-dependent queries. Then, `LOQOMOTION` [18], an architecture that benefits from agent technology to overcome such challenges, is presented.

6.1. Challenges for location-dependent query processing

In a tourism context, a user could specify his interests by issuing (e.g., with the help of a graphical user interface) *location-dependent queries*, which are queries whose answer depends on the location of relevant *objects* (interesting entities). As an example, a tourist guide could want to track the locations of his customers (calling them if they get too far away), or a tourist may want to track the nearby tourist buses. These queries must be handled as *continuous* queries [23], that is, their answers must be continually refreshed because they depend on the locations of the involved objects. For example, the answer to a query issued by a tourist guide to track his nearby customers will change as the tourist guide and/or customers move around. Moreover, it would be very interesting to continuously show the updated locations of those customers to the user, locating them on a map. Therefore, the answer to the query must be updated with new location data even if the set of customers satisfying the query condition does not change (as their locations do change continuously).

As the previous examples show, the most frequent types of location-dependent queries in the context of Tourism applications are those about entities that are near the current user's location. Those entities may be static (monuments, restaurants, etc.) or moving (other tourists, tourist buses, tourist guides, etc.). A user querying about the surrounding area of a different entity (e.g., a tourist agency manager asking about tourists near the tourist guides working for his company) is less frequent, although also useful in some situations.

In these contexts where users with mobile devices must be provided with context-aware information, two important challenges arise. On the one hand, the most evident difficulty is derived from the mobility aspect. As the user moves, the interesting entities (*objects*) around him can change continually. Moreover, the user may be interested in monitoring nearby entities that are themselves mobile (*moving objects*), such as the people or friends they travel with. On the other hand, in a wireless network an object can be detected only by the *base station* that provides it with coverage; in other words, the data about the objects are naturally distributed. This is also a requirement from the point of view of performance. Thus, a single computer storing all the relevant data and also processing all the requests of data would easily become a bottleneck: such a solution would not scale with the number of users, the volume of data (number of objects), or the frequency

of data changes (which would be very high for location data). Instead, several computers are needed to manage data and queries related to different geographic areas. Along with its clear advantages, a distributed architecture introduces some difficulties from the point of view of data processing; for example, a request about nearby objects could span several geographic areas and, therefore, involve several computers.

6.2. An agent-based approach for dynamic location tracking

Existing commercial products only allow queries about static objects whose attributes do not change with time (e.g., information about nearby restaurants, whose information and location do not change); in that way, they can safely assume that the interesting information is available on the mobile device itself. In a more general context, a different solution is required. In this section, the system `LOQOMOTION` (*LOcation-dependent Queries On Moving ObjecTs In mObile Networks*) [18], an agent-based architecture which benefits from the use of mobile agents to efficiently support general query processing in a distributed environment, is presented.

The underlying infrastructure of `LOQOMOTION` is a set of computers that manage data (location and other attributes) about objects within their areas. It is based on a layered hierarchy of mobile agents that move autonomously over the mobile network in order to track efficiently the relevant moving objects (they keep themselves close to the interesting data in order to carry out the processing tasks wherever they are needed), correlate partial results and, finally, present and continually update the answer to the user's query.

The agent-based query processing approach in `LOQOMOTION` follows a divide-and-conquer strategy, with agents that cooperate to process the queries in the distributed environment and keep the answers up-to-date. The main component of a user device is a graphical application that allows the user to launch queries whose answer will be shown (and automatically updated) on the screen. Six steps can be distinguished in the query processing. The following paragraphs briefly explain how those steps are executed on a distributed environment of computers which manage data about objects within different geographic areas:

1. *Obtaining the query.* In a context where the user could be driving or walking while posing queries, it is important to require minimum user interaction. The ideal user interface would consist of: 1) a voice recognition system to allow the user to orally express a query, and 2) a convenient way to return the answers to the user (depending on the type of answers required, they could be provided through a graphical user interface, a text-to-speech engine, etc.).
2. *Analysis of the query.* The application creates a *QueryMonitor* agent in charge of the query. The *QueryMonitor* will identify the class of objects and geographic area that are interesting to the user. For example, suppose that a user wants to track the tourist buses that are close to him. In this case, the class of interesting objects would be the set of tourist buses. Regarding the

query condition, as *closeness* is a relative attribute, it must be defined previously by the user. For example, he can consider that all the tourist buses within an area of one mile are close: this is called the *relevant area*. The goal of the query processing is therefore to retrieve the objects (of a certain class) within the relevant area. Notice that this area moves with the user.

3. *Translate into a database query.* The previous query cannot be executed over a traditional Database Management System because it depends on the continuously changing location of the user. Therefore, the location-dependent query is transformed into a *database query*, that is, a query where the user is not mentioned or referenced explicitly: instead, his current location is used as part of the query conditions.
4. *Deployment of a network of agents.* Processing the query in a distributed environment is challenging because the location data that the system must correlate are handled by different computers (depending on the current location of the user) and those data are constantly changing. A suitable mechanism to perform distributed monitoring is a major concern of `LOQOMOTION`. To overcome this difficulty, the system deploys a network of agents to perform the query processing in a distributed and cooperative manner:
   (a) The *QueryMonitor* agent creates a *MonitorTracker* mobile agent which travels to the computer in charge of the user's location (see Figure 11, step 1). The *MonitorTracker* represents the mobile user in the fixed network (in a similar way to the idea of *user agent* in the *Mobile Shadow* project [21]), and plays the role of *mediator* between the mobile user and the fixed network. It follows the monitor wherever it goes, according to the approach proposed in works such as [22, 20]. Moreover, it tracks the current location of the user and processes his queries[1].
   (b) The *MonitorTracker* creates one *Updater* agent on each computer whose area intersects (totally or partially) with the area of the query (see Figure 11, step 2), which are the computers relevant to the query, in order to detect objects that satisfy the query constraints. Every *Updater* keeps track of a portion of the interesting area, so any object entering such an area will be detected by one of the *Updaters*.
5. *Execution of traditional database queries and obtaining of an initial answer.* In this step, the different types of agents in our architecture work cooperatively in order to obtain an initial answer to the location-dependent query. Firstly, each *Updater* executes its database query against the computer where it resides, with the goal of retrieving data about the relevant target objects, which it communicates to its *MonitorTracker*, which correlates the results

[1]For the sake of clarity, many aspects of `LOQOMOTION` have been simplified in this explanation. In the real architecture, the *MonitorTracker* creates a network of *Tracker agents* to process the query. Such an additional step is needed in order to process queries where an entity different from the user is explicitly mentioned in the query, which may be not so important in the context of Tourism.

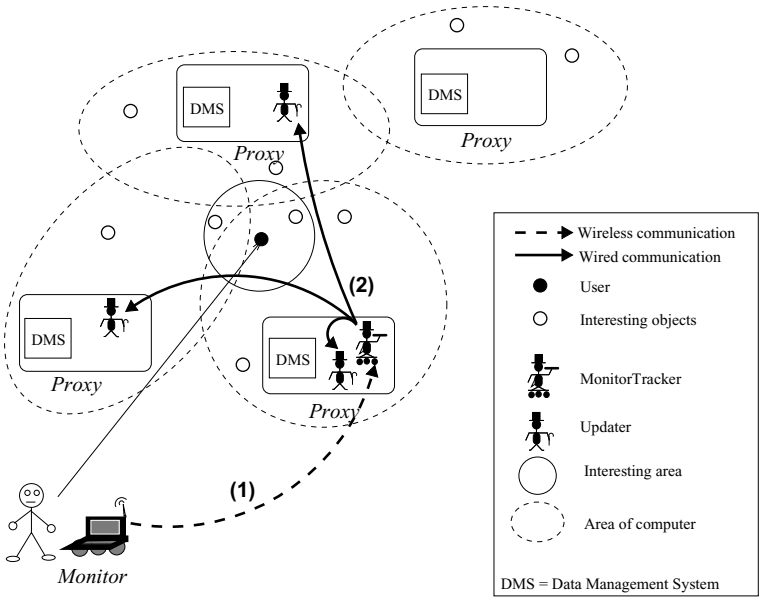

FIGURE 11. Deployment of the agent network.

it receives, and so on until the *QueryMonitor* obtains an initial answer and shows it to the user.

Notice that the only wireless data transfers needed to obtain an answer to the query occur: 1) at query initialization, when the *QueryMonitor* creates a *MonitorTracker* which must travel to the computer that manages the user's data; and 2) when the *MonitorTracker* sends an answer to the *QueryMonitor*. Any other communication occurs among fixed computers using the wired network. Thus, fixed networks are used whenever possible, following the recommendations of [19].

6. *Continuous Processing.* The previous process must be repeated periodically in order to keep the answer up-to-date. For that, two related tasks must be performed: 1) to keep the network of agents ready to obtain a new answer according to the current locations of the interesting objects (e.g., the set of relevant computers and associated *Updaters* change as the interesting area moves), and 2) to update automatically the answer to the query. During the query processing, the agents must synchronize among themselves in order to obtain the necessary data at the right time. The proposed synchronization approach [17, 16] can deal with situations where some agents cannot communicate (e.g., due to disconnections).

One of the main difficulties is that the previous steps must be executed in a distributed environment, correlating location data managed by different computers. As mentioned before, this is a basic requirement because a centralised solution to manage all the data and process all the queries is not possible at medium or large scale. Moreover, a distributed query processing approach brings important benefits; for example, objects and queries regarding different geographic areas do not compete for the same resources.

6.3. Conclusions and future trends

Processing location-dependent queries is important for Tourism applications. The use of mobile agents is very interesting in this context because they provide the right mechanisms to process location-dependent queries in a distributed and efficient manner. The different query processing tasks must *move* from computer to computer following their relevant data, which is naturally achieved by packaging those processing tasks as mobile agents. An alternative *client/server* approach could lead to a non-flexible and more difficult to implement solution. Thus, there should be a server process on each computer with the capability of launching new threads encapsulating the behaviours of the agents, and agent migrations across computers would be simulated by invoking remote procedures that create and destroy threads. Moreover, the programmer should keep track of where each relevant thread is executing (e.g., to communicate with them). This implies a significant programming effort, that can be saved using a mobile agent platform. Moreover, a mobile agent platform is also expected to perform better. Thus, the use of an architecture based on mobile agents is very interesting in this scenario.

Besides the difficulties of designing an efficient mechanism for dynamic location tracking for Tourism applications, testing the proposed solutions is also challenging. Existing alternatives are usually evaluated by means of simulations. Although they are expected to perform similarly in a real environment, some difficulties will undoubtedly arise to validate them in a wireless network (e.g., currently there is no real interoperability between different mobile agent platforms, so all the devices would need to use the same agent platform). Existing wireless network protocols could also be improved in the future to facilitate the transmission of location data from moving objects.

7. Summary and conclusions

The agent-based systems described in this chapter, albeit having mainly an academic focus, prove how the properties of agents (autonomy, proactiveness, social capability, intelligent behaviour with high level reasoning and planning) seem to fit quite nicely with the needs of Tourism service providing systems. Furthermore, the current trend towards ever more affordable and computationally powerful PDAs and mobile phones permits us to predict a mid-term future in which tourists will have *Personal Agents* running in their pockets, and these agents will be able to effortlessly communicate with agents providing touristic information on the fly.

Thus, tourists undoubtedly will demand the possibility of requesting information, making reservations or getting proactive personalised recommendations at any point of the city at any time, paving the way for the development of new styles of e-business in touristic destinations.

References

[1] Aamodt A. and Plaza E. *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches*. AI Communications **7**. No 1, March 1994.

[2] Bergmann, R., Muñoz-Ávila, H., Veloso, M. and Melis, E. *CBR Applied to Planning*. In Lenz, M. Bartsch-Sporl, B., Burkhard, H. and Wess, S. (Eds.) Case-Based Reasoning Technology: From Foundations to Applications. Lecture Notes in Computer Science 1400, Springer, 1998, 169–200.

[3] Corchado J. M. And Laza R. *Constructing Deliberative Agents with Case-based Reasoning Technology*, International Journal of Intelligent Systems. **18**, No. 12, December 2003.

[4] Corchado J.M., Pavón J., Corchado E.S, Castillo L.F. *Development of CBR-BDI Agents: A Tourist Guide Application*. ECCBR 2004. LNAI vol. 3155, Springer Verlag. (2005), 547–559.

[5] González-Bedia M. and Corchado J. M.*A planning strategy based on variational calculus for deliberative agents*. Computing and Information Systems Journal. **10**, No 1, (2002), 2–14.

[6] González-Bedia M., Corchado J. M., Corchado E. S. and Fyfe C. *Analytical Model for Constructing Deliberative Agents*, Engineering Intelligent Systems, **3**, (2002), 173–185.

[7] Martín F. J., Plaza E., Arcos J.L. *Knowledge and experience reuse through communications among competent (peer) agents*. International Journal of Software Engineering and Knowledge Engineering, **9**, No. 3, (1999), 319–341.

[8] Olivia C., Chang C. F., Enguix C.F. and Ghose A.K. *Case-Based BDI Agents: An Effective Approach for Intelligent Search on the World Wide Web*, AAAI Spring Symposium on Intelligent Agents, Stanford University, USA, 1999.

[9] Rao, A. S. and Georgeff, M. P. *B*DI Agents: From Theory to Practice. First International Conference on Multi-Agent Systems (ICMAS-95). San Francisco, USA, 1995.

[10] Wendler J. and Lenz M. *CBR for Dynamic Situation Assessment in an Agent-Oriented Setting*. Proc. AAAI-98 Workshop on CBR Integrations. Madison (USA), 1998.

[11] Wooldridge, M. and Jennings, N. R. *Agent Theories, Architectures, and Languages: a Survey*. In: Wooldridge and Jennings, editors, Intelligent Agents, Springer-Verlag, (1995), 11–22.

[12] Marsá, I., López, M.A., Velasco, J.R. and Navarro, A. *Mobile Personal Agents for Smart Spaces*, Proceedings of the IEEE International Conference on Pervasive Services (IEEE ICPS 06), Lyon, France, (2006), 299–302.

[13] De la Hoz, E., Marsá, I., López, M.A. and Alarcaos, B. *A Hierarchical, Agent-based Approach to Security in Smart Offices*, Proceedings of the International Conference on Ubiquitous Computing (ICUC 06), Alcalá de Henares, Spain, (2006), 11–19.

[14] Marsá, I. *A Hierarchical, Agent-based Architecture for Smart Spaces*, Technical Report 2006-101, Telematics Services Engineering Group, Univ. of Alcalá, (2006).

[15] FIPA. *FIPA Agent Management Specification, document SC00023K*, (2004). Available at www.fipa.org.

[16] Ilarri, S., Mena, E. and A. Illarramendi, A. *Dealing with Continuous Location-Dependent Queries: Just-in-Time Data Refreshment*. Procs. of the First IEEE Annual Conference on Pervasive Computing and Communications (PerCom), Dallas, (2003), 279–286.

[17] Ilarri, S., Mena, E. and A. Illarramendi, A. *Self-synchronization of Cooperative Agents in a Distributed Environment*. Procs. of Third International/Central and Eastern European Conference on Multi-Agent Systems (CEEMAS), Prague, (2003), 51–60.

[18] Ilarri, S., Mena, E. and A. Illarramendi, A. *Location-Dependent Queries in Mobile Contexts: Distributed Processing Using Mobile Agents*, IEEE Transactions on Mobile Computing **8**, 1029–1043, 2006.

[19] Badrinath, B.R., Acharya, A. and Imielinski, T. *Impact of mobility on distributed computations*. CM Operating Systems Review **27**, No. 2, 1993.

[20] Burg, B., Tab, R. and Schmitt, A. *Information Services Provisions in a Telecommunications Network*. United States Patent 6512922, (2003).

[21] Fischmeister, S. *Mobile Software Agents for Location-Based Systems*. Procs. of Agent Technologies, Infrastructures, Tools, and Applications for E-Services, NODe 2002 Agent-Related Workshops, Erfurt, Germany, (2002), 226–239.

[22] Liu, G.Y. and Maguire, G.Q. *Efficient Mobility Management Support for Wireless Data Services*. Procs. of 45th IEEE Vehicular Technology Conference (VTC'95), Chicago, (1995).

[23] Terry, D., Goldberg, D., Nichols, D. and Oki, B. *Continuous queries over append-only databases*. Procs. of ACM SIGMOD International Conference on Management of Data (SIGMOD'92), San Diego, (1992), 321–330.

[24] López-Carmona, M.A. and Velasco, J.R. *A Fuzzy Constraint Based Model for Automated Purchase Negotiations*, TADA/AMEC 2006, Lecture Notes in Artificial Intelligence, Vol. 4452, 234–247. Springer Verlag (2007).

[25] López, J.S., Bustos, F.A. and Julián, V. *Tourism Services Using Agent Technology: A MultiAgent Approach* INFOCOMP - Journal of Computer Science - Special Edition pp. 51–57. (2007)

[26] Balabanovic, M., Shoham, Y. *Content-based, collaborative recommendation*. Communications of the ACM **40-3**, 66–72, 1997.

[27] Jain, A.K., Dubes, R.C. *Algorithms for clustering data*. Prentice-Hall, (1988).

[28] Chen, H. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*, PhD Thesis, University of Maryland,(2004).

[29] Román, M., Hess, C.K., Cerqueira, R., Ranganathan, A., Campbell, R. and Nahrstedt, K. *Gaia: A Middleware Infrastructure to Enable Active Spaces*, IEEE Pervasive Computing, 74–83, Oct-Dec (2002).

[30] Johanson, B., Fox, A. and Winograd, T. *The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms*, IEEE Pervasive Computing, 67–74, (2002).

Antonio Moreno
Departamento de Ingeniería Informática y Matemáticas,
Universidad Rovira i Virgili,
ETSE, Campus Sescelades,
Av. Països Catalans, 26,
43007 Tarragona,
Spain
e-mail: antonio.moreno@urv.cat