RESUMEN DE TESIS DOCTORAL

Autonomous Agents in Collaborative Ubiquitous Computing Environments

Agentes Autónomos en Ambientes de Cómputo Colaborativos Ubicuos

Graduated: Marcela D. Rodríguez
Facultad de Ingeniería
Universidad Autónoma de Baja California, UABC
Blvd. Benito Juárez S/N, ex ejido Coahuila
Mexicali, B.C. México C.P. 21900
marcerod@uabc.mx
Graduated on: August 29, 2005

Advisor: Jesus Favela
Ciencias de la Computación
CICESE
Km. 107 Carretera Tijuana - Ensenada
Ensenada, B.C. México C.P. 22860
favela@cicese.mx

Abstract
The idea of ubiquitous computing (ubicomp) is an environment dominated by computing and communication devices of different scales which are seamlessly integrated to the users activities. The features of ubiquitous computing environments require developers to face important challenges in dealing with the complexities associated to the development of ubiquitous computing systems. This thesis describes a middleware to facilitate developers to manage some of the complexities associated with the development of ubiquitous computing systems by means of the use of autonomous agents, which enable ubiquitous computing technology to respond to users' particular conditions and demands. Autonomous agents were used to implement the desirable features of ubiquitous computing systems and for enhancing the interactions of the users with the environment. The contributions of this thesis focus on presenting the functional requirements of autonomous agents for implementing ubiquitous computing systems and the agent SALSA middleware, which was created with the aim of facilitating the implementation and evolution of ubicomp systems. Finally, this thesis provides evidence of the SALSA flexibility for enabling the progressive development of ubicomp systems.
Keywords: ubiquitous computing, autonomous agents, middleware

Resumen
La idea del cómputo ubicuo propone un ambiente físico dotado de dispositivos de diferentes escalas, con capacidades computacionales y de comunicaciones, los cuales se integran de forma natural a nuestras actividades diarias. Las características de los ambientes de cómputo ubicuo, conlleva a los desarrolladores a abordar varias complejidades de estos sistemas y a enfrentar importantes retos. Esta tesis describe un middleware que facilita a los desarrolladores manejar algunas de las complejidades asociadas con el desarrollo de los sistemas de cómputo ubicuo por medio del uso de agentes autónomos. Los agentes autónomos se utilizaron para implementar las características deseables de estos sistemas de cómputo ubicuo y para mejorar la interacción de los usuarios con el ambiente. Las contribuciones de esta tesis se enfocan en presentar los requerimientos funcionales de los agentes autónomos para implementar sistemas de cómputo ubicuo, y el diseño e implementación del middleware SALSA creado para facilitar la implementación y evolución de sistemas de cómputo ubicuo. Finalmente, se proporciona evidencia de la flexibilidad de SALSA para desarrollar progresivamente sistemas de cómputo ubicuo.
Palabras clave: cómputo ubicuo, agentes autónomos, middleware

1 Introduction and motivation

The term ubiquitous computing (ubicomp) was coined by Mark Weiser in 1988 to define an environment in which computers are embedded in the objects we use everyday (Weiser, 1991). The goal of ubiquitous computing is to enhance computer use by making many computers available throughout the physical environment that is effectively invisible to the user (Weiser, 1993). This environment is furnished with computational resources of all scales that provide information and services when and where desired, such as, digital tablets, wall-sized electronic whiteboards, laptops, handhelds and personal digital assistants (PDAs), some of which allow greater user mobility in pervasive environments. Undoubtedly, the development and deployment of the necessary infrastructure to support continuous mobile computation is arriving (Abowd and Mynatt, 2000). However, ubicomp promises more than just infrastructure, it suggests new paradigms of interaction and collaboration between users and/or services, inspired by widespread context-aware access to information and computational capabilities (Abowd and Mynatt, 2000), which have led developers of ubicomp systems to cope with several challenges for creating these systems. Such challenges have major implications for the software infrastructure that must facilitate the progressive development of a ubicomp system.

1.1 Ubiquitous computing challenges

Building a ubicomp system requires developers to address several challenges in order to cope with the complexities associated with the development of ubiquitous computing systems. Among these complexities are the *routine failures* that happen due to unpredictable events, for instance those related to disconnections of mobile users; *heterogeneity* in computing, communication and sensing devices embedded in the physical environment that leads to challenges related to the *adaptation* of information which must often take place without human intervention by taking into account the *user's context*, which is highly dynamic; *discovery of service/devices* should be provided by the ubicomp infrastructure by requiring minimal or no configuration from the user; *scalability* with respect to devices, people, and time since new devices and people can join the environment at any time; finally, *providing software support for building ubicomp systems* (such as toolkits, frameworks and middleware) is a major challenge identified by the ubicomp research community (Banavar and Bernstein, 2002; Davies and Gellersen; 2002, Kindberg and Fox, 2002). However, the existing development architectures provide support for dealing with some of the complexities of ubiquitous computing systems, but they do not address how to facilitate the evolution of a pervasive system. Developers have also adopted programming techniques already used for developing complex distributed software systems, such as software agents. For instance, some projects use the agents' paradigm to speed-up concurrent processing, and provide more reliability because of the lack of a single point of failure and improve the responsiveness of the system. However, they do not use autonomous agents as an abstraction tool for the design and construction of these systems (Campo, 2002; Carolis and Pizzutilo, 2002; Laukkanen et al., 2002; Villate et al., 2002). The following section explains what an autonomous agent is and why they are more appropriate for conceiving ubiquitous computing systems.

1.2 Agents as an alternative to deal with the challenges of developing ubicomp systems

A software agent is a software entity that acts on behalf of someone to carry out a particular task which has been delegated to it. To do this, an agent might be able to infer users' preferences and/or needs by taking into account the peculiarities of users and situation (Bradshaw, 1997). Each agent might possess a greater or lesser degree of attributes which have to be consistent with the requirements of a particular problem. Some of these attributes are: autonomy (to act on their own), re-activity (to respond to changes in the environment), pro-activity (to reach goals), cooperation (with other agents to efficiently and effectively solve tasks), adaptation (to learn from its experience) and mobility (to migrate to new places) (Bradshaw, 1997; Wooldridge and Jennings, 1995). The motivation for this thesis to explore the use of autonomous agents for creating ubiquitous computing systems is that ubicomp environments possess the characteristics of distribution, reactivity, collaboration and adaptation of their artifacts, thus sharing several characteristics with agents. Besides that, autonomous agents offer several advantages for building ubicomp systems:

- Autonomous agents are considered software components that offer greater flexibility and adaptability than traditional components (Griss and Pour, 2001).

- Implementing a ubicomp environment as a multi-agent system can hide the fact that devices, services and information are disseminated all over the physical environment and makes it possible to create an environment with autonomous components that provide largely invisible support for tasks performed by users.
- As agents are components with the ability to initiate and respond to interactions in a flexible manner (Jennings, 2001), the agent oriented approach can be the natural way to deal with unpredictable associations and interaction among ubicomp systems' components.
- To explore novel forms of interaction in order to achieve Mark Weiser's vision of naturally integrating computer technology with our daily activities, autonomous agents can seamlessly assist users in their interactions with the ubicomp environment.

1.3 Research Objective

Based on the thesis motivation and the identified advantages of using autonomous agents for conceiving ubicomp systems, the following thesis objective was stated.

Design and develop a middleware to allow developers to manage some of the complexities associated with the development of ubicomp systems by means of the use of autonomous agents.

1.4 Thesis outline

Before presenting the middleware, named SALSA, section 2 presents an analysis of the facilities provided by existing development platforms for ubiquitous computing; section 3 describes the research methodology followed during this thesis; section 4 presents an scenario that illustrate how ubiquitous computing technology enhances users' activities, and that were analyzed to identify the requirements for the SALSA middleware; section 5 presents these requirements and the design and implementation of SALSA; section 6 illustrates the use of SALSA for implementing a ubiquitous computing system; section 7 describes the evaluation of SALSA; and finally, section 8 presents conclusions.

2 Related work

Several of the middlewares for creating ubicomp environments, focus on facilitating the development of applications for mobile devices that need to interact with other devices in the ubicomp environment. YCab is a lightweight API that developers use to rapidly create applications that allow mobile users to collaborate in ad-hoc wireless networks (Buzko et al., 2001). This framework focuses on providing a fault-tolerant environment by allowing collaborators to float in and out of an ad-hoc network without causing disruption to the collaborative session. Since the applications that can be developed with YCab are specifically for mobile devices of ad-hoc networks, it only allows interactions between mobile users, restricting the user's opportunity of interacting spontaneously with any other device that could offer a relevant service that may enhance the collaborative activities of the user. DACIA is a development framework that provides mechanisms for building groupware applications that adapt to available resources and support user mobility (Litiu and Parkash, 2000). Similarly, DACIA copes with the disconnections of mobile users. A mobile application can be parked while its user is disconnected or idle, at this point it becomes the client agent. The support that DACIA provides for intermittent connectivity is weak in the sense that it does not allow the client agent to continue interacting on behalf of the user when the disconnection was produced due to a failure of the network. The activities that a client agent can carry out while the user is disconnected are too specific and simple, such as, save messages; inform other parties that the user is not active or forward notifications. In (Popovici et al., 2003) a platform is presented to enable mobile computing devices to dynamically extend or modify the functionality of an application. The idea is to let the environment proactively adapt the application rather than forcing the application to adapt itself to every possible environment. This is a different approach to adaptation from the way in which it is usually addressed in ubiquitous computing, in which the devices in the environment are intelligent enough to adapt themselves to a user's context in order to enhance his interaction with the ubicomp environment. CARISMA is a mobile computing middleware that exploits the principle of reflection to enhance the construction of mobile applications that have to adapt to changes in context, such as variations in network bandwidth, memory or battery power (Capra et al., 2003). The model followed by CARISMA, assumes that the behavior of the middleware with respect to a particular service is determined, at any time, by a policy. This middleware focuses on providing a protocol to enable mobile devices to negotiate for services available in the

environment by adapting the access policies to these services. Similar to the middleware previously presented, the adaptation provided by CARISMA is not based on the context or preferences of users.

Thus, the above mentioned middlewares do not take into account the users' context to enable the provision of adapted services or for enhancing the collaboration among users. To address these issues, other middlewares focus on supporting the challenges associated with the provision of context-aware services, such as, the gathering, representation and dissemination of context information. RCSM is a middleware designed to facilitate the development of applications which require context aware ad hoc communication (Yau et al., 2002). By using RCSM, the developer focuses on implementing the actions, in any language, without worrying about context monitoring, detection, and analysis. Semantic Space is a pervasive computing infrastructure that exploits Semantic Web technologies to support explicit representation, expressive querying, and flexible reasoning of context in smart spaces (Wang et al., 2004). The infrastructure of Semantic Space provides mechanisms to let applications retrieve context information using declarative queries, infer higher-level contexts by using heuristic rules, enable devices to dynamically join the environment, and monitor these devices in order to abstract context information from them. In (Kim et al., 2004) a context-awareness middleware is presented, which utilizes a standardized context scheme called Human Interaction Markup Language (HIML) to manipulate context information. It was created with the aim of providing a common language for expressing context-awareness for any kind of device, and for facilitating the user's interaction with the context aware system by modifying his current contextual information. This middleware enables the components of the ubicomp system to describe and construct context messages and store them in a database for later use. Context-Aware Middleware for Ubiquitous computing Systems (CAMUS) is a middleware that focuses on providing context composition and an efficient separation of concerns between different sensing techniques and context formation processes (Hgo et al., 2004). The data extracted from sensors, called features, are represented as a Feature Tuple Space (FTS), which is the communication and storage mechanism used by the middleware. Tuples are mapped to convert a given feature into context, which is saved in an ontology repository. The ontology contains the domain concepts and properties with formal semantics to enable the categorization of context entities into agents, devices, environment, location and time.

Finally, other software platforms provide programming support for addressing ubicomp challenges related to the operating system or middleware over which a ubicomp environment is built. These platforms provide an API and a set of services to cope with some of the complexities for implementing ubicomp systems. The ActiveSpaces project developed the Gaia meta-operating system, which is a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space (Román et al., 2002). The Gaia OS kernel provides a set of basic services to ubicomp applications, such as, managing events or changes that occur in the active space, letting applications register and query for particular context information, or managing users' data automatically. Gaia also provides an application framework, which lets users construct pervasive systems and an infrastructure to add context-awareness features to systems (Ranganathan and Campbell, 2003). One.world enables the development of adaptable pervasive applications (Grimm, 2004). Among the facilities supported by one.world is the discovery of resources. However, the one.world mechanism of service discovery does not facilitate the specialized look up, such as searching for a service with specific features, or searching for the services within a sub-area of the physical environment. One.world expresses all communication among components, through asynchronous events, which limits the quantity of data to be communicated since the focus of the event model is just to notify listener objects of changes in their runtime context.

The development platforms presented in this section provide support for addressing some of the complexities associated with the development of ubiquitous computing systems by using different programming techniques (i.e. Aspect Oriented Programming), models for representing data (i.e. ontologies) and mechanisms to handle the system's behavior (i.e. reflection). Due to the characteristics of software agents, explained in Section 1.2, this thesis explores the use of software agents as the key computer-based components of a ubiquitous computing system, and proposes a middleware that facilitates the implementation of autonomous agents which are the main components of a ubiquitous computing system. The following section presents the methodology followed to create this middleware.

3 Methodology

The research methodology consisted of several iterative phases as illustrated in Figure 1. The first phase consisted of selecting scenarios that illustrate how ubiquitous computing technology enhances users' activities. These scenarios then were analyzed to identify how autonomous agents can be used for designing ubicomp systems. In the next phase the requirements of a middleware to support the development of these ubicomp systems were identified. Based on these requirements the SALSA middleware was designed and implemented. Finally, the use of SALSA agents for designing and implementing ubicomp systems was evaluated. The aim of this evaluation phase was to assess whether the thesis objective was reached. In order to do this, the following properties were identified as being relevant and desirable for evaluating in a middleware:

• *Utility.* This criterion determines whether others can build real pervasive applications atop the infrastructure.

• *Completeness.* This criterion determines whether the architecture is sufficiently powerful and extensible to support interesting user-space programs.

• *Ease of use*. Development frameworks should be evaluated on how readable programs using the programming language are by other programmers, how learnable it is, how convenient it is for expressing certain algorithms, and how comprehensible it is to novice users.

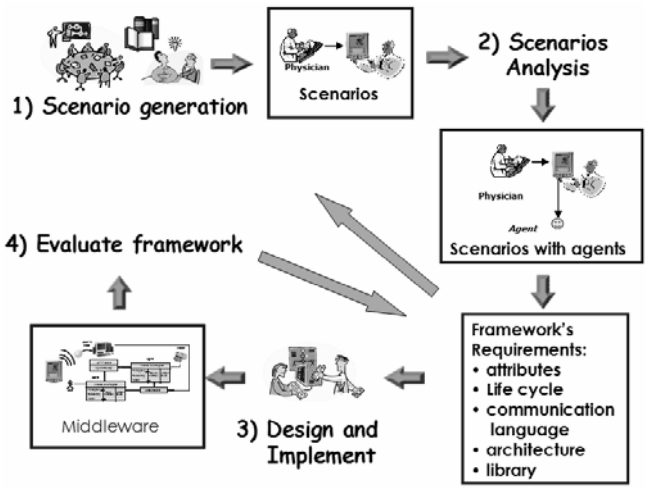

Fig. 1. Methodology followed to create and evaluate the agent middleware for ubicomp systems

4 Autonomous agents for designing ubiquitous computing systems for hospitals

For analyzing how autonomous agents can enable developers to deal with the complexities of ubiquitous computing systems, we decided to study the medical activities in Hospitals, which are settings characterized by the need for coordination and collaboration among specialists with different areas of expertise, an intense information exchange, and the mobility of hospital staff, patients, documents and equipment. The following is one of the scenarios that show how a hospital can become a ubiquitous computing environment that supports the medical activities of the hospital staff.

4.1 Scenario: Context-aware access to medical information

While Dr. Diaz is checking the status of a patient (in bed number 1 of room 222), he realizes that he needs to request an ordinary laboratory test for her. Through his PDA, he adds this request to the patient's clinical record of the Hospital's Information System. The chemist (responsible for taking the samples for the analysis) visits the internal medicine area

every morning. His PDA informs him that inside room 222 there are three patients that require a medical analysis. When the chemist stands in front of the patient, his PDA shows him the samples that have to be taken and the type of analysis to be performed. He labels the samples and at the end of his round he takes them to the lab to perform the analyses. The results are added to the patient's clinical record. When the doctor is about to finish his shift and while walking through the corridor, his PDA alerts him that the test results of the patient in bed number 1 in room 222 are available. Dr. Diaz goes back to the patient's room and when he stands near the patient's bed the results of the analysis are displayed on his PDA. At that point, the doctor revaluates the patient and based upon the results just received, decides to prepare him for surgery.

4.2 Design issues regarding autonomous agents

The selected scenarios were analyzed with the purpose of discovering how autonomous agents can enhance the activities of users. As a result of this analysis, the following design issues arose regarding the functionality of autonomous agents for creating ubicomp systems:

- *Autonomous agents are decisions makers.* The agents would review the context and make decisions about what activities to do, when to do them, and what type of information to communicate to whom.
- *Autonomous agents are reactive to the contextual elements of the environment.* Agents may need to monitor contextual elements (such as: location, delivery time, users' role, and location and state of artifacts and users) to opportunistically providing information and services to users. For this, agents need mechanisms to perceive, recognize and disseminate different types of context information.
- *Autonomous agents can represent users, act as proxies to information resources of the environment or to wrap a complex system's functionality.* In the scenarios, a doctor can interact with her personal agent that will request, on her behalf, medical information from an agent acting as proxy to the hospital information server. Finally, agents can be wrappers of complex system's functionality, such as an agent that continually estimates the user's position.
- *Autonomous agents should be able to communicate with other agents, or directly to users and services.* For this, agents need a platform of communication that enables them to convey information to other agents, users, and information resources by using the same protocol of communication.
- *Autonomous agents act as proxies to devices.* Agents representing devices, such as the public display, can be aware of the presence of other agents and users available in the environment. These agents enable allowed users and other agents to use the devices.
- *Autonomous agents need mechanisms for authentication.* Autonomous agents representing devices or services need mechanisms for authenticating users and agents that want to access them.
- *Autonomous agents need to communicate different types of messages.* Agents need a communication language that enables them to convey messages for requesting information from devices or services, (i.e. requesting medical information from the hospital information system) and responding to such requests, notifying information to users and devices (i.e. notifying that the lab results are available to the user), and requesting from another agent the execution of an action (i.e. personalizing the hospital map for the user).
- *Autonomous agents may have a reasoning algorithm as complex as the logic of its functionality.* An autonomous agent needs to be aware of information regarding the environment in order to generate its actions. For this, agents may need a reasoning algorithm which may include a simple set of rules or conditions or a more complex reasoning.

5 The SALSA Middleware

Once the use of autonomous agents for designing ubicomp systems as the previously presented was explored, the next step was to abstract the functional requirements for a middleware that facilitated the development of these autonomous agents.

5.1 SALSA's functional requirements

- To implement autonomous agents as decision makers, the middleware should provide mechanisms for implementing the agents' components for perceiving, reasoning, and acting.

- Autonomous agents in a ubiquitous computing environment can perceive different types of context information. The middleware should provide higher-level mechanisms to enable agents to perceive context information from other agents, directly from the devices or services, or from the users.
- Once an agent received information, the reasoning component of an autonomous agent has to analyze it to decide how to act. The middleware should be flexible enough to enable developers to endow agents with any reasoning algorithm, and enables developers to easily modify or update the agent's reasoning requiring little or no modifications to the other agent's components.
- Agents may need to update its reasoning algorithm at execution time. For this, it is proposed that the middleware provides facilities for enabling agents to update their reasoning algorithm by acquiring the code from a trusted server.
- The actions of autonomous agents may require them communicate with other agents by interchanging different types of messages for conveying the intention of the interaction and the information content.
- The middleware should provide a communication platform that enables agents to convey information to other agents, users, devices and services by using the same protocol and communication channel. The communication platform should enable users to be aware of the presence of other users and agents that offer relevant services for their activities, but they should be unconscious of other agents with whom they do not need to interact explicitly or that hide a complex functionality. Finally, it should enable agents to negotiate services with other agents, or request them to execute an action.
- The communication language should be flexible enough to allow programmers specify the content of each of these messages and the programming language of the middleware should facilitate the creation of these messages.
- The middleware should provide an infrastructure of services that enable the naming, registration, authentication, and location of agents representing users, devices and services.

5.2 Design of SALSA

To enable developers to create the software entities of a ubiquitous computing environment with which users need to seamlessly interact, the Simple Agent Library for Smart Ambients (SALSA) was created (Rodríguez et. al., 2005). Figure 2 presents the Architecture of the SALSA middleware which consists of the following layers:

5.2.1 Communication Platform

The communication channel among agents and users is a Broker component which is responsible for coordinating the communication, and enables the configuration of the environment in a manner that is transparent for users since they do not know the location of the agents even though they can be aware of the presence of some of these agents. Thus, an Agent Broker is the component that should handle the communication among the ubiquitous devices, services and users, which are represented by agents. The implementation of the Agent Broker is the Jabber Instant Messaging and Presence (IM&P) server (www.jabber.org). SALSA also provides a protocol of communication which consists of an expressive language that enables the exchange of different types of objects between agents (such as perceived information, requests for services), between agents and users (such as events generated by the user's actions), and between agents and services (such as the state of a service). This information will be sent or perceived by the agent through a proxy to the Broker, which is an agent's component. The Broker's Proxy and the set of messages that can be communicated among agents are created by developers by using the SALSA API as explained in the next section.

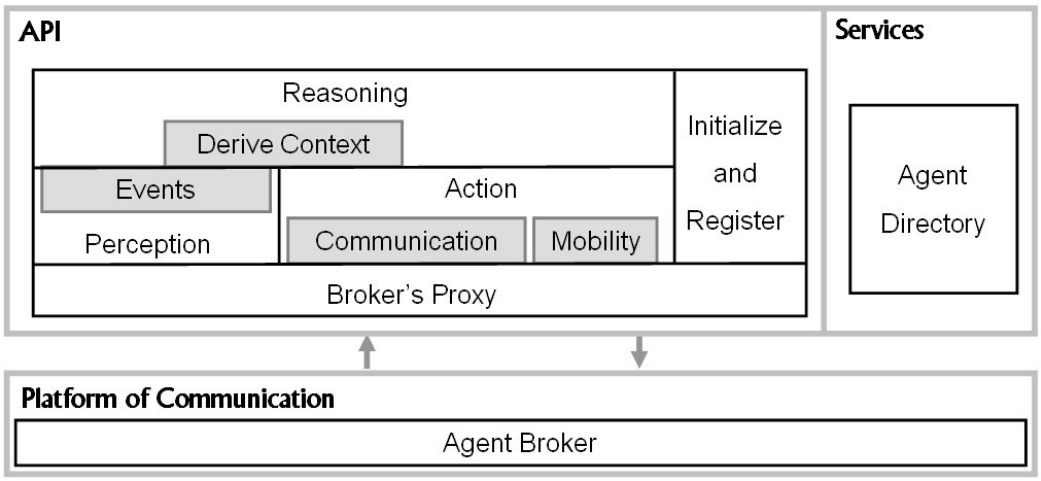

Fig. 2. SALSA's Architecture

5.2.2 API (Application Programming Interface)
This is a class framework designed for facilitating the implementation of the agents' components and the use of the services provided by the SALSA middleware. Thus, it is the SALSA API that enables developers to implement the agent's components for perceiving, reasoning and acting, and to control the agent's life cycle. The Perception component gathers context information from the environment's sensors and devices, from the users through a graphical user interface, and from other agents through the Broker's Proxy. The perceived information generates events which are captured by the Reasoning component, which governs the agent's actions. The programmer, based on the logic of the agent, implements this component by using a reasoning algorithm, such as a simple condition-action rule, a neural network or case based reasoning. The Action component contains the action plan to follow based on the agent's reasoning. It also includes sub-components that allow the agent Communication, Mobility in order to update its reasoning component, and to Derive Context information based on information perceived by the agent.

5.2.3 Services
The SALSA middleware provides an Agent Directory service which is accessible through the Initialize and Register module of the SALSA API. It provides a set of classes that allow programmers to register the agent's attributes in one or more Agent Directories, and enable agents to look for services provided by other agents.

5.3 SALSA class framework
The SALSA class framework provides a set of classes to facilitate the implementation of the internal architecture of an agent and control its life cycle. The set of classes provided by SALSA are depicted in figure 3 and explained in the following sections.

5.3.1 Agent perception
Two types of perception were identified for SALSA agents: active and passive.

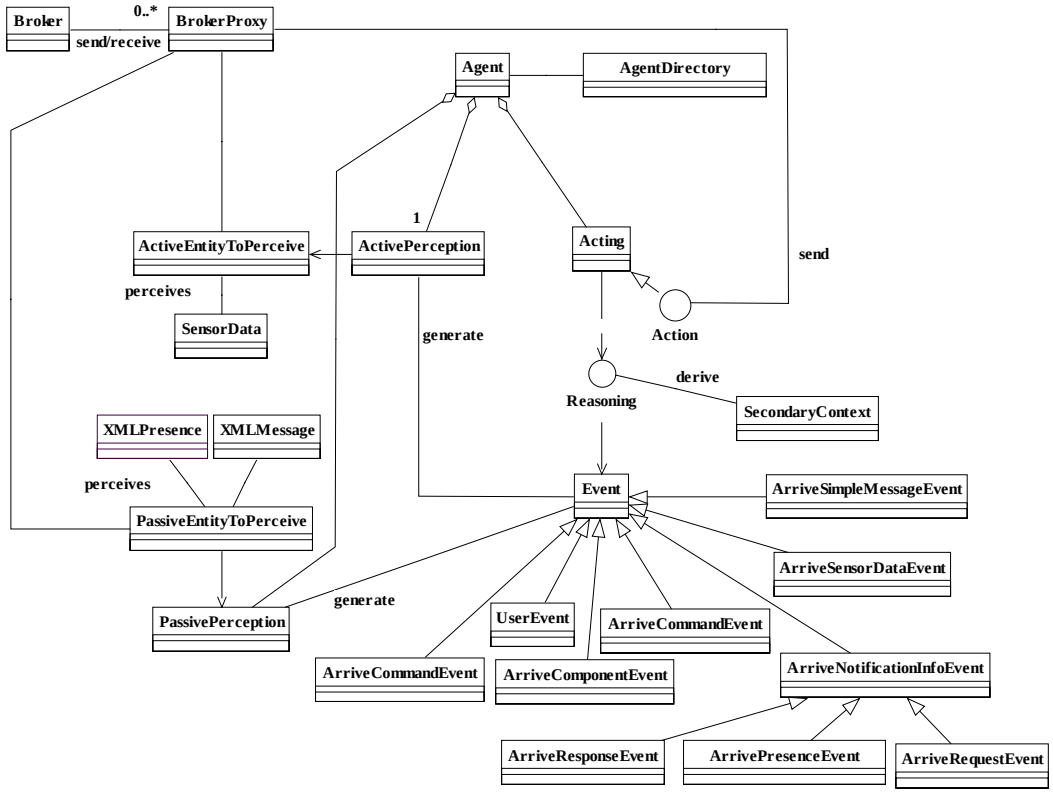

Fig. 3. Class library of SALSA

The passive perception was implemented based on the Observer design pattern (Breemen, 2003). This type of agent perception starts when a user, device or other agent sends data to an agent through the Agent Broker. In this case an agent has the role of observing the environment and acting according to the information received. The `PassiveEntityToPerceive` class represents the subject to be observed by the agent; and the `PassivePerception` class captures the information sent by the subject. For the active perception, an agent decides on its own when to sense an environment entity, and requests this information from another agent, or directly from a sensor's entity. This type of perception implements the Adapter design pattern. The classes for implementing active perception are `ActiveEntityToPerceive`, which represents the environment entity that obtains data from a sensor or device. An agent decides when to perceive information by invoking the method `passivePerception.perceive()` from the Action object. The only active perception supported by SALSA, is when the agent perceives data directly from a sensor or device. The passive perception of a SALSA agent, in which data is received through its IM client, is due to another agent that sends information by using the communication methods of the SALSA API presented in Table 1.

Table 1 XML format of three of the SALSA communication methods

```
sendCommandRequest():
     <message to='agentA@server_jabber' from='agentB@server_jabber'>
           <x xmlns='x:command'> <params><type>TypeOfCommand </type>
              // TAGS DEFINE BY THE DEVELOPER  </params>
           </x> </message>
sendResquest():
     <message to='agentA@server_jabber' from='agentB@server_jabber'>
           <x xmlns='x:request'> <params><type>TypeOfRequest </type>
              // TAGS DEFINE BY THE DEVELOPER </params>
           </x> </message>
sendNotificationInfo():
      <message to='agentA@server_jabber' from='agentB@server_jabber'>
        <x xmlns='x:notificationInfo'><params><type>TypeOfNotification </type>
              // TAGS DEFINE BY THE DEVELOPER </params>
        </x> </message>
```

When any of the perception components receive information, a SALSA event is generated indicating the type of information to the reasoning component. In addition to this, a SALSA event also contains the perceived data, which can be an XML message received through the Agent Broker, or an object containing the data that was read directly from a sensor's interface.

5.3.2 Agent reasoning

The information perceived by an agent is subtracted from the event by the reasoning component in order to be analyzed. The programmer, based on the logic of the agent, implements this component by using an appropriate reasoning algorithm, such as production rules, a neural network or case base reasoning. The Reasoning class contains the abstract method `think()` that should be implemented by the developer. The reasoning component can use the facilities of SALSA to derive context information from the primary context information perceived by an agent. For this, SALSA provides the class `DeriveContext` which uses an XSL file as a filter in which the developer specify a set of rules to deduce secondary context from the data perceived by the agent. The derive context component returns an XML message to the agent's reasoning in order to indicate the inferred situation.

5.3.3 Agent action

To implement the action component, the framework provides the Action class with an abstract method that a developer should overwrite to specify how the agent must react. From, the action component, the agent can invoke the methods of communication provided by SALSA in order to collaborate with other agents. These methods are presented in Table 1. The acting component also enables the agent mobility. It was implemented based on the pattern Rc2s (request a component to a server) which enables an agent to update its reasoning component by getting a copy of the reasoning algorithm from other agent residing on a server (Koukoumpetsos and Antonopoulos, 2002).

5.3.4 Agent communication protocol

The SALSA development framework provides a friendly agent language taking advantage of XML to encode any kind of message. SALSA provides developers with an API that facilitates the composing, sending, and receiving of messages between agents. However, the code for every content message type of the communicative act is left to the programmer, because it depends on the intent of the message generated by each agent in the ubiquitous environment. The API of SALSA for implementing the communication among agents consists of several methods that form the message that wants to be communicated as illustrated in Table 1. For instance, the `sendCommandRequest(map@serverJabber,"personalize",marcerod)` method is used by an agent to request another agent to execute a specific action or service. When it is invoked, it will form an XML message (as the illustrated in Table 1) with tags that specify to whom the message is addressed `(mapa@serverJabber)`, the type of service requested `(personalize)`, and the parameters needed to perform the action `(userID)`.

6 Creation and evolution of ubicomp systems with SALSA

To illustrate the flexibility of SALSA for implementing a ubiquitous computing system, this section explains how an autonomous agent can be implemented with SALSA, and then, how this agent can be easily integrated to a ubiquitous computing system. For this, we revisit the scenario presented in section 4.1 which illustrates the functionality of the Context-aware Hospital Information System (CHIS) (Muñoz et al., 2003).

6.1 Implementing an autonomous agent for estimating the mobile user's location

To estimate the location of the mobile users of the CHIS system, we used an autonomous agent that wraps in its reasoning component a neural network trained to map RF signals from a WLAN to 2D coordinates. Once trained, the neural network was used to classify incoming patterns into labeled classes, X,Y coordinates. More details of the neural network implementation are explained in (Rodriguez et al., 2004).

Figure 4a) shows a class diagram based on the AUML (Agent Unified Modeling Language) notation that illustrates the design of the location-estimation agent (LE-a). At the top of the diagram is specified the information that define the state of the agent: the perceived information (SNR- Signal to Noise Ratio) and the output information (coordinates X,Y). After that, the diagram defines the actions of the agent; i.e. notifying the estimated user's position to the location-aware client, for which the LE-a uses the SALSA communication protocol (`sendData()`) as indicated at the diagram bottom. Figure 4b) shows how the agent's components for perceiving, reasoning and acting, interact to estimate the user's location. These components are instances of SALSA classes.

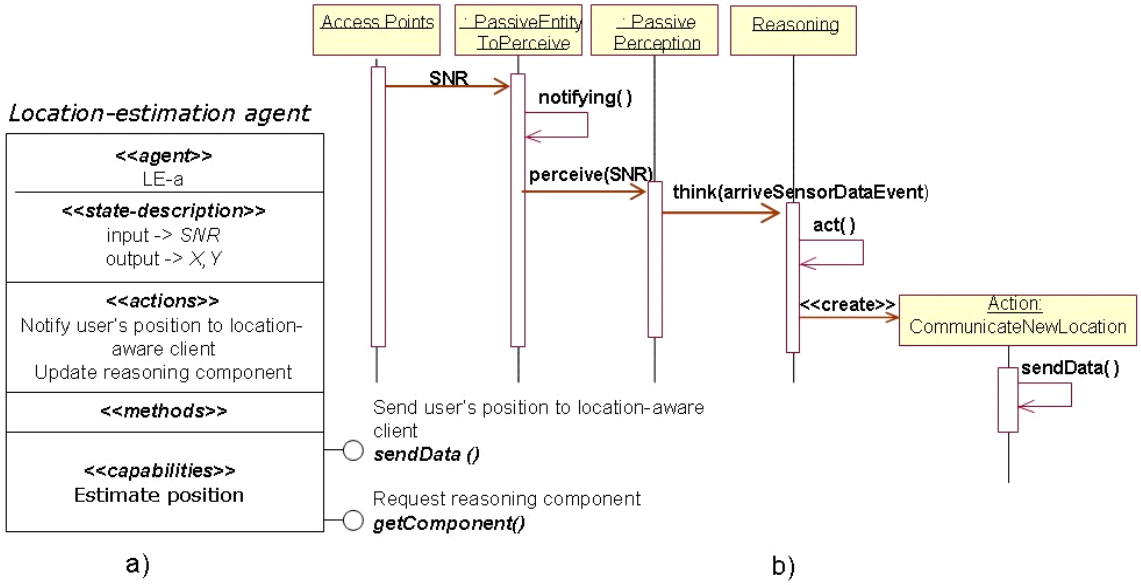

Fig. 4. AUML class diagram (a), and sequence diagram of the Location-estimation agent (b)

For implementing the perception component developers need to implement just the interface that reads the data from a device/sensor and to use the SALSA classes to connect the interface with the perception component, which is automatically activated when an instance of the Agent class is created. Figure 5a) shows the pseudo code for implementing the perception component of the LE-a using the SALSA classes. The entity for perceiving information is the `WirelessCardInterface` that contains the code for reading the SNR from the wireless LAN card. When a new SNR is read, it is passed to the passive perception component by invoking the method `pp.notifying()`.

As illustrated in Figure 5b), the `ReasoningLE` class is specialized from the Reasoning abstract class of SALSA. Its think method was overwritten to process the perceived input and then, to indicate to the agent what action should be executed. If the received SALSA event was of type `arriveSensorDataEvent`, it indicates that a new estimation of the user's location has to be calculated. Then, the `estimatesLocation()` method is invoked, which implements the trained Neural Network. When a new user's location is estimated, the reasoning component decides to communicate it to the location-aware client which is also executing in the handheld computer. To do this, the `execute()` method of the abstract class Action was overwritten to invoke the SALSA method `sendData()`.

```
import SALSA.*;                                          1
public class WirelessCardInterface{

    PassiveEntityToPerceive pp;

    public WirelessCardInterface(Agent LE_a){
       pp=new PassiveEntityToPerceive();
       pp.attach(LE_a.passivePerception);
    }

     protected void read_SNR() {
      // Code to get the SNR
         . . .
      //Creates an instance of the LE-a                  2
      pp.notifying(new Input(snr));
    }
}
```

a)

```
import SALSA.*;                                          3
class ReasoningLE_a extends SALSA.Reasoning{

  public void think(EventObject ev){
    SALSA.Events.Event event = (SALSA.Events.Event) ev;
        //if the SNR changed
    if (event.getType() == event.arriveSensorDataEvent) {
      coordinates = estimatesLocation(ev.input);

        //Invokes the action component
      agent.acting.act(new CommunicateNewLocation());
    }

        //if the new neural network for that floor was obtained
    else if (event.getType() == event.ArriveComponentEvent){
      agent.acting.act(new IntegrateComponent());
    }
  . . . .
```

b)

Fig. 5 a) Code for implementing the entity for perceiving information from the WLAN card**.** b) Code of the reasoning component

Finally, once the Perception, Reasoning, and Action components of the agent were implemented, the main body of the agent had to be created by extending the SALSA Agent class. Thus, when an instance of this agent is created, its components are activated and the life cycle of the agent begins.

6.2 Context-aware Hospital Information System (CHIS)

With SALSA we built the context-aware hospital information system (CHIS) with the aim to support the activities of the hospital staff (Favela et al., 2004; Muñoz et al., 2003). CHIS is a handheld-based system that enables users to locate relevant documents, such as patient's records and laboratory results; locate patients and colleagues; and locate and track the availability of devices such as medical equipment, and other computational resources such as public displays.

6.2.1 Architecture of CHIS

Figure 6 presents the design of the system architecture illustrating the main nodes in which the system's components are executing. These components are SALSA agents communicating through the Agent Broker (an IM server).

In the handheld computer resides the Location-aware client which notifies the user's location to other users and agents, provides mobile users with information relevant to their location, and communicates with other members of the hospital staff. In the handheld also resides the location-estimation agent (LE-a) with the purpose of obtaining the user's position (X,Y coordinates), and informs it to the location aware client. By using the mobility attribute of agents, the LE-a on the PDA can update its reasoning component by getting the reasoning component from the server in which resides the LE-a that holds a trained neural network for a specific building's floor. The Agent Directory provides information of the agents available in the environment. The hospital information system agent (HIS-a) acts as proxy of the HIS that manages and stores the patient's clinical records and other data relevant to the hospital. Finally, the Context-aware agent (Ca-a) is the system's component that sends the messages that depend on contextual variables for their delivery, such as the recipients, location and role.

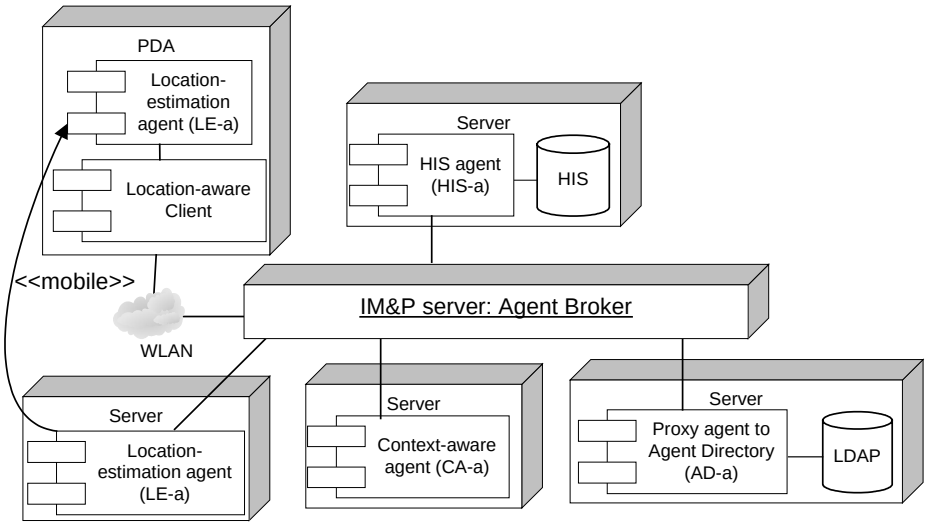

Fig. 6 Architecture of the context-aware hospital information system

6.2.2 Integrating the location-estimation agent to the context-aware hospital information system

To integrate the LE-a to the context-aware hospital information system, it was modified the location-aware client. As indicated in the pseudo code of Figure 5b), in order for the location-aware client to perceive the user's X,Y coordinates from the LE-a, a `PassiveEntityToPerceive` object was created in the location-estimation through which the passive perception of the Location-aware agent was attached to the LE-a. The reasoning and action components were not modified during the integration process of the LE-a to the context-aware system for a hospital environment. Figure 7 illustrates the interactions among the components of CHIS which includes the location-estimation agent. When Dr. Diaz visits his patient, then the SNR to the access points change. This is perceived by the LE-a (`perceive(SNR)`), which estimates the user's location based on its trained neural network (think(estimate user's position)). Thus, the agent obtains the user's position (X,Y coordinates) which is communicated to the location aware client (`sendDataSensor()`). The reasoning component of the location aware client translates the X,Y coordinates to an id of the place in which is located the user, i.e. a bed number (`location: think()`). And finally, its acting component communicates the user's location to the rest of the system's agents and users (`sendPresence(state,location)`).

7 Evaluation

This section presents the results of evaluating the utility, completeness and ease of use of SALSA through different evaluation methods which included: i) evaluating scenarios of use with real users, ii) in-lab programming experiments, and iii) design exercises for evaluating the facilities provided by SALSA agents.

The criterion for determining the utility of the middleware stated that if others can build real pervasive applications atop the infrastructure, then the architecture is useful. The first experiment determined that the selected scenarios of use of ubicomp systems, which were implemented with SALSA, were perceived as being useful by real users (such as physicians and nurses) therefore SALSA can be considered useful for creating ubicomp systems in which autonomous agents enable users to seamlessly interact with other users, services and devices. To evaluate the completeness of SALSA, an evaluation experiment was conducted to determine if the architecture was sufficiently powerful and extensible to support interesting user-space programs. For this, a hypothesis arose stating that SALSA was flexible enough to enable the evolution and iterative implementation of ubicomp applications. For demonstrating this hypothesis, section 6 presented how SALSA facilitated the implementation and extension of the context-aware hospital information system which was proposed for supporting medical activities carried out in a hospital (Muñoz et al., 2004; Favela et al.; 2004; Rodríguez 2004). The results of the programming exercise provided evidence that the execution model of SALSA

and the facilities to implement it are comprehensible. However, for some of the participants the use of autonomous agents as an abstraction for the deployment of an ubicomp system was not innate since participants had to understand various concepts related with agents and the facilities provided by SALSA, such as the agents' communication protocol, and the use of XSL for deriving context information. The design exercise, in which the students participated in, approximately two weeks after the programming exercise, the student were able to identify the agents' components as defined by the execution model of SALSA. Autonomous agents were used to represent the system's components that need to proactively act to enable the seamlessly user's interaction with the computing devices and services of the environment. The results of this design exercise provided evidence that SALSA is easy to learn and that enables developers to easily conceive a ubicomp application.

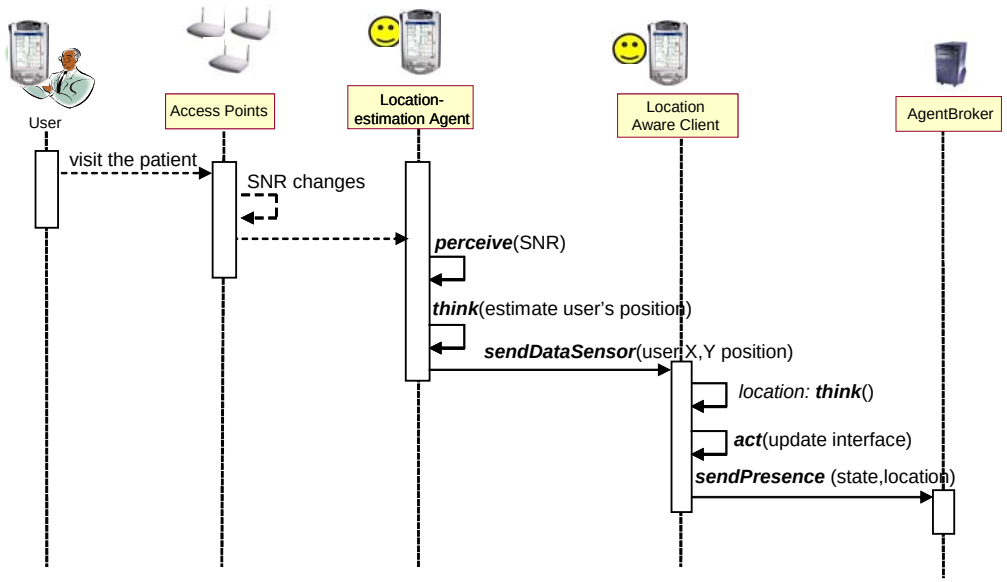

Fig. 7 Location-estimation agent interacting with the agents of the context-aware hospital information system

8 Conclusions

This thesis has presented how a ubiquitous computing system can be implemented by using autonomous agents as its main components. Autonomous agents have characteristics that can be used to implement some of the desirable features of ubicomp systems. In those systems, autonomous agents were mainly used to enhance the user's interaction with the ubicomp environment in which autonomous agents were the software components that represented users, devices, and services. This was possible since autonomous agents are reactive to the environment (to the users' context) and proactive to decide how to act (providing services and information to users). This thesis takes advantage of these agents' characteristics in order to provide an agent middleware (SALSA) that facilitates the development of ubicomp systems. The contributions of this thesis was in providing evidence of how autonomous agents can be used as an abstraction tool for designing and implementing ubiquitous computing systems, identifying the design issues regarding autonomous agents for developing ubiquitous computing systems, and providing a middleware that facilitates their implementation.

References

1. **Abowd, G.D.** and **Mynatt, E.D.,** "Charting Past, Present, and Future Research in Ubiquitous Computing", in *ACM Transactions on Computer-Human Interaction*, Vol. 7, No. 1, 2000, pp. 29–58
2. **Banavar, G.** and **Bernstein, A.,** "Software Infrastructure and Design Challenges", *in Communications of the ACM*, Vol. 45, No. 12, 2002, pp. 92-96.
3. **Bradshaw, J.,** *Software Agents*, AAAI Press/MIT Press, 1997.
4. **Breemen, A.J.N.v.**, "Integrating Agents in Software Applications", in *Proceedings of the Agent Technology Workshops LNAI 2692, Springer-Verlag*, 2003, pp. 343-354.
5. **Buzko, D., Lee, W.,** and **Helal, A.,** "Decentralized Ad-Hoc Groupware API and Framework for Mobile Collaboration", in *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work, Boulder, Colorado*. 2001, pp. 5-14.
6. **Campo, C.**, "Service Discovery in Pervasive Multi-Agent Systems", presented at the *First International Joint Conference on Autonomous Agents and Multiagents Systems, (AAMAS)* Bologna Italy, 2002
7. **Capra, L., Emmerich, W.,** and **Mascolo, C.**, "CARISMA: Context-Aware Reflective Middleware System for Mobile Applications", in *IEEE Transactions on Software Engineering*, Vol. 29, No. 10, 2003, pp. 929-945.
8. **Carolis, B.D.** and **Pizzutilo, S.,** "A MultiAgent Infrastructure supporting Personalized Interaction with Smart Environments", in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagents Systems (AAMAS)*, Bologna, Italy, 2002.
9. **Davies, N.** and **Gellersen, H. W.,** "Beyond Prototypes: Challenges in Deploying Ubiquitous Computing Systems", *IEEE Pervasive Computing*, Vol. 1, No. 1, 2002, pp. 26-35.
10. **Favela, J., Rodríguez, M., Preciado, A.,** and **González, V.M**., "Integrating Context-aware Public Displays into a Mobile Hospital Information System", *in IEEE Transactions on Information Technology in Biomedicine*, Vol. 8, No. 3, 2004, pp. 279-286.
11. **Grimm, R.,** "One.world: Experiences with a Pervasive Computing Architecture", in *IEEE Pervasive Computing*, Vol. 3, No. 3, 2004, pp. 22-30.
12. **Griss, M.L.** and **Pour, G.,** "Accelerating Development with Agent Components", in *IEEE Computer*, 2001, pp. 37-43.
13. **Hgo, H.Q., Shehzad, A., Liaquat, S., Riaz, M.,** and **Lee, S.,** "Developing Context-aware Ubiquitous Computing Systems with a Unified Middleware Framework", in *Proceedings of International Conference on Embedded and Ubiquitous Computing (EUC), LNCS 3207, Springer-Verlag*, Aizu, Japan. August 25-27. 2004, pp. 672-681.
14. **Jennings, N.R.,** "An Agent-based Approach for Building Complex Software Systems", in *Communications of the ACM*, Vol. 44, No. 4, 2001, pp. 35-41.
15. **Kim, G., Shin, D.,** and **Shin, D.,**. "Design of a Middleware and HIML (Human Interaction Markup Language) for Context Aware Services in a Ubiquitous Computing Environment", in *Proceedings of International Conference on Embedded and Ubiquitous Computing (EUC), LNCS 3207, Springer-Verlag*, Aizu, Japan. August 25-27, 2004, pp. 682-691.
16. **Kindberg, T.** and **Fox, A.,** "System Software for Ubiquitous Computing", in *IEEE Pervasive Computing*, Vol. 1, No. 1, 2002, pp. 70-81.
17. **Koukoumpetsos, K.** and **Antonopoulos, N.,** "Mobility Patterns: An Alternative Approach to Mobility Management", in *Proceedings of the 6th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI)*, Orlando , Florida, USA. 14-18 July, 2002
18. **Laukkanen, M., Helin, H.,** and **Laamanen, H.,** "Tourists on the Move", in *Proceedings of International Workshop Series on Cooperative Information Agents (CIA), Springer-Verlag*, Madrid, Spain. September 18-20, 2002, pp. 36-50.
19. **Litiu, R.** and **Parkash, A.,** "Developing Adaptive Groupware Applications Using a Mobile Computing Framework", in *Proceedings of Computer Supported Cooperative Work (CSCW), ACM Press*, Philadelphia, Pennsylvania, USA, December 2-6, 2000, pp. 107-116.
20. **Muñoz, M.A., Rodriguez, M., Favela, J., Martinez-García, A.I.,** and **Gonzalez, V.M.,** "Context-aware mobile communication in hospitals", in *IEEE Computer*, Vol. 36, No. 9, 2003, pp. 38-46.

21. **Popovici, A., Frei, A.,** and **Alonso, G.,** "A Proactive Middleware Platform for Mobile Computing", in *Proceedings of International Middleware Conference, LNCS 2672, Springer,* Rio de Janeiro, Brazil. June 16-20. 2003, pp. 455-473.
22. **Ranganathan, A.** and **Campbell, R.H.,** "An infrastructure for context-awareness based on first order logic", in *Personal and Ubiquitous Computing,* Vol. 7, 2003, pp. 353-364.
23. **Rodriguez, M.D., Favela, J., Martínez, E.A.,** and **Muñoz, M.A.,** "Location-aware Access to Hospital Information and Services", *IEEE Transactions on Information Technology in Biomedicine,* Vol. 8, No. 4. 2004, pp. 448- 455.
24. **Rodríguez, M. D., Favela, J., Preciado, A., Vizcaino, A.,** "Agent-based ambient intelligence for healthcare", *AI Communications, IOS Press,,* Vol. 18, No. 3, 2005, pp. 201-216.
25. **Román, M., Hess, C., Cerqueira, R., Ranganatha, A., Campbell, R.H.,** and **Nahrstedt, K.,** *"A Middleware Infrastructure for Active Spaces",* in *IEEE Pervasive Computing,* Vol. 1, No.4, 2002, pp. 74-83.
26. **Villate, Y., Illarramendi, A.,** and **Pitoura, E.,** "Keep your data safe and available while roaming", in Proceedings of Mobile Networks and Applications (MONET), ACM Press. 2002, pp. 315-328.
27. **Wang, X., Song-Dong, J., Yau-Chin, C.,** and **Ravipriya-Hettiarachchi, S.,** "Semantic Space: An Infraestructure for Smart Spaces", in *IEEE Pervasive Computing,* 3(3): 2004, 32-39 p.
28. **Weiser, M..,**"The Computer for the Twenty-First Century", in *Scientific American,* Vol. 265, No. 3, 1991**,** pp. 94-104.
29. **Weiser, M.,** "Some Computer Science Issues in Ubiquitous Computing", in *Communications of the ACM, Special issue on computer augmented environments: back to the real world,* Vol. 36, No. 7, 1993, pp. 75-84.
30. **Wooldridge, M.** and **Jennings, N.,** "Intelligent Agents: Theory and Practice. Knowledge Engineering" *Review, Cambridge University Press,* Vol. 10, No. 2, 1995, pp. 115-152.
31. **Yau, S., Karim, F., Wang, Y., Wang, B.,** and **Gupta, K.S.,** "Reconfigurable Context-Sensitive Middleware for Pervasive Computing", in *IEEE Pervasive Computing,* Vol. 1, No.4, 2002, pp. 33-40.

Marcela D. Rodríguez *is a professor in computer engineering at the Autonomous University of Baja California. Her research interests include ubiquitous computing, autonomous agents, human-computer interaction, and computer-supported collaborative work. She received a PhD in computer science from CICESE. She is a member of the ACM and Sociedad Mexicana de Ciencia de la Computación.*

Jesus Favela *is a professor of computer science at CICESE, where he leads the Collaborative Systems Laboratory and heads the Department of Computer Science. His research interests include computer-supported collaborative work, ubiquitous computing, and medical informatics. He received a PhD in computer-aided engineering from MIT. He is a member of the ACM and the America Medical Informatics Association, and a former president of Sociedad Mexicana de Ciencia de la Computación.*