Assessing the SALSA Architecture for Developing Agent-Based UbiHealth Applications

Marcela D. Rodríguez¹ and Jesús Favela²

¹ Facultad de Ingeniería, UABC, Mexicali, México
² Ciencias de la Computación, CICESE, Ensenada, México
marcerod@uabc.mx, favela@cicese.mx

Abstract. We have proposed the use of autonomous agents for coping with some of the challenges for creating ambient computing systems. The motivation of this research is that ambient computing environments are characterized by the distribution, reactivity, collaboration and adaptation of their artifacts, which are also characteristics attributed to software agents. To assist developers in creating the software entities of an ambient computing environment, the Simple Agent Library for Smart Ambients (SALSA) was created. The SALSA middleware and architecture enables the creation of autonomous agents reactive to the context of the ambient computing environment. SALSA agents can represent users, resources, or wrap complex system functionality of the environment. The aim of this paper is to provide evidence that SALSA facilitates the implementation of ambient computing services through autonomous agents. To reach this end, we present the results of empirical evaluation conducted to assess the use of SALSA. This study included in-lab programming experiments and design exercises to evaluate the programming facilities provided by SALSA agents. Even though for some of the evaluation participants the use of autonomous agents as an abstraction for the development of ubiquitous computing systems was not innate, the evaluation results demonstrate that the execution model of SALSA and its facilities to implement ubicomp systems are comprehensible.

Keywords: ubiquitous computing, autonomous agents, middleware.

1 Introduction

Ubiquitous computing or ambient computing suggests a new paradigm of interaction and collaboration between users and/or services, inspired by widespread context-aware access to information and computational capabilities, which have led developers of ambient computing systems to face several challenges associated with the development of these systems. Such challenges have major implications for the software infrastructure that must facilitate the progressive development of an ambient computing system. Among these complexities are those due the heterogeneity in computing, communication and sensing devices embedded in the physical environment;
adaptation of information by taking into account the user’s context; 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 ambient computing systems (such as architectures, toolkits, frameworks and middleware) is a major challenge identified by the ubiquitous computing research community [1][2]. 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 their evolution. For instance, the Gaia meta-operating system, developed as part of the ActiveSpaces project, is a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space [3]; and One.world enables the development of adaptable pervasive applications and the discovery of resources [4].

Developers have also used software agents as a technique for developing complex distributed software systems. For instance, some projects use the agents’ paradigm to speed-up concurrent processing, which are more reliable since they lack a single point of failure, and improve the responsiveness of the system [5][6][7]. However, these systems do not use autonomous agents as an abstraction tool for their design and construction.

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 take into account the peculiarities of users and situation. Each agent might possess attributes, such as: autonomy (to act on their own), reactivity (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). The motivation of our research work is to explore the use of agent-based architectures for creating ambient computing systems characterized by distribution, reactivity, collaboration and adaptation of their artifacts. Thus, Simple Agent Library for Smart Ambients (SALSA) was developed to facilitate the creation of the software entities of an ambient computing environment with which users need to interact seamlessly [8]. The methodology for exploring the use of autonomous agents for ambient computing, and for creating SALSA, included several iterative phases. The first phase involved selecting scenarios that illustrate how ubiquitous computing technology enhances users’ activities [9][10]. These scenarios then were analyzed to identify how autonomous agents can be used for designing and implementing ambient computing systems. From these analyses we identified design issues of autonomous agents and the requirements of the SALSA middleware [11]. The aim of this paper is to provide evidence that SALSA facilitates the implementation of ambient computing services through autonomous agents. To reach this end, we present the results of the evaluation conducted with the framework to assess its suitability as a development
platform. Before presenting the results of the SALSA evaluation, in section 2 we present some ubiquitous computing systems for healthcare and also architectures for developing ubicomp environments. Section 3 presents our motivation to select the domain healthcare for identifying the requirements of the SALSA middleware. Section 4 presents the SALSA agents’ design issues and requirements identified from ambient computing scenarios for healthcare. Section 5 explains the SALSA middleware. Section 6 and 7 presents the results of the empirical evaluation of SALSA. And finally, Section 8 presents our conclusions and future work.

2. Related Work

To define the SALSA requirements that aim to facilitate the development of ubiquitous and context-aware computing systems that enhance users’ activities, we focused on the domain of healthcare. Thus, before presenting how SALSA enables implementing UbiHealth systems and for situating the contribution of our work, we discuss previous research on the use of software agents for developing not only pervasive healthcare applications, but for creating architectures that facilitate the development of ubiquitous computing systems. We end this section by presenting our conclusions of the limited work that has been published related to the empirical evaluations of some of the presented architectures.

2.1 Agent-based healthcare applications

Software agents are appropriate constructs for implementing ambient intelligence applications, in which computing is pervasive, given that they are autonomous, reactive, and flexible [12]. Agent technology has indeed been used to create pervasive systems that aim to address specific needs of healthcare environments. For instance, an agent-mediated Organ Transplant Management Application (OTMA) was designed to improve the management of clinical information within a hospital [13]. The system used autonomous agents to enable physicians to trace the origin of medical decisions and audit if proper procedures were followed during an organ transplant process.

Other research projects have focused on providing support for the management of clinical information that may be distributed among different hospitals and departments within a hospital. Agents along with grid technology have also been proposed to merge organizational knowledge from various hospitals [14]. While agents deal with the management of information, such as retrieving and presenting information, grid technology copes with issues of security, such as authentication and authorization
services. Similarly, other work proposes to use mobile-agent technology to provide secure mechanisms for the integration of medical data which is distributed among several institutions [15].

Agents have also been used to provide better healthcare services, such as monitoring the state of a patient and providing assistance services to remote patients. GrusMA is a project to develop agent applications that enable users to access medical services remotely, such as accessing their medical records at any time, making online booking for appointments with a specialist, or providing support for querying about the doctors available in a certain medical center [16].

These projects support the management of patient information through software agents that have well defined objectives and that must collaborate among themselves to support different activities related to the management of medical information. Thus, through a multi-agent system these information management tasks are transparent to the users. To do this, these systems provide agents at different system’s layers. For instance, they provide agents at the user interface layer, that interpret the user’s request or that present the information to them. Then, agents at a higher layer are in charge of extracting this information and passing it to the interface agents in order to be presented to the user. The use of agents enabled these projects to cope with the distribution of clinical information in an heterogeneous environment. In spite of the fact that some of the above projects introduce the concept of ubiquitous computing for healthcare, these systems were not created with specialized architectures for addressing some of the complexities for implementing ubiquitous computing environments. However, these systems are a step towards creating a ubiquitous healthcare computing infrastructure, since they can provide abstract functional requirements that should be incorporated in ubiquitous and context aware architectures.

### 2.2 Agent-based architectures for ubiquitous computing systems

Several architectures have been proposed to facilitate the creation of ubiquitous computing systems. The SOCAM architecture is a distributed middleware which consists of independent components that provide support for acquiring, representing, interpreting and inferring context information [17]. Even though SOCAM was designed to facilitate the rapid prototyping of context-aware applications, it was only evaluated in terms of the performance of its components. Similarly to SOCAM, other middleware have placed special emphasis on providing mechanisms for facilitating the management of contextual information [18]. For instance, the OCP (Open Context Platform) middleware provides a representational model for enabling the merging of context information captured from different sources [19]. The LAICA project...
follows an approach similar to the one we propose in which abstractions for Ambient Intelligence are represented by autonomous agents [20]. Specifically, the LAICA project aims to provide a distributed middleware that addresses the performance of AmI systems, including issues of synchronization and coordination among agents by providing a look up table of the available agents in the environment and an intelligent component that decides when agents should provide their services. Thus, this middleware does not provide an API for creating AmI systems, but a set of components that enables the coordination of AmI agents.

To our knowledge, these proposals haven not been evaluated in terms of their effectiveness at facilitating the development of applications. Additionally, not much research has been published on evaluating this aspect in architectures specifically proposed for facilitating the development of ubiquitous computing systems. Software Engineering evaluation methods have been adapted to evaluate architectures’ performance and reliability when producing an application, as was done in the evaluation made to SOCAM. Research efforts have been made for proposing a model to enable developers to estimate, during the analysis and design stages, the tentative cost of implementing agent-based systems [21]. This cost can be measured by considering variables such as the source lines of code and unadjusted function points of previous developments which are similar in complexity to the one that will be implemented. However, these metrics do not provide evidence of the programmer experience when using an infrastructure for system implementation. Empirical studies suggest that for evaluating development architectures, several aspects should be assessed such as utility, learn ability, comprehensibility and modifiability [22] [4].

3 Ambient Computing to Enhance Users Activities

Healthcare environments, such as hospitals, are characterized by the need for coordination and collaboration among specialists with different areas of expertise, the integration of data from many devices or artifacts and the mobility of hospital staff, patients, documents, and equipment. Hospital staff demands to promptly extract useful pieces of data from different artifacts to perform their work. In addition, hospital personnel need to opportunistically access information and services relevant for their activities. That is, the right information has to be in the right place, whenever it is needed by whoever needs it, in whatever representation they need it. With ubiquitous computing technology it is feasible to meet these medical environment needs. Ubiquitous computing technology deployed in heterogeneous devices such as handheld computers and large public displays, can provide timely access to information and services thru a context-aware system. The next sections present the
desirable features of ubiquitous computing systems in order to support the work of hospital staff.

3.1 Context-aware Communication and Activity Coordination

We identified four critical contextual elements that an ambient computing system has to consider in supporting hospital’s activity coordination and information management [9]:

1. **Location.** Hospitals are characterized by the mobility of the professionals that work there and that of the artifacts they use, such as clinical records or medical equipment, and even that of the patients, who are moved from one hospital area to another as required. When a doctor examines a patient, he needs to move to obtain the patient’s clinical records and other documents. Hospital workers also need to move to locate information displayed in whiteboards. For instance, the schedule of patient’s operations for the current day is displayed on the whiteboard in the office of the chief surgeon. Boards help to communicate information regarding patients’ condition and location, and hospital staff often refers to the boards to find this information. Thus, hospital workers require access to information from anywhere within the hospital. Where hospital staff members are at a particular time determines in part the type of information they require. The current hospital’s boards can be substituted by large public displays. For instance, a public display should recognize a user as he approaches it and give him access to relevant clinical data without cumbersome login procedures.

2. **Delivery timing.** Communication exchanges in a hospital tend to be time sensitive, which means that a message might be relevant for only a certain period. For example, a doctor might leave a message that describes recommendations for a treatment to any nurse on the next shift. So, the system should be able to deliver the message when the nurse on the next shift is with the patient that should receive the recommended treatment.

3. **Role reliance.** In hospitals, parties who might be strangers or rarely meet must communicate with each other. A user often addresses messages not to particular individuals but to “the nurse on the afternoon shift,” or “the next doctor to visit the patient.” Thus, the system must be able to recognize roles as well as particular individuals.

4. **Artifact state.** An artifact, particularly a device, can have many states. The state of devices (temperature reading) and other artifacts (availability of lab results) can be important triggers for appropriate actions, including information exchanges. Medical staff might need to communicate directly with documents or devices. For example, a doctor might want to display the patient’s lab analysis on her office desktop as soon as results become available.
3.2 Content Adaptation and Personalization based on Contextual Information

Contextual information, such as location, role, and identity, should be taken into account to adapt and personalize the presentation of information to the user. For instance, when a physician is in front of a public display, the pervasive environment should display the messages addressed to him and enable him to visualize the patients’ records he is examining. Thus, information overload is prevented by personalizing the display to provide immediate access to the clinical records of those patients assigned to the doctor [10].

3.3 Information Transfer between Heterogeneous Devices

In the hospital, users frequently transfer information from public spaces to personal spaces. For instance, the chief nurse might leave a note on a public board in order to advertise the date of the next meeting; then, another nurse would write this information into her personal agenda. A few physicians actually carry Personal Digital Assistants (PDA) and reported using them to record information displayed on whiteboards or corkboards. A pervasive environment furnished with devices of all scales, should support the simple and safe transfer of information between devices of different scales. Some of these devices are public, such as large displays, while others, such as mobile devices like smart phones or PDAs, are personal. For instance, a doctor may want to transfer information from his PDA to a public display in order to discuss it with a colleague; and when the doctor approaches the board and is authenticated, the display will use information stored in the user’s PDA to personalize the information and applications running in the public display.

3.4 Context-aware Retrieval of Medical Information

The electronic patient record offers the opportunity to retrieve relevant medical cases with little effort from the user. To retrieve this information, the pervasive environment has to take into account the type of clinical problem. The recommended clinical cases have to be ranked by similarity with the current case. For instance, it may be more relevant for the physician to consult first how she solved previous cases akin to this, and after that, find out the treatment or diagnosis given by other clinicians to similar problems. The environment may also opportunistically display medical guides relevant to the patient’s diagnosis as supporting information for doctors or even locate and establish contact with a specialist who might be available [23].
4 Autonomous Agents for Designing Ambient Computing Systems

Software agents have been used in computer science from several perspectives. From a software engineering perspective, we adopt an agent-oriented approach for decomposing a problem into multiple, autonomous software agents that can act and interact in flexible ways to achieve their objectives. From the distributed systems stance, the technology of autonomous agents appears appropriate for building systems in which data, control, expertise, or resources are distributed [24]. Implementing a ubiquitous computing environment as a multi-agent system assists developers by hiding some of the complexities associated with 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 [25], the agent-oriented approach can be the natural way to deal with unpredictable associations and interactions among ubiquitous computing system components. Finally, one of the major research directions for Human-Computer Interaction (HCI) has been exploring the novel forms of interaction that can achieve Mark Weiser’s vision of naturally integrating computer technology with our daily activities. In this sense, autonomous agents can seamlessly assist users in their interactions with the ubiquitous computing environment in a variety of different ways: they hide the complexity of difficult tasks, they perform tasks on the user’s behalf, they can train or teach the user, they help different users collaborate, and they monitor events and procedures.

In the following scenario we illustrate how an agent-oriented approach enables developers to address some of the desirable features of ambient computing systems for hospitals. The devices used in the system illustrated in the scenario include mobile devices which enable users not only to access context-aware medical information, but to continually track their location. In addition, it includes public displays that present medical information to the hospital staff, such as X-ray images and medical guides. Particularly, the scenario shows how autonomous agents are the main system software entities that implement the desirable features of a ubiquitous hospital environment with regards to context-aware adaptation and personalization of medical information on public displays [10].

4.1 Usage Scenario of a Hospital Ambient Computing System

As Dr. García examines the patient in bed 234, the X-ray results he requested for the patient in bed 225 are added to the electronic record of the
Hospital Information (HI) System. The HI System agent, which provides access to, and monitors the state of the hospital information system, notifies that the X-ray results are available to the doctor by sending him a message. As illustrated in figure 1a), Dr. García approaches the nearest public display when he finishes with his current patient, and before visiting patient 225. The doctor’s location, which is constantly being tracked by the location-estimation agent on his mobile device (PDA), is notified to all users and agents in the environment. Thus, the Public Display agent which enables users to access the Public Display and have control over the applications, acknowledges the doctor’s presence by displaying his photograph, which indicates that he has been logged into the system. Then, the applications on the display are personalized for Dr. García. To reach this end, the Public Display agent requests the Map agent to personalize the public map of the hospital floor by indicating the location of hospital staff and services available (printer, public display, etc) and by highlighting the beds of patients assigned to the doctor as presented in figure 1b). The public map also shows messages addressed to the user that depend on his location. This includes messages related to his patients, such as additions to their electronic records (e.g.

Figure 1a) Autonomous agents’ interactions to personalize information on the public display. b) Personalization of the hospital map.

laboratory results). As Dr. García wants to examine the most recent X-rays of the patient in bed 225, he selects the “notification message” attached to bed 225 which indicates that the X-rays of this patient are available. The Public Display agent requests the latest additions to the electronic patient
record (EPR) to the HI System agent and these are presented to Dr. García by opening the X-ray image.

4.2 Autonomous Agents for Designing Ambient Computing Systems

Scenarios, like the one previously presented, were the basis to identify the requirements for SALSA [8][11]:

− Autonomous agents are decisions makers that 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. The middleware should provide mechanisms for implementing the agents’ components for perceiving, reasoning, and acting.
− Autonomous agents are reactive to the contextual elements of the environment. For this, agents need mechanisms to perceive, recognize and disseminate different types of context information such as role, location (of users and devices), state (of documents, services, devices, and users) and time.

Figure 2. SALSA Architecture
Autonomous agents should be able to communicate with other agents, or directly to users and services. For this, agents need a communication platform that enables them to convey information to other agents, users, and information resources by using the same protocol.

Agents need a communication language that enables them to convey different types of messages, such as messages for negotiating services offered by other agents, requesting information from devices or services, and responding to such requests. The communication platform should enable users to be aware of the presence of other users and agents that offer relevant services for them.

Autonomous agents can represent users, act as proxies to information resources, services and devices, or wrap a complex system functionality. Autonomous agents may have a reasoning algorithm as complex as the logic of its functionality. A reasoning algorithm may consist of a simple set of rules or a more complex reasoning. Thus, the middleware should provide abstract classes that enable developers to easily implement and modify the agents’ reasoning and actions.

5 The SALSA Middleware

Figure 2 presents a physical view of the architecture of the SALSA middleware which consists of several modules represented by packages and nodes. Packages include the collection of SALSA classes while nodes contain the components that represent the SALSA services. We first provide a detailed description of the architecture of SALSA before describing the SALSA API.

5.1 The Architecture of the SALSA Middleware

The SALSA middleware consists of the following modules:
- An Agent Broker which is a component that is responsible for coordinating the agents’ communication. The implementation of the Agent Broker is the Jabber Instant Messaging and Presence (IM&P) server (www.jabber.org). The IM&P server stores the status of people and agents and notifies their changes to other agents subscribed to them through XML messages. Developers can define new states according to the device or service the agent represents. For instance, an agent representing a printer may notify if it is busy or has a paper jam. This allows developers to provide a consistent interface for users to interact with other people, devices, and services.
- A Directory of Agents that allows programmers to Register and Initialize the agents by using XML messages. The SALSA class framework
provides a set of classes that facilitates the use of the services offered by the Agent Directory.

- The Client and Utilities packages provide the classes to facilitate the development of the agent’s proxy to the Broker and to parse the messages communicated via the Broker respectively.

- A class framework represented by a stereotyped package in figure 2, which includes the collection of SALSA classes. These classes are grouped in collaborations that shape the way in which they were organized. The main classes provided by SALSA enable the implementation of the internal architecture of an agent as well as control its life cycle. Thus, the collaborations of classes represent the mechanisms to implement the agent’s components for perceiving information, reasoning and acting. The complete set of classes provided by SALSA are depicted in figure 3 and explained in the following section.

![Figure 3. SALSA Class Framework](image)

### 5.2 SALSA Class Framework

The Perception component gathers context information from the sensors in the environment, or directly from the users, other agents or services. Two types of perception can be implemented with SALSA: active and passive. The
passive perception was implemented based on the Observer design pattern. 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 (see figure 3) represents the subject to be observed by the agent; and the `PassivePerception` class captures the information sent by the subject. The passive perception of a SALSA agent, in which data is received through its Broker Proxy (an IM client), is due to another agent that sends information by using the communication methods of the SALSA API. For the active perception, an agent decides on its own when to sense an environment entity. This type of perception implements the Adapter design pattern. The only active perception supported by SALSA, is when the agent perceives data directly from a sensor or device. When any of the perception components receive information, a SALSA event is generated indicating the type of information to the reasoning component. A SALSA event contains the perceived data. The information perceived by an agent is subtracted from the event by the reasoning component in order to be analyzed. The `Reasoning` class contains the abstract method `think()` that should be implemented by the developer with a reasoning algorithm, such as a simple condition-action rule or a neural network. 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 module. 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 facilitate composing, sending, and receiving 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. For instance, the `sendCommandRequest(mapa@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 one illustrated in figure 4a) 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, such as `userID`. 
5.3 Implementing an Ambient Computing System with SALSA

By revisiting the scenario presented in section 4, figure 4b depicts a sequence diagram illustrating how the autonomous agents of the system interact by using the SALSA communication protocol (the SALSA methods are in bold font style on the diagram). In this scenario, the Hospital Information System agent perceives a change on the medical records of the hospital information system and notifies it to Dr. García by sending a message \( \text{sendNotificationInfo} \) indicating that the X-ray results of the patient on bed 225 are available. When the location-aware client receives this message, it will act by updating its instant messaging interface. Dr. García approaches the nearest public display when finished with her current patient and before visiting patient 225. The doctor’s location, tracked by the location-estimation agent on her PDA, is notified \( \text{sendPresence} \) to all users and agents in the environment by the Location-aware client. To estimate the location of the mobile users of the 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. We were particularly interested in implementing a location estimation technique that makes use of an existing wireless LAN (Local Area Network) infrastructure, since it have better scalability and less installation and maintenance costs than ad-hoc solutions. More details of the neural network implementation are explained in [26].

When the Public Display agent perceives Dr. García’s location, it acts by displaying the user’s photograph, indicating with this that he has been logged into the system. Then, the Public Display agent requests \( \text{sendCommandRequest} \) to the Map agent that personalizes the map application for Dr. García. Finally, the Public Display agent also requests \( \text{sendRequest} \) the contextual messages recently received to the location-aware client, and displays them on the floor map. Thus, the physician can continue accessing the records of his patients and other medical information by interacting with the public display.

SALSA was used to implement this system by extending the functionality of our Context-aware Hospital System (CHIS) which included agents that enable the context-aware communication among mobile users [9]. The main components of this ambient computing system, implemented to assist medical activities, are SALSA agents that respond autonomously in accordance with the context surrounding the activities performed at the hospital. Similar ambient computing scenarios were used to evaluate SALSA during in-lab sessions, as described in the following section.

SALSA facilitates the implementation and maintenance of ubicomp systems since its communication platform and the agent’s execution model
enable to implement a ubicomp system as a set of decoupled components (autonomous agents). The architecture of SALSA is based on the instant messaging (IM) paradigm, which allows a standardized form of interaction among users and services represented by autonomous agents. In the same form, users are aware of the presence of other users, they are also aware of the presence and state of the environment’s services and devices. The scalability of a system implemented with SALSA is enabled by the IM server used as an Agent Broker, since it scales to a high volume of streaming XML connections serving hundreds of thousands of simultaneous users and agents. By using SALSA, developers may easily integrate any reasoning mechanism, or change the reasoning algorithm, without modifying the code of the rest of the agent’s components. For instance, we may change the neural network of the Location estimation agent with a nearest neighbor algorithm without altering the perception and acting components. SALSA agents have well-defined interfaces to interact with their environment, and mechanisms to encapsulate its implementation. For this reason, these agents may be considered as units of independent deployment or components, which may be re-used or integrated to any SALSA ubiquitous system, facilitating thus, the progressive development or evolution of ubiquitous computing systems.

**Figura 4**

a) XML-based format of a SALSA message. b) SALSA Agents interacting for retrieving medical information.
5.4 Creating Real Ubicomp Applications with SALSA

Scenarios as the ones presented in section 4, enabled us to visualize how medical practices could be augmented with ubicomp technology. These scenarios were grounded in workplace studies in which we observed the activities of users in their work settings [9][10]. Afterwards, the ubicomp applications depicted in some of the scenarios were evaluated by to validate that they represented actual user activities, and to identify new opportunities for the use of ubicomp technology and autonomous agents.

The evaluation results show that physicians are attracted to the idea of having computerized access to patient records and favored the mobility offered by PDAs for clinical records, and large displays for lab results because of the limited screen size of handhelds. In addition, the need to share information with other people favored large displays over traditional personal computers. More information regarding the methodology followed in this evaluation and the results obtained are reported in [10].

The evaluation to the application scenarios determined not only that the use of ubicomp systems were considered realistic by the hospital personnel, but that these systems were also perceived as being useful for supporting their daily work activities. The results from this evaluation showed that SALSA enabled the building of real and useful pervasive systems and facilitated their implementation as presented in section 5.3. This offers evidence of usefulness of implementing this type of pervasive systems with autonomous agents. The following sections evaluate SALSA’s flexibility and ease of use for developing autonomous agents for ubicomp systems.

6. Evaluating the SALSA API

Inspired by empirical studies for evaluating infrastructures used for implementing interactive systems [4][22], we identified that evaluating the ease of use of a middleware should include the readability of the programs written with the middleware’s programming language, how learnable they are, how convenient they are for expressing certain algorithms, and how comprehensible they are to novice programmers [22]. We conducted two experiments to evaluate the ease of use of SALSA for programming agents that implement features similar to those identified for ambient computing in hospital settings, which were presented in section 3. The first experiment evaluated the API of SALSA in an in-lab experiment, and in the second one, we evaluated the use of SALSA agents as design abstractions for conceiving ambient computing systems. A group of graduate students taking the Object Oriented Analysis and Design (OOAD) class at the CICESE Research Center participated in these experiments. We decided to use graduate students in the
evaluation to gather preliminary evidence of the ease of use of SALSA for implementing ubiquitous computing systems. Students have been commonly used in software engineering evaluations in lieu of professional developers [27][28][29]. There is empirical evidence that graduate students tend to act and think like professionals in these types of evaluation, however this is not the case with undergraduates [29].

6.1 Ease of Use of the SALSA API

This experiment was conducted to: “Evaluate the ease of use of the SALSA class framework to develop ubiquitous computing healthcare applications.” To achieve this objective, an experiment conducted in three separate sessions was carried out.

In the first session, we assessed the participants’ experience in developing software systems. Sixteen (16) students of a group of nineteen (19) students were selected to participate in the experiment. The results from this survey indicated that the participants did not have the same level of knowledge in some issues relevant to the experiment, such as the use of autonomous agents (25% stated that they had no idea or only a vague idea of what an agent is). Their programming and design background was evaluated in order to adapt the experiment to their level of expertise by providing information they would need during the experiment. In the second session we explained to them the concepts of ubiquitous computing, software agents, SALSA API, including a programming example using SALSA, and the in-lab experiment in which they were going to participate. The evaluation session was conducted in a computing laboratory.

6.1.1 In-lab Evaluation

During this evaluation all (19) nineteen students from the OOAD course participated. Through three programming tasks, participants were asked to implement and extend the functionality of one of the autonomous agents of a ubicomp system that monitors the state of a patient with diabetes, and based on this information, decide on a course of action. During the experiment, we recorded the participants’ comments and questions, and saved their code for further review. When the participants finished their programming exercises, they were asked to answer a questionnaire to evaluate the ease of use of the SALSA API. To illustrate the desired functionality of the system, the following scenario was provided to the participants:

“George, a patient with diabetes, is alone at home watching a soccer game. Suddenly, he feels sick and the ubicomp system detects that he has hypoglycemia. That is, his glucose levels are low (76mg/dl). Thus, the system
recommends him to take 2 tablespoons of sugar. Later, the system measures the glucose level to be lower (70 mg/dl) and starts monitoring the patient’s pulse and his level of perspiration, which at that moment are normal. At the same time, the system notifies George’s daughter of his condition. The system continues monitoring George’s vital signs. As he still presents hypoglycemia, and his pulse and level of perspiration increase, the system decides to send a warning message to George’s doctor.

```
5     public class AgentePaciente extends Agent{
6         JabberClient jc;
7     
8     public AgentePaciente(){
9         //Specify agent's attributes
10         attributes.setJabberID("marcorod@pc-coolab3");
11     
12         //Jabber client
13         jc=new JabberClient("pc-coolab3","5222","marcorod","anroll26");
14     
15         //Initialize the reasoning component:
16         HelloWorldReasoning reasoning=new HelloWorldReasoning();
17     
18         //Create and activate agent's components:
19         activate(reasoning, jc);
20     }
21  
22     public static void main(String[] args) {
23         AgentePaciente agentePaciente = new AgentePaciente();
24     }
25  
26 }
```

Figure 5. Code of the patient’s agent

The main components of the system depicted in the scenario were modeled with three autonomous agents acting as proxies to the sensors, which monitor the levels of glucose, pulse, and perspiration, respectively. These agents send the information detected to a fourth agent, the patient’s agent, which determines the patient’s health-state and decides what action to execute. Participants were asked to develop the patient’s agent. For the other agents the participants were given executables that simulated that they were actually perceiving information from the sensors. Thus, the conditions of the patient, such as hypoglycemia (low glucose) or hyperglycemia (high glucose), were simulated, which enabled the participants to verify if their agent acted as expected. We gave the participants the code of the general structure of the patient’s agent presented in figure 5. This is a class that extends the `Agent`
SALSA class. In the constructor of this class an instance of the Perception component is created, which is the Broker Proxy (JabberClient) through which the agent will communicate with other agents and users. It also contains instances of the Reasoning component which will invoke the Action component. Thus, when an instance of this class is created, its components for perceiving, reasoning and acting are activated and the life cycle of the agent begins. As part of the programming experiment, participants had to implement the agent’s reasoning and action components according to the execution model of SALSA, and finally attached them to the agent’s patient presented in figure 5.

```java
public class AgentePacienteReasoning extends Reasoning{
    public void think(EventObject ev){
        SAlsa.Events.Event evento = (SALSA.Events.Event) ev;
        if (evento.getType() == evento.ArrivedSensorDataEvent) {
            Utilidades.Parser p = new Utilidades.Parser( evento.xml );
            String tipoSolicitud = p.getTag( "type" );
            if ( tipoSolicitud.equals( "glucose" ) ) {
                int glucose = Integer.parseInt( p.getTag( "glucose" ) );
                if ( glucose > 120 && glucose < 140 )
                    acting.act( new pacienteAgentRecommendation() );
                else if ( glucose > 140 )
                    acting.act( new pacienteAgentActionHospital() );
            }
        }
    }
}
```

**Figure 6.** Agent’s reasoning diagnosing hyperglycemia.

Task 1 asked the students to code the conditions for the patient’s agent to diagnose hyperglycemia and the patient’s agent should take an appropriate action. We expected the participants to implement the reasoning component of the agent, as presented in figure 6, with the appropriate conditions to determine that the patient had hyperglycemia. Participants also were asked to implement the agent’s actions which included that the agent recommends the patient to drink two glasses of water as illustrated in figure 7a. If the agent
again detected hyperglycemia 15 minutes later, it has to notify the patient’s doctor by sending a warning message to him, as illustrated in Figure 7b. As shown in Figure 6, the Reasoning of the patient’s agent is specialized from the Reasoning abstract class of SALSA. Its think() method should be 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 measure of the user’s glucose level was taken. The actions that can be invoked from the reasoning component are implemented in the classes: PatientAgentRecommendation and PatientAgentActionHospital depicted in figure 7a and 7b respectively. These classes extend the abstract SALSA class Action and overwrite its execute() method from which the SALSA communication methods can be invoked. For instance, in the PatientAgentActionHospital class it uses the method sendMessage() to notify a doctor of the patient’s condition.

```java
public class patientAgentRecommendation extends Action {
    public Object execute() {
        System.out.println("Tome al menos dos vasos de agua!");
        return null;
    }
}
```

```java
public class patientAgentActionHospital extends Action {
    public Object execute() {
        sendMessage("jvlez@lap-psantana",
                    "Jorge Sanchez presenta hiperglicemia grave!", "chat", "");
        return null;
    }
}
```

Figure 7. Agent’s actions when hyperglycemia is detected. a) The agent recommends drinking water by printing a message in the user’s console. b) The agent notifies the doctor about the patient’s condition by sending an IM message.

In task 2, participants were asked to modify the agent to detect hypoglycemia by using the facilities provided by SALSA to derive context. We provided the participants with the code of the XSL filter with the
conditions to detect hypoglycemia as presented in figure 8. The XSL filter reads the primary context, which in this case was the information of the levels of glucose, perspiration and pulse, and then deduced a secondary context which indicated that the patient was having hypoglycemia.

**Figure 8. XSL filter to detect hypoglycemia**

Figure 9 and figure 10 present how we expected the participants to code the reasoning and action components. The constructor of the class that implements the reasoning component (see figure 9) specifies that the variables used to derive contextual information are: the level of glucose, perspiration and pulsations. Thus, when an event of type `ArriveSensorDataEvent` is passed to the method `think()`, the context information is abstracted from the event by the `derive(ev)` method to return the context derived as an XML message in the `secondaryContext` variable. This message is then parsed to check the level of hypoglycemia of the patient. Thus, if the patient presents hypoglycemia for the first time, the agent acts by recommending a dose of sugar, and requests the other sensing agents to start monitoring the patient’s pulse and perspiration as illustrated in figure 10a. If the level of glucose does not stabilize and the patient continues presenting hypoglycemia, the agent notifies this to the patient’s doctor, as illustrated in the code in figure 10b.
public class patientAgentReasoning extends Reasoning {
    SALSA.SecondaryContext sc;

    public patientAgentReasoning() {
        String xmlFile = "C:\EVALUATION_SALSA\patientagent\secondaryContext.xsl";
        sc = new SALSA.SecondaryContext(xmlFile);
        sc.setContextualVariable("glucose", "30");
        sc.setContextualVariable("palpitations", "0");
        sc.setContextualVariable("mudaracm", "0");
    }

    public void think(EventObject ev) {
        SALSA.Events.Event event = (Event) ev;
        String secondaryContext = "";

        if (event.getType() == event.ActivitiesDataEvent)
            secondaryContext = sc.derive(ev);

        Utilidades.Parser p = new Utilidades.Parser(secondaryContext);
        String context = p.getText("context");
        String tipo = p.getText("tipo");

        if (tipo.equals("hipoglucemia")) {
            switch (Integer.parseInt(context)) {
            case 1:
                action.act(new patientAgentActionRecommend());
                break;
            case 2:
                action.act(new patientAgentActionHospital());
                break;
            }
        }
    }
}

Figure 9. Reasoning component detecting hypoglycemia

Finally, in task 3, we required the participants to modify the XSL filter to include the same conditions to detect hyperglycemia as was indicated in task 1.
Figure 10. Agent’s actions when hypoglycemia is detected. a) The agent recommends the patient to eat a dose of sugar. b) The agent notifies the physician the patient’s health-condition

6.1.2 Results from the Inspection of the Code

The source code of the patient’s agent produced by the students was analyzed to evaluate whether they comprehended the execution model and the communication protocol of SALSA agents. Only five (5) participants did not distribute the functionality of the agent as we expected. These participants implemented part or the whole reasoning logic (a set of rules) in the action component as illustrated in figure 11, in contrast with the code presented in figure 6 and 7. For this, the information perceived in the reasoning component
was passed to the action component to detect the patient’s health condition and decide how to act, rather than having the reasoning component do this. Three (3) of the participants failed at implementing the perception of information. They did not verify whether the event received in the reasoning component was of type `ArriveSensorDataEvent`, which means that any perceived message, such as a presence message, is analyzed by the reasoning component to find out the patient’s condition. Most of the participants were able to successfully implement each of the agent’s components, but not without some difficulties. Participants were allowed to check SALSA’s user manual during the study or they could ask questions to any of the two graduate students running the experiment. Even though some of the participants understood the execution model of SALSA, they did not remember the name of the abstractions and methods of the SALSA API explained during the 2-hours course. Among the main questions they made were: “What are the methods for communicating information to other agents?” and “What are the events received in the reasoning object?” But, the main concerns of the participants during the first activity were related to XML. Their major questions were about how to abstract the information from the message by using the facilities provided by SALSA since the majority of the participants were familiar with HTML, but not with XML. During the second and third activity the participants’ main questions were about XSL and the facilities offered by SALSA to derive secondary context using an XSL filter, which was new for most of them.

```java
public class AgentesPacienteAction extends SALSA.Action{
    String type,value;

    public AgentesPacienteAction(String type, String value) {
        this.type=type;
        this.value=value;
    }

    public Object execute(){
        if (type.equals("glucose")) {
            if ((Integer.parseInt(value) > 120) &
                (Integer.parseInt(value) < 140))
                System.out.println("tome al menos 2 vasos de agua");
            if (Integer.parseInt(value) >= 140)
                sendMessage("hombre@lo-pantano",
                           "Jose Sanchez presenta hiperglicemia grave",
                           "chat", "");
        }
        return null;
    }
}
```

Figure 11. Action component implemented by one of the participants.
6.1.3 Perception of Ease of Use

When the participants finished their programming tasks, they completed a survey which included topics related to their perception on the use of SALSA. The survey included a questionnaire for evaluating the perceived ease of use of the SALSA API which included six assertions, each one with 7 Likert-scale answers [13]. The participants perceived the API to be easy to use since most of them either “agree” (6) or “slightly agree” (5) with all the assertions as illustrated in Table I. Even though some of the participants “slightly agree” that learning (Q1) and interacting (Q3) with SALSA was easy (5.27) and understandable (5.267), most of them “agree” that it would be easy for them to become skillful at using the SALSA API (Q5). The Action component was considered as the agent’s functionality that is the easiest to implement.

Table I. Results of TAM questionnaire

<table>
<thead>
<tr>
<th>Questions</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Learning the SALSA API would be easy for me.</td>
<td>5.27</td>
</tr>
<tr>
<td>Q2: I would find it easy to implement intelligent systems with SALSA</td>
<td>5.667</td>
</tr>
<tr>
<td>Q3: My interaction with SALSA would be clear and understandable</td>
<td>5.267</td>
</tr>
<tr>
<td>Q4: I would find SALSA to be flexible to interact with</td>
<td>5.4</td>
</tr>
<tr>
<td>Q5: It would be easy for me to become skillful at using the SALSA API</td>
<td>5.8</td>
</tr>
<tr>
<td>Q6: I find SALSA easy to use</td>
<td>5.467</td>
</tr>
</tbody>
</table>

Two final questions were included to find out the agent’s functionalities that were perceived as easier to implement, and which ones were considered the most difficult ones. The Action component was considered as the agent’s functionality that is easiest to implement. For instance one of the participants claimed so “because it does not need so many lines of code for implementing the communication of the agent”; another participant stated: “you just need to select the appropriate method of communication according to the type of information to convey”. Implementing the Reasoning component was considered easy by eight (8) of the participants. One subject stated that “it was easy, because you just have to overwrite the reasoning logic”, which means that once they were familiar with the implementation of the first programming activity, they considered that it was easy to modify the agent’s behavior for the second and third activity. The agent’s functionality that was considered the most difficult to implement, by six (6) of the participants, was to derive context information by using the XSL filter. This is comprehensible, since these persons had no previous experience with XML and XSL, as they
stated: "I have never used XSL files", and "programming in XML is new to me". For the same reason, some of the subjects stated that the reasoning component was also difficult to implement. One of them commented: "It was difficult to parse the XML message".

6.2 Discussion

Even though participants had concerns about the SALSA API to implement some of the functionalities of the agent during the programming exercise, most of them could successfully create the agent as we expected. This provides evidence that the execution model of SALSA and the facilities to implement it are comprehensible. For some of the participants the use of autonomous agents as an abstraction for the deployment of an ubicomp system was not innate. The problems faced by some of the participants for distributing the agent’s functionality among the action and the reasoning components were due to their lack of experience in implementing SALSA agents and that they were not familiarized with the execution model of SALSA (defined in the SALSA agent’s life cycle). However, the participants faced minimum problems for implementing and understanding the functionality of the perception component. This was due to the fact that most of the agent’s perception is left to the SALSA infrastructure which automatically creates and activates this component when the agent is instantiated, and the programmer only has to extract the received information from the launched event. Thus, even tough some of the participants did not remember how to capture the event generated when the information is passed to the reasoning component, this was easily solved by consulting the documentation of SALSA. Participants had to understand that agents communicate in a different way than other type of components, such as objects or processes. The subjects perceived that the communication protocol was easy to understand. They expressed that to implement the agent’s communication they just had to select the appropriate methods according to the information that they want to convey. The difficulties experienced by the participants for implementing the agents’ communication was due to agents using a communication language that is semantically richer than that commonly used by object technology. The SALSA communication language is based on XML, which is too verbose and adds additional complexity to the API of SALSA, a fact that was also noticed by the participants with no previous experience parsing XML messages. Even though SALSA provides some facilities for extracting the content of a SALSA message, the lack of knowledge in XML by the subjects did not enable them to easily comprehend the syntaxes of the SALSA messages. In the same way, participants also experienced difficulties using XSL since they had no previous experience
with it. This did not enable participants to perceive that the mechanisms in SALSA to derive secondary context were easy to use. Thus, we conclude that the SALSA API needs to include mechanisms at a higher level of abstraction to facilitate the management of XML messages and the creation of the XSL filter.

7 Evaluating SALSA Agents for Designing Ambient Computing Systems

The objective of the second evaluation experiment was: “To evaluate the ease of use of SALSA agents in designing a ubicomp system”. During this experiment, eighteen (18) graduate students (of the nineteen) participated in a design problem of an ambient computing system by using autonomous agents. For this, the design problem was included in the evaluation exam of the OOAD course taken by these students. As presented in figure 12 the exam included a description of a usage scenario of a ubicomp system for an Airport setting, a sequence diagram of the interactions among the system agents, and the following four exercises (a-d): First, participants had to analyze the description of the ubicomp system and its sequence diagram to identify the main components of the system (a). They were asked to extend the application functionality elaborating a sequence diagram (b) and a component diagram (c). Finally, they had to provide a detailed description of each of the agents’ components (d). The total score of the activities of the design exercise was 50 points. Each exercise was reviewed to verify if it fulfilled a set of conditions. If a condition was not achieved, the value of the exercise was decremented by a pre-establish penalization value.

7.1 Results

The diagram presented in figure 13 illustrates how the system can be designed by using autonomous agents. It presents a version of the Sequence Diagram that illustrates the interaction of the agents. It shows that the functionality of the system was extended by adding two agents that enable the user to access the available services: the Agent Directory which acts as a proxy to the Airport Directory by providing information of all the available services and the map of the airport; and the Library Agent which enables the user to access information of the Library. In this design the communication protocol of SALSA was used to enable the interactions among agents. The next sections describe the results of the experiment exercises.
The Mexico City airport has an ubiquitous computing system that allows users to consult flight information in a public display or the user’s handheld computer (PDA). The following sequence diagram illustrates the interaction of the components of the system, which have been implemented as SALSA agents. You have been hired to extend the functionality of the system with SALSA agents, so that the system can also recommend services available within the airport that could be useful to the passengers while they wait for their flight. The system should have the following functionality:

- When Mr. Jorge Gomez enters the airport’s waiting room, his PDA notices him that his flight has been delayed by 40 minutes. The system in the PDA personalizes the map according to the user’s preferences and available time, highlighting services such as restaurants, bookstores, internet café, etc.
- A scenario of use of the extended system will be as follows:
  - “When Mr. Jorge Gomez enters the airports waiting room, his PDA notices him that he has received a message indicating that his flight has been delayed by 40min. The system in the PDA requests to the airport’s Service Directory those services that are available to determine those that could be useful to Mr. Gomez and that he could take advantage of while he waits for his flight. The Service Directory provides this information and a map with the location of the services. The PDA, knowing the time he has and his preferences, elects to highlight the location of the Vips restaurant and the ‘Mexico’ bookstore, since Mr. Gomez needs to buy a book for his daughter. Mr. Gomez goes to the restaurant and while he waits for his food, accesses information of the bookstore, selecting in the map a link to the bookstore’s web page to consult the availability of the book he wants.”

You are asked to:

a. Create a diagram showing the components of the original system. (10 pts.)
b. Modify the previous sequence diagram, incorporating the components that implement the new functionality required by the system. (20 pts.)
c. Modify the components’ diagram elaborated in question a. to incorporate the new components of the extended system. (10 pts.)

Describe the sequence diagram you have extended, explaining the behavior of the agents. That is, what functionality is implemented by each of the components of each agent (perception, reasoning, action) (10 pts.)

**Figure 12.** SALSA agents interacting to provide flight information to airport users
7.1.1 Exercise a: Create a diagram showing the components of the original system

The average grade of all participants was 7.67 out of 10 points. The most common mistake was related to conceptual problems for modeling Component Diagrams with UML. For instance, most of the participants (13) made mistakes in establishing the relationships among the components. Two (2) persons described the system functionality in terms of components or subsystems instead of agents in exercise d. Although most participants said they had a vague idea of what an agent is and of its application, the majority of them (16 persons) had no difficulty identifying the agents of the proposed system as components which wrap the main functionality of the ubicomp system.

7.1.2 Exercise b: Modify the previous sequence diagram, incorporating the components that implement the new functionality

The value of exercise b) was 20 points. The group averaged 16.22 points. The most frequent mistake was that the scenario was incomplete: Seven (7)
participants did not include the interaction with the library service which had to be represented by a proxy-agent; two (2) participants did not specify the interaction to personalize the map for the user’s preferences; and one (1) participant did not include the Service Directory, but the User Agent requested the available services to the Information System Agent. In spite of the fact that two weeks before the participants made a laboratory practice using SALSA, not all of them comprehended the scope of SALSA agents to extend the functionality of a system. This, may be because they were not familiar with the use of agents for developing systems. The second major error in which five (5) participants incurred was that the Airport Agent was used as a communication intermediary between the User Agent and the Service Directory. In spite of this, the protocol of interaction of SALSA agents was well expressed in the diagram sequence for the majority of the participants. They correctly and clearly specified the agents’ interactions to send or request information to other agents. Even though they were not requested to use the real names of SALSA’s methods for communication, several of them (8 persons) remembered and used them appropriately.

7.1.3 Exercise c: Modify the components diagram created in exercise a, to incorporate the new components of the extended system

The group average for this exercise was 6.83 out of 10 points. To grade this exercise, we did not take into account the conceptual problems in which participants incurred in solving exercise a. Nine (9) persons did not identify the Library Service as an agent, even though two (2) of them included this agent in the sequence diagram. Two (2) of these nine (9) students did not include the Directory Service in the component diagram. Some of the mistakes made in this exercise were due to incorrect design decisions made in the sequence diagram. For instance, if the participants did not indicate the interactions with a proxy-agent to the Library in the sequence diagram, they also were not included in the component diagram. However, the other nine (9) participants created agents to add the new functionality to the system following the approach suggested by SALSA to develop ubicomp systems.

7.1.4 Exercise d: Describe the sequence diagram you have extended, explaining the behavior of the agents

From this exercise, we expected participants to be able to explain the functionality the agents’ components for perceiving, reasoning and acting. The average of the group for this exercise was 7.61 of 10 points. Five (5) of the participants failed to explain each of the components of the system’s
agents, but they gave a general description of each of the system agent. Most of the participants (14) described the agents components clearly. They stated how and when the agents perceive; reason based on the perceived information; and specified what events trigger an agent’s action. One of them envisioned that the user’s agent may perceive from the to-do list of his PDA that he had to buy a book for his daughter.

7.2 Discussion

The results of exercise a) show that the participants could abstract the agents as the main system building blocks from the ubicomp application scenario. However some of them (8 persons) did not model correctly the extended system as requested in exercise c) since they did not identify all the agents’ interactions in exercise b). The results of this evaluation show that using autonomous agents for designing a ubicomp system requires developers to clearly understand the agent-oriented programming paradigm. This evaluation also shows that even though the students had learned about the use of SALSA agents approximately 2 weeks before, they were able to identify the agent’s components and some of them used the SALSA methods for communication.

8. Conclusions and Future Work

A ubiquitous computing environment is characterized by the distribution, reactivity, collaboration and adaptation of their artifacts, thus sharing these characteristics with autonomous agents. This provided us with the motivation to explore the use of agents as an abstraction tool for the design and implementation of ubiquitous computing systems. We used autonomous agents as a technique to model and design ubiquitous computing systems since they provide a natural and elegant means to manage the system’s complexities. We identified the design issues of autonomous agents for designing and implementing ubicomp systems for the healthcare domain. These design issues were the foundation for creating the SALSA middleware that provides the mechanisms to facilitate the development of UbiHealth systems. Thus, we have presented how the SALSA middleware facilitates the implementation of ubiquitous computing systems for hospital settings in which autonomous agents are the proactive components that enable users to seamlessly and opportunistically interact with the users, devices and services of the environment.

Finally, we conducted an evaluation for assessing SALSA thru a programming experiment and a design exercise. The results of the
programming exercise provided evidence that the execution model of SALSA and the facilities to implement it are comprehensible. 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 to agents and the facilities provided by SALSA, such as the agents’ use of XSL for deriving context information. The results of the design exercise show that even though the students had learned about the use of SALSA agents approximately 2 weeks before, they were able to identify the agents as the main system building blocks from the ubicomp application scenario, and some of them used the SALSA methods for agent communication. Thus, autonomous agents were used by the participants as the main building blocks or components which wrap the main functionality of the ubicomp system.

During the evaluation of SALSA, one of the issues that arose was that the middleware needs to facilitate the creation of the XSL filter which enables the agents to predict the context situation based on a simple set of conditions. Besides that, the context information that can be derived by SALSA is limited to the conditions that can be specified in the XSL filter. Based on these findings, we are exploring the use of an ontology to infer high-level context. We plan to incorporate to SALSA appropriate mechanisms for modeling context information through OWL based ontologies and rule-based agents that will access these ontologies to infer secondary context information. For designing the context representational model we use an ontology that is general enough for activity-aware computing systems in different domains, such as hospitals and ambient assisted living [30]. Thus, we are creating the CARe ontology (Context-Aware Representation of e-activities) which represents general concepts such as activities, location, used objects; and the ELDeR ontology (Enabling Living inDependently of Risks) which is intertwined with the CARe ontology and includes specific concepts regarding how the elderly carry out activities of daily living at their homes, such as type of Activity of Daily Living (i.e. basic, instrumental) and community that helps elders to execute their activities. We are currently extending SALSA by including services and classes that facilitate how developers implement agents that consult and update the CARe ontology. The use of ontologies will support developers in designing the structure of the content of the communication messages of SALSA agents.

Finally, the use a centralized broker may not be appropriate to handle the unpredictable mobility and disconnections of mobile clients in a ubiquitous computing environment. To enhance the broker-based communication platform of SALSA, we are considering other alternatives such as a distributed network of brokers as proposed in [31].
References


