Applying a service-oriented approach for developing a distributed multi-agent system for healthcare

Dante I. Tapia*, Juan Francisco De Paz, Sara Rodríguez, Cristian Pinzón, Rosa Cano, Javier Bajo and Juan M. Corchado

Departamento de Informática y Automática,
Universidad de Salamanca,
Plaza de la Merced s/n, 37008,
Salamanca, Spain
E-mail: dantetapia@usal.es
E-mail: fcofds@usal.es
E-mail: srg@usal.es
E-mail: cristian_ivanp@usal.es
E-mail: rcano@usal.es
E-mail: jbajope@usal.es
E-mail: corchado@usal.es
*Corresponding author

Abstract: This paper presents a service-oriented architecture that allows a more efficient distribution of resources and functionalities. The architecture has been used to develop a multi-agent system aimed at enhancing the assistance and healthcare for Alzheimer patients living in geriatric residences. Most of the system functionalities have been modelled as independent and distributed services, including reasoning, planning and security mechanisms. The results obtained after testing the architecture in a real healthcare scenario demonstrate that a service-oriented approach is far more robust and has better performance than a centralised one.

Keywords: multi-agent systems; service-oriented architectures; case-based reasoning; case-based planning; healthcare; security.

Reference to this paper should be made as follows: Tapia, D.I., De Paz, J.F., Rodríguez, S., Pinzón, C., Cano, R., Bajo, J. and Corchado, J.M. (2010) 'Applying a service-oriented approach for developing a distributed multi-agent system for healthcare', *Int. J. Computer Applications in Technology*, Vol. 39, No. 4, pp.234–244.

Biographical notes: Dante I. Tapia received a PhD in Computer Science from the University of Salamanca (Spain) in 2009. He obtained the Engineering degree in Computer Sciences in 2001 and the MSc in Telematics from the University of Colima (Mexico) in 2004. He has collaborated with the Government of the State of Colima (Mexico), where he obtained a scholarship to complete his academic formation. He has been involved in several R&D projects. He has been a member of the organising and scientific committee of several international symposiums. He has also been co-author of papers published in recognised journals, workshops and symposiums.

Juan Francisco De Paz is a PhD student at the University of Salamanca. He obtained a research scholarship from the Science and Education Ministry, and is involved in several R&D projects. He obtained the Engineering degree in Informatics in 2005 and the Statistics Diploma in 2007 from the University of Salamanca. He has been a member of the organising and scientific committee of several international symposiums. He has also been co-author of papers published in recognised journals, workshops and symposiums.

Sara Rodríguez is a PhD student at the University of Salamanca. She obtained the Engineering degree in Informatics from the University of Salamanca in 2007. She has been a member of the organising and scientific committee of several international symposiums. She has also been co-author of papers published in recognised journals, workshops and symposiums.

Cristian Pinzón is a PhD student at the University of Salamanca. He obtained a research scholarship from the National Secretary of Science, Technology and Innovation (SENACYT-Panama) to complete his academic formation. He obtained the Bachelor degree in Informatics from the Technological University of Panama (Panama) in 2003. He obtained a Master in Intelligent Systems from the University of Salamanca in 2007. He has been co-author of papers published in recognised journals, workshops and symposiums.

Rosa Cano is a PhD student at the University of Salamanca. She obtained the Engineering degree in Informatics in 1986 and the MSc in Computer Sciences from the Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM), México, in 1991. She has collaborated with the government and private industry in México. She has been involved in several R&D projects. She has been a member of the organising and scientific committee of several international symposiums. She has also been co-author of papers published in recognised journals, workshops and symposiums.

Javier Bajo received a PhD in Computer Science from the University of Salamanca in 2007. He is an Assistant Professor at the University of Salamanca. He obtained the Information Technology degree from the University of Valladolid (Spain) in 2001 and Engineering in Computer Sciences degree from the Pontifical University of Salamanca (Spain) in 2003. He has been a member of the organising and scientific committee of several international symposiums. He has also been co-author of papers published in recognised journals, workshops and symposiums.

Juan M. Corchado is Dean of the Faculty of Sciences at the University of Salamanca. He received a PhD in Computer Science from the University of Salamanca in 1998 and a PhD in Artificial Intelligence from the University of Paisley, Glasgow, UK, in 2000. He has led several artificial intelligence research projects sponsored by Spanish and European public and private institutions and has supervised several PhD students. He is co-author of over 130 books, book chapters, journal papers, technical reports, etc. He has been the President of the organising and scientific committee of several international symposiums.

1 Introduction

The continuous growth of the internet requires frameworks for web application integration (Oren et al., 2007). Web applications are executed in distributed environments, and each part that composes the program can be located in a different machine. The absence of a strategy for integrating applications generates multiple points of failure that can affect the systems' performance. Some of the technologies that have acquired a relevant paper in the web during the last years are the multi-agent systems and Service-Oriented Architectures (SOA). This work describes a novel architecture for developing multi-agent systems and explains how it has been designed and applied to a real scenario. The architecture presents important improvements in the vision of the integration of web applications. One of the most important characteristics is the use of intelligent agents as the main components in employing a service-oriented approach, focusing on distributing the majority of the systems' functionalities into remote and local services and applications. The architecture proposes a new and easier method of building distributed multi-agent systems, where the functionalities of the systems are not integrated into the structure of the agents; rather they are modelled as distributed services which are invoked by the agents acting as controllers and coordinators.

Agents have a set of characteristics, such as autonomy, reasoning, reactivity, social abilities, pro-activity, mobility, organisation, etc., which allow them to cover several needs for highly dynamic environments. Agent and multi-agent systems have been successfully applied to several scenarios, such as education, culture, entertainment, medicine, robotics, etc. (Corchado et al., 2008b). The characteristics of the agents make them appropriate for developing dynamic and distributed systems, as they possess the capability of adapting themselves to the users and environmental characteristics (Jayaputera et al., 2007). Most of the agents are based on the deliberative Belief, Desire, Intention (BDI) model (Wooldridge and Jennings, 1995), where the agents' internal structure and capabilities are based on mental aptitudes, using beliefs, desires and intentions (Bratman, 1987). Nevertheless, complex systems need higher adaptation, learning and autonomy levels than pure BDI model (Bratman, 1987). This is achieved by modelling the agents' characteristics (Wooldridge and Jennings, 1995) to provide them with mechanisms that allow solving complex problems and autonomous learning. Some of these mechanisms are Case-Based Reasoning (CBR) (Aamodt and Plaza, 1994) and Case-Based Planning (CBP) (De Paz et al., 2008), where problems are solved by using solutions to similar past problems (Corchado et al., 2008a; Corchado et al., 2008b). Solutions are stored into a case memory, which the mechanisms can consult in order to find better solutions for new problems. CBR and CBP mechanisms have been modelled as external services. Deliberative agents use these services to learn from past experiences and to adapt their behaviour according the context.

This paper briefly describes a *Flexible User and ServIces Oriented multi-ageNt Architecture* (FUSION@), a service-oriented alternative for distributed multi-agent architecture. This architecture has been used for developing ALZ-MAS 2.0, a multi-agent system aimed at enhancing the assistance and healthcare for Alzheimer patients living in geriatric residences. ALZ-MAS 2.0 is based on FUSION@ and implements a services oriented approach, where functionalities, including CBR and CBP mechanisms, are not integrated into the structure of the agents, rather they are modelled as distributed services and applications which are invoked by the agents. This paper also describes the security mechanisms used for protecting sensitive information.

In the next section, the problem description that motivated this work is presented. Section 3 briefly presents the FUSION@ architecture and different functionalities. Section 4 describes the basic components of ALZ-MAS 2.0 and shows

how this system has been designed according the distributed approach defined by FUSION@. Finally, Section 5 presents the results and conclusions obtained in this work.

2 Problem description

Excessive centralisation of services negatively affects the systems' functionalities, overcharging or limiting their capabilities. Classical functional architectures are characterised by trying to find modularity and a structure oriented to the system itself. Modern functional architectures like SOA (Service-Oriented Architecture) consider integration and performance aspects that must be taken into account when functionalities are created outside the system. These architectures are aimed at the interoperability between different systems, distribution of resources and the lack of dependency of programming languages (Cerami, 2002). As described by OASIS (2008), "A SOA-based system is a network of independent services, machines, the people who operate, affect, use, and govern those services as well as the suppliers of equipment and personnel to these people and services". The term *service* can be defined as a mechanism that facilitates the access to one or more functionalities (e.g. functions, network capabilities, etc.). Services are linked by means of standard communication protocols that must be used by applications in order to share resources in the services network (Ardissono et al., 2004). The compatibility and management of messages that the services generate to provide their functionalities is an important and complex element in any of these approaches.

One of the most prevalent alternatives to these architectures is the multi-agent systems technology which can help to distribute resources and reduce the central unit tasks (Ardissono et al., 2004). A distributed agents-based architecture provides more flexible ways to move functions to where actions are needed, thus obtaining better responses at execution time, autonomy, services continuity, and superior levels of flexibility and scalability than centralised architectures (Camarinha-Matos and Afsarmanesh, 2007). Additionally, the programming effort is reduced because the agents cooperate in solving problems and reaching specific goals, thus giving the systems the ability to generate knowledge and experience.

Agent and multi-agent systems combine classical and modern functional architecture aspects. Multi-agent systems are structured by taking into account the modularity in the system, and by reuse, integration and performance. Nevertheless, integration is not always achieved because of the incompatibility among the agents' platforms. The integration and interoperability of agents and multi-agent systems with SOA and Web Services approaches has been recently explored (Ardissono et al., 2004). Some developments are centred on communication between these models, while others are centred on the integration of distributed services, especially Web Services, into the structure of the agents (Li et al., 2004; Shafiq et al., 2006; Liu, 2007; Ricci et al., 2007). Although these developments provide an adequate background for developing distributed multi-agent systems integrating a service-oriented approach, most of them are in early stages of development, so it is not possible to actually know their potential in real scenarios.

3 The FUSION@ architecture

The continuous evolution of software requires creating increasingly complex and flexible applications, so there is a trend towards reusing resources and share compatible platforms or architectures. In some cases, applications require similar functionalities already implemented into other systems which are not always compatible. At this point, developers can face this problem through two options: reuse functionalities already implemented into other systems; or re-deploy the capabilities required, which means more time for development, although this is the easiest and safest option in most cases. While the first option is more adequate in the long run, the second one is most chosen by developers, which leads to have replicated functionalities as well as greater difficulty in migrating systems and applications. This is a poorly scalable and flexible model with reduced response to change, in which applications are designed from the outset as independent software islands.

FUSION@ has been designed to facilitate the development of distributed multi-agent systems with high levels of human-system-environment interaction, since agents have the ability to dynamically adapt their behaviour at execution time. It also provides an advanced flexibility and customisation to easily add, modify or remove applications or services on demand, independently of the programming language. FUSION@ is based on a SOA approach, but modifying this model to fit our requirements and goals. FUSION@ formalises four basic blocks: Applications, Services, Agents Platform and Communication Protocol. These blocks provide all the functionalities of the architecture:

1 *Applications*: These represent all the programs that can be used to exploit the system functionalities. Applications are dynamic and adaptable to context, reacting differently according to the particular situations and the services invoked. They can be executed locally or remotely, even on mobile devices with limited processing capabilities, because computing tasks are largely delegated to the agents and services.

2 *Agents Platform*: This is the core of FUSION@, integrating a set of agents, each one with special characteristics and behaviour. An important feature in this architecture is that the agents act as controllers and administrators for all applications and services, managing the adequate functioning of the system, from services, applications, communication and performance to reasoning and decision-making. In FUSION@, services are managed and coordinated by deliberative BDI agents. The agents modify their behaviour according to the users' preferences, the knowledge acquired from previous interactions, as well as the choices available to respond to a given situation.

3 *Services*: These represent the activities that the architecture offers. They are the bulk of the functionalities of the system at the processing, delivery and information acquisition levels. Services are designed to be invoked locally or remotely. Services can be organised as local services, web services, GRID services, or even as

individual stand alone services. Services can make use of other services to provide the functionalities that users require. FUSION@ has a flexible and scalable directory of services, so they can be invoked, modified, added, or eliminated dynamically and on demand. It is absolutely necessary that all services follow the communication protocol to interact with the rest of the architecture components.

4 *Communication Protocol*: This allows applications and services to communicate directly with the agents' platform. The protocol is completely open and independent of any programming language. This protocol is based on SOAP specification to capture all messages between the platform and the services and applications (Cerami, 2002). Services and applications communicate with the agents' platform via SOAP messages. A response is sent back to the specific service or application that made the request. All external communications follow the same protocol, while the communication among agents in the platform follows the FIPA Agent Communication Language (ACL) specification. This is especially useful when applications run on limited processing capable devices (e.g. cell phones or PDAs). Applications can make use of agents platforms to communicate directly (using FIPA ACL specification) with the agents in FUSION@, so while the communication protocol is not needed in all instances, it is absolutely required for all services. These blocks are managed by means of pre-defined agents that provide the basic functionalities of FUSION@:

- *CommApp Agent*: This agent is responsible for all communications between applications and the platform. It manages the incoming requests from the applications to be processed by services. It also manages responses from services (via the platform) to applications. CommApp Agent is always on 'listening mode'. Applications send XML messages to the agent requesting a service and then the agent creates a new thread to start communication by using sockets. The agent sends all requests to the Manager Agent which processes the request. The socket remains open until a response to the specific request is sent back to the application using another XML message. All messages are sent to Security Agent for their structure and syntax to be analysed.
- *CommServ Agent*: It is responsible for all communications between services and the platform. The functionalities are similar to CommApp Agent but backwards. This agent is always on 'listening mode' waiting for responses of services. Manager Agent signals to CommServ Agent which service must be invoked. Then, CommServ Agent creates a new thread with its respective socket and sends an XML message to the service. The socket remains open until the service sends back a response. All messages are sent to Security Agent for their structure and syntax to be analysed. This agent also periodically checks the status of all services to know if they are idle, busy, or crashed.
- *Directory Agent*: It manages the list of services that can be used by the system. For security reasons (Snidaro and Foresti, 2007), the list of services is static and can only be modified manually; however, services can be added, erased or modified dynamically. The list contains the information of all trusted available services. The name and description of the service, parameters required and the IP address of the computer where the service is running are some of the information stored in the list of services. However, there is dynamic information that is constantly being modified: the service performance (average time to respond to requests), the number of executions and the quality of the service. This last data is very important, as it assigns a value between 0 and 1 to all services. All new services have a Quality of Service (QoS) value set to 1. This value decreases when the service fails (e.g. service crashes, no service found, etc.) or has a subpar performance compared to similar past executions. QoS is increased each time the service efficiently processes the tasks assigned. Information management is especially important on healthcare environments because the data processed is very sensitive and personal. Thus, security must be a major concern when developing this kind of systems. For this reason FUSION@ does not implement a service discovery mechanism, requiring systems to employ only the specified services from a trusted list of services. However, agents can select the most appropriate service (or group of services) to accomplish a specific a task.
- *Supervisor Agent*: This agent supervises the correct functioning of the other agents in the system. Supervisor Agent periodically verifies the status of all agents registered in the architecture by sending ping messages. If there is no response, the Supervisor Agent kills the agent and creates another instance of that agent.
- *Security Agent*: This agent analyses the structure and syntax of all incoming and outgoing XML messages. If a message is not correct, the Security Agent informs the corresponding agent (CommApp or CommServ) that the message cannot be delivered. This agent also directs the problem to the Directory Agent, which modifies the QoS of the service where the message was sent.
- *Manager Agent*: Decides which agent must be called by taking into account the QoS and users preferences. Users can explicitly invoke a service, or can let the Manager Agent decide which service is best to accomplish the requested task. If there are several services that can resolve the task requested by an application, the agent selects the optimal choice. An optimal choice has higher QoS and better performance. Manager Agent has a routing list to manage messages from all applications and services. This agent also checks if services are working properly. It requests the CommServ Agent to send

ping messages to each service on a regular basis. If a service does not respond, CommServ informs Manager Agent, which tries to find an alternate service, and informs the Directory Agent to modify the respective QoS.

- *Interface Agent*: This kind of agent was designed to be embedded in users' applications. Interface agents communicate directly with the agents in FUSION@ so there is no need to employ the communication protocol, rather the FIPA ACL specification. The requests are sent directly to the Security Agent, which analyses the requests and sends them to the Manager Agent. The rest of the process follows the same guidelines for calling any service. These agents must be simple enough to allow them to be executed on mobile devices, such as cell phones or PDAs. All high demand processes must be delegated to services.

3.1 A service example

FUSION@ also facilitates the inclusion of technologies that allow systems to automatically obtain information from users and the environment in an evenly distributed way, focusing on the characteristics of ubiquity, awareness, intelligence, mobility, etc. The goal in FUSION@ is not only to distribute services and applications, but to also promote a new way of developing systems focusing on ubiquity and simplicity. Figure 1 shows the readCHIP service. This service has been implemented to facilitate indoor location based on RFID (Radio Frequency IDentification) technology. When a RFID reader detects the presence of a chip, the readCHIP service is automatically invoked. The inputs considered for this service consists of the device identification, the type of device, and the location of the device. At this moment the service checks the type of CHIP and calculates the location information, that is, the identification for the chip, the user identification and the coordinates which determine the physical position. This information is then sent to the Devices Agent of ALZ-MAS in order to be automatically processed.

Figure 1 Functioning of RFID technology

SERVICE		DESCRIPTION	
readCHIP		This service identificates a chip once it has been detected for a RFID reader	
PROFILE	ClientRole UserAgent	ProviderRole DevicesAgent	
	Inputs	Outputs	
	NoDivice: string typeDevice: string deviceLocation: location	[typeCHIP OK] idCHIP: string idUser: string chipLocation: location	[typeCHIP NOT OK]

3.2 A security mechanism for protecting sensitive data

SOAP-level security mechanisms are very flexible, reason for which, web services may be vulnerable to a great number of attacks based on the unauthorised manipulation, malicious interception and transmission of SOAP messages. These attacks are referred as XML rewriting attacks (Rahaman and Schaad, 2007). In the case of FUSION@ the received messages are executed from many points. This situation supposes a security risk because of the sensitive information transmitted on each message.

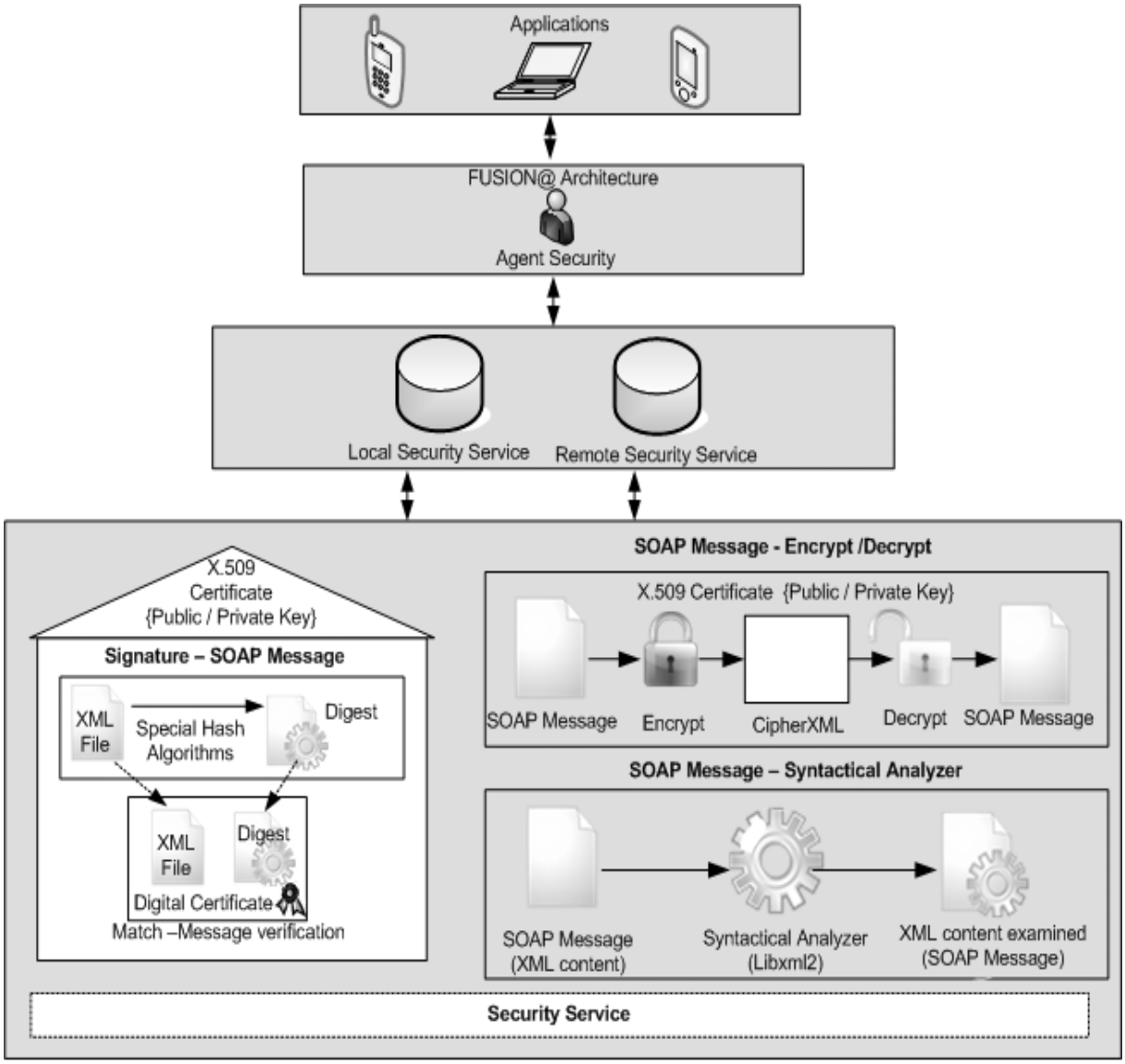
Several standards have been proposed as a solution to tackle attacks on web services, such as WS-Security (OASIS, 2004), WS-Policy (W3C, 2002; W3C, 2008) among other. We have considered to apply a solution set that have been implemented y tested in real scenarios and recommended by the most important institutions in web service security such as W3C (W3C, 2002; W3C, 2008) and OASIS (Advancing Open Standards for the information society) (OASIS, 2004). The main objective is to ensure the typical requirements such as integrity, confidentiality and availability of the information transmitted within the SOAP message.

A security solutions set has been included as a service within FUSION@. The Security Agent in FUSION@ manages the security functions. When a message is received, the Security Agent captures the message and carrying out a call for a security service. The SOAP message is examined and a response is returned about the validity of the SOAP message. The security strategy implemented as a service within FUSION@ is based on XML Digital Signature (W3C, 2008) and XML Encryption (W3C, 2002). XML Digital Signature describes how to attach a digital signature to some XML data. This will ensure data integrity and non repudiation. XML Encryption describes how to encode an XML document or some parts of it, so that its confidentiality can be preserved. This process is implemented by means of XML Security Library (XMLSec Library) (Sanin, 2008). XML Security Library performs signature or encryption by processing input xml or binary data and a template that specifies a signature or encryption skeleton: the transforms, algorithms and the key selection process. XML Security Library supports a variety of features and algorithms. XML security Library is divided in two parts: core library (xmlsec) and crypto library. In this case, the mechanism for the signature and verification of the message is built on by using X.509 certificate. Once the XML security Library has been correctly installed and configured, all the SOAP messages may be validated and signed. Finally, other element incorporated within the security service of FUSION@ is a mechanism to make valid the syntax of each SOAP message. We use Libxml2 (Libxml2-WEB, 2008), which supports the XML 1.0 and contains advanced parser functionality such

as W3C's XML Schema recommendation 1.0 among other. When a SOAP message with bugs is detected, a message is returned by the security service to the Security agent for immediately to block its entrance and execution within the FUSION@'s service database.

Figure 2 shows the security solution set incorporated in FUSION@ for validating each incoming and outgoing SOAP message and avoiding events that can endanger the security in FUSION@. Each incoming SOAP message is verified whether the digital signature has not been changed throughout the path between sender and recipient. Next, sensitive XML data included in the body of the SOAP message are decrypted for finally to analyse its syntactical structure. If the SOAP message is valid then the request of the service is processed. On the other hand, when an outgoing message of response is sent from FUSION@, the first phase is to carry out a syntactical analysis and remove any bug within the message. Next, the sensitive XML data are encrypted and finally the SOAP message is signed for send back the response to the user. By using XML security Library to the digital signature and encryption and Libxml2 as syntactical analyser for the SOAP message, the performance to the architecture is little impacted taking the advantages into account provided by these solutions and theirs close relationship with the SOAP service technology.

Figure 2 Security solution adopted in FUSION@



In the next section, ALZ-MAS 2.0 is presented, where FUSION@ has helped to distribute most of its functionalities and re-design a completely functional multi-agent system aimed at improving several aspects of dependent people.

4 ALZ-MAS 2.0

ALZ-MAS 2.0 is an improved version of ALZ-MAS (ALZheimer Multi-Agent System) (Corchado et al., 2008a; Corchado et al., 2008b), a multi-agent system aimed at enhancing the assistance and healthcare for Alzheimer patients living in geriatric residences. The main functionalities in the system are managed by deliberative BDI agents, including Case-Based Reasoning (CBR) and Case-Based Planning (CBP) mechanisms.

ALZ-MAS structure has five different deliberative agents based on the BDI model (BDI Agents), each one with specific roles and capabilities:

- *User Agent*: This agent manages the users' personal data and behaviour (monitoring, location, daily tasks and anomalies). The User Agent beliefs and goals applied to every user depend on the plan or plans defined by the super-users.
- *SuperUser Agent*: This agent inserts new tasks into the Manager Agent to be processed by a CBR and CBP mechanisms.
- *ScheduleUser Agent*: It is a BDI agent with a CBP mechanism embedded in its structure. It schedules the users' daily activities and obtains dynamic plans depending on the tasks needed for each user. There is one ScheduleUser Agents for each nurse connected to the system.
- *Admin Agent*: It runs on a Workstation and plays two roles: the security role that monitors the users' location and physical building status (temperature, lights, alarms, etc.) through continuous communication with the Devices Agent; and the manager role that handles the databases and the task assignment.
- *Devices Agent*: This agent controls all the hardware devices. It monitors the users' location (continuously obtaining/ updating data from sensors), interacts with sensors and actuators to receive information and control physical services (wireless devices status, communication, temperature, lights, door locks, alarms, etc.).

In the initial version of ALZ-MAS, each agent integrated its own functionalities into their structure. If an agent needs to perform a task which involves another agent, it must communicate with that agent to request it. So, if the agent is disengaged, all its functionalities will be unavailable to the rest of agents. This has been an important issue in ALZ-MAS, since agents running on PDAs are constantly disconnecting from the platform and consequently crashing, making it necessary to restart (killing and launching new instances) those agents. Another important issue is that the CBR and CBP mechanisms are integrated into the agents. These mechanisms are busy almost all the time, overloading the respective agents. Because CBR and CBP mechanisms are the core of the system, they must be available at all times. The system depends on these mechanisms to generate all decisions, so it is essential that they have all processing power available in order to increase overall performance. In addition, the use of CBR and CBP mechanisms into deliberative BDI agents makes these agents complex and unable to be executed on mobile devices. In ALZ-MAS 2.0, these mechanisms have been modelled as services to distribute resources.

The entire ALZ-MAS structure has been modified, separating most of the agents' functionalities from those to be modelled as services. However, all functionalities are the same in both approaches, since we have considered it appropriated to compare the performance of both systems in identical conditions. As an example showing the differences between both approaches, the next sub-section describes the CBP mechanism that has been extracted from the ScheduleUser Agent structure and modelled as a service.

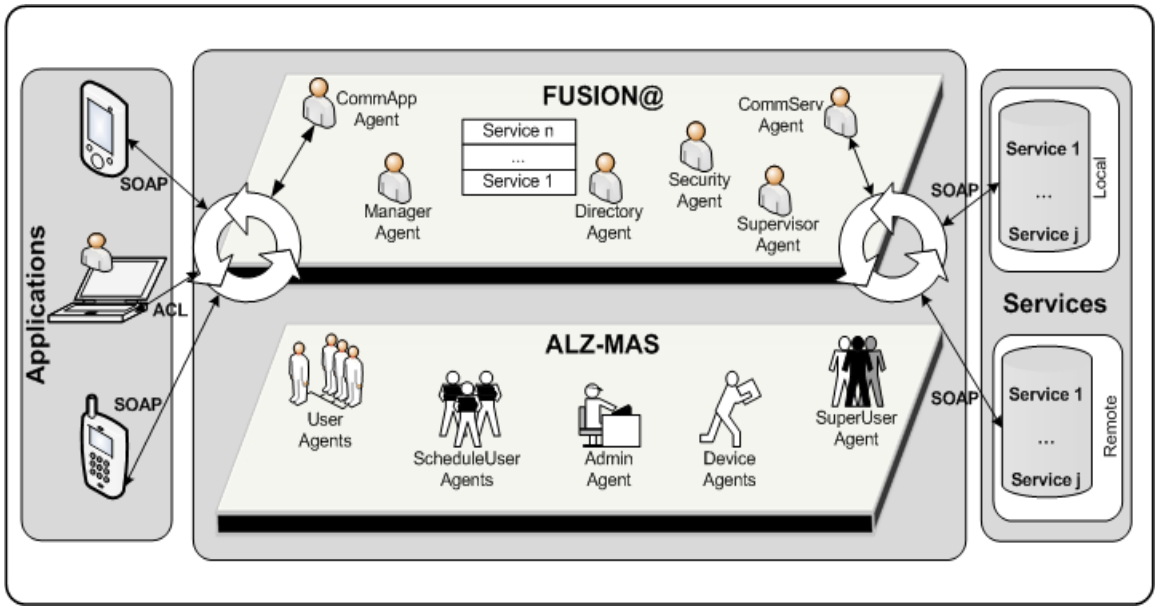
As seen on Figure 3, the entire ALZ-MAS structure has been modified according to FUSION@ model, separating most of the agents' functionalities from those to be modelled as services. However, all functionalities are the same in both approaches, since we have considered it appropriated to compare the performance of both systems to prove the efficiency of FUSION@ model.

4.1 A case-based planning mechanism for scheduling daily activities

As previously mentioned, some agents in ALZ-MAS integrate CBR and CBP mechanisms (then modelled as services in ALZ-MAS 2.0), which allow them to make use of past experiences to create better plans and achieve their goals. These mechanisms provide the agents greater learning and adaptation capabilities. The main characteristics of the CBP mechanism are described in the remainder of this section.

Case-Based Reasoning (CBR) is a type of reasoning based on past experiences (Aamodt and Plaza, 1994). CBR solve new problems by adapting solutions that have been used to solve similar problems in the past, and learn from each new experience. The primary concept when working with CBR is the concept of case, which is described as a past experience composed of three elements: an initial state or problem description that is represented as a belief; a solution, which provides the sequence of actions carried out in order to solve the problem; and a final state, which is represented as a set of goals. CBR manages cases (past experiences) to solve new problems. The way cases are managed is known as the CBR cycle, and consists of four sequential phases: retrieve, reuse, revise and retain. The retrieve phase starts when a new problem description is received. Similarity algorithms are applied so that the cases with the problem description most similar to the current one can be retrieved from the cases memory. Once the most similar cases have been retrieved, the reuse phase begins by adapting the solutions for the retrieved cases in order to obtain the best solution for the current case. The revise phase consists of an expert revision of the proposed solution. Finally, the retain phase allows the system to learn from the experiences obtained in the three previous phases, and consequently updates the cases memory.

Figure 3 ALZ-MAS 2.0 basic structure (see online version for colours)



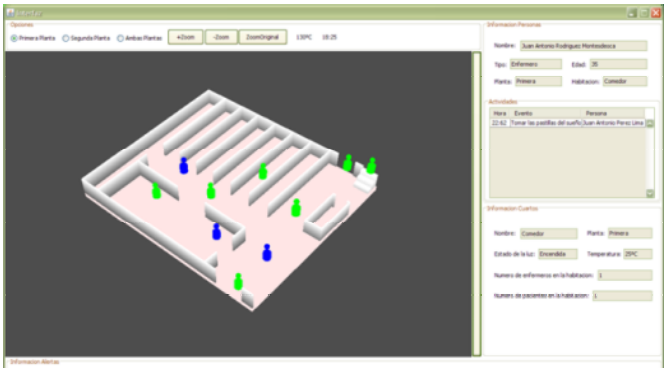
CBP comes from CBR, but is specially designed to generate plans (sequence of actions) (Corchado et al., 2008a; Corchado et al., 2008b). In CBP, the proposed solution for solving a given problem is a plan. This solution is generated by taking into account the plans applied for solving similar problems in the past. The problems and their corresponding plans are stored in a plans memory. The reasoning mechanism generates plans using past experiences and planning strategies, which is how the concept of Case-Based Planning is obtained (Corchado et al., 2008b; De Paz et al., 2008). CBP consists of four sequential stages: the retrieve stage, which recovers the past experiences most similar to the current one; the reuse stage, which combines the retrieved solutions in order to obtain a new optimal solution; the revise stage, which evaluates the obtained solution; and retain stage, which learns from the new experience. Problem description (initial state) and solution (situation when final state is achieved) are represented as beliefs, the final state as a goal (or set of goals), and the sequences of actions as plans. The CBP cycle is implemented through goals and plans. When the goal corresponding to one of the stages is triggered, different plans (algorithms) can be executed concurrently to achieve the goal or objective. Each plan can trigger new sub-goals and, consequently, cause the execution of new plans. In practice, what is stored is not only a specific problem with a specific solution, but also additional information about how the plans have been derived. As with CBR, the case representation, the plans memory organisation, and the algorithms used in every stage of the CBP cycle are essential in defining an efficient planner.

In the initial version of ALZ-MAS, the CBR and CBP mechanisms are deeply integrated into the agents' structure. In ALZ-MAS 2.0, these mechanisms have been modelled as services linked to agents, thus increasing the system's overall performance. To generate a new plan, a ScheduleUser Agent (running on a PDA) sends a request to the platform. The message is processed and the platform invokes the mechanism (or service). The mechanism receives the message and starts to generate a new plan. Then, the solution is sent to the platform which delivers the new plan to all ScheduleUser Agents running. The CBP service creates optimal paths and scheduling in order to facilitate the completion of all tasks defined for the nurses connected to the system (Corchado et al., 2008b).

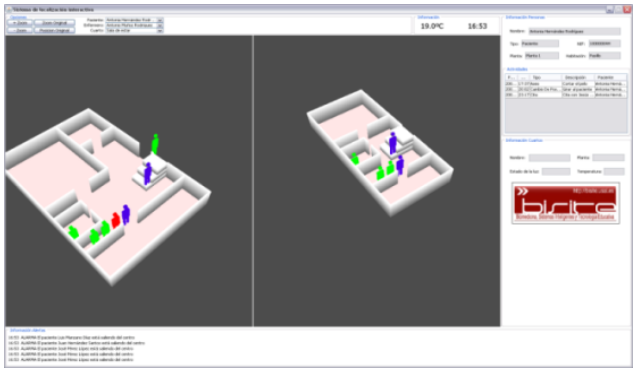
5 Results and conclusions

This paper has presented FUSION@, an architecture which proposes a novel approach for integrating: applications, agents and services. FUSION@ also facilitates the inclusion of technologies that allow systems to automatically obtain information from users and the environment in an evenly distributed way. FUSION@ has been employed to develop an improved version of ALZ-MAS (ALZheimer Multi-Agent System) (Corchado et al., 2008a), a multi-agent system aimed at enhancing assistance and healthcare for Alzheimer patients in geriatric residences. Figure 4 shows the main user interface of ALZ-MAS (left) and ALZ-MAS 2.0. These systems have the same functionalities and share almost the same user interface. The interfaces show basic information about nurses and patients (name, tasks that must be accomplished, schedule, location inside the residence, etc.) and the building (outside temperature, specific room temperature, lights status, etc.). Both interfaces are managed by the Manager Agent and appear similar to users. However, the performance of ALZ-MAS 2.0 has been highly improved, mainly because most of the functionalities have been modelled as distributed and independent services.

Figure 4 (a) ALZ-MAS main user interface; (b) ALZ-MAS 2.0 main user interface (see online version for colours)



(a)



(b)

Several tests have been done to demonstrate if a distributed approach is appropriate to optimise the performance of multi-agent systems, in this case ALZ-MAS 2.0. The tests consisted of a set of requests delivered to the CBP mechanism which in turn had to generate paths for each set of tasks (i.e. scheduling). For every new test, the cases memory of the CBP mechanism was deleted in order to avoid a learning capability, thus requiring the mechanism to accomplish the entire planning process. As can be seen in Table 1, a task is a Java object that contains a set of parameters. ScheduleTime is the time in which a specific task must be accomplished, although the priority level of other tasks needing to be accomplished at the same time is factored in. The CBP mechanism increases or decreases ScheduleTime and MaxTime according to the priority of the task: ScheduleTime = ScheduleTime-5min*TaskPriority and MaxTime = MaxTime+5min*TaskPriority. Once these times have been calculated, the path is generated taking the RoomCoordinates into account. There were 30 defined agendas each with 50 tasks. Tasks had different priorities and orders on each agenda. Tests were carried out on seven different test groups, with 1, 5, 10, 15, 20, 25 and 30 simultaneous agendas to be processed by the CBP mechanism. 50 runs for each test group were performed, all of them on machines with equal characteristics. Several data have been obtained from these tests, notably the average time to accomplish the plans, the number of crashed agents, and the number of crashed services. For ALZ-MAS 2.0, five CBP services with exactly the same characteristics were replicated.

Figure 5 shows the average time needed by both systems to generate the paths for a fixed number of simultaneous agendas. The previous version of ALZ-MAS was unable to handle 15 simultaneous agendas and time increases to infinite because it was impossible to perform those requests. However, ALZ-MAS 2.0 had five replicated services available, so the workflow was distributed and allowed the system to complete the plans for 30 simultaneous agendas. Another important data is that although the previous version of ALZ-MAS performed slightly faster when processing a single agenda, performance was constantly reduced when new simultaneous agendas were added. This fact demonstrates that the overall performance of ALZ-MAS 2.0 is better when handling distributed and simultaneous tasks (e.g. agendas), instead of single tasks.

Table 1 Tasks description

Task	*Data*
TaskId	36
TaskType	32
TaskDescript	Description
TaskPriority	3
TaskObjective	0
TaskIncidents	0
UserId	7
UserNecessities	2
MinTime	10 min
MaxTime	60 min
ScheduleTime	12:00
RoomCoordinates	(1, 3)
TaskResources	2,4,8

Figure 5 Time needed for both systems to generate paths for a group of simultaneous agendas

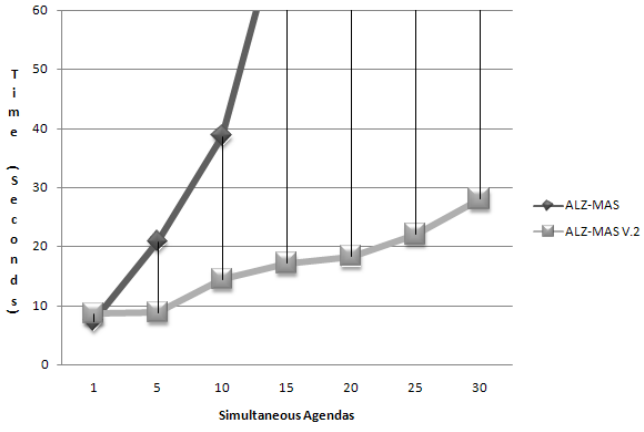
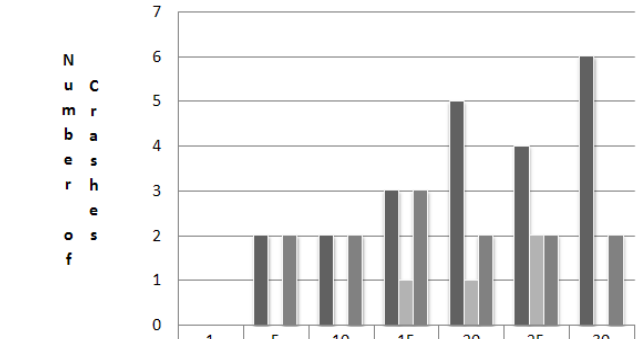


Figure 6 shows the number of crashed agents for both versions of ALZ-MAS during tests. None of the tests where agents or services crashed were taken into account to calculate the data presented in Figure 6, so these tests were repeated. As can be seen, the previous version of ALZ-MAS is far more unstable than ALZ-MAS 2.0. These data demonstrate that this approach provides a higher ability to recover from errors.

Figure 6 Number of agents crashed



	1	5	10	15	20	25	30
ALZ-MAS Agents	0	2	2	3	5	4	6
ALZ-MAS V.2 Agents	0	0	0	1	1	2	0
ALZ-MAS V.2 Services	0	2	2	3	2	2	2

Although these tests have provided us with very useful data, it is necessary to continue experimenting with FUSION@. A SOA approach is an efficient way to distribute resources and develop more robust multi-agent systems, especially when handling complex mechanisms as the CBP presented.

Moreover, another important aspect in FUSION@ is the security issue. Frequently, the security is not considered as an important issue in the first stages of the software development. This situation endangers the use and the quality of the product, in this case the software application. FUSION@ incorporates a security mechanism based on XML Digital Signature and XML Encryption for analysing the integrity of the syntactical structure of each incoming and outgoing SOAP message.

We are currently exploring alternative case studies for applying this architecture and demonstrate that the service-oriented approach presented is flexible enough to be implemented in other scenarios. One main issue to be taken into account is that the architecture is still under development so it is necessary to define it by means of Agent-Oriented Software Engineering (AOSE) tools (Chan and Sterling, 2003) such as INGENIAS (Pavón et al., 2005), MESSAGE (Caire et al., 2002), GAIA (Wooldridge et al., 2000) or MaSE (DeLoach, 2001) among others.

Acknowledgements

This work has been supported by the IMSERSO 137/2007, the UPSA U05E1A-07L01 and the MCYT TIN2006-14630-C03-03 projects.

References

Aamodt, A. and Plaza, E. (1994) 'Case-based reasoning: foundational issues, methodological variations, and system approaches', *AI Communications*, Vol. 7, pp.39–59.

Ardissono, L., Petrone, G. and Segnan, M. (2004) 'A conversational approach to the interaction with Web Services', *Computational Intelligence*, Vol. 20, pp.693–709.

Bratman, M.E. (1987) *Intentions, Plans and Practical Reason*, Harvard University Press, Cambridge, MA.

Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J.J., Pavon, J., Kerney, P., Stark, J. and Massonet, P. (2002) *Eurescom P907: MESSAGE – Methodology for Engineering Systems of Software Agents*, Available online at: http://www.eurescom.de/public/projects/P900-series/p907/default.asp

Camarinha-Matos, L.M. and Afsarmanesh, H. (2007) 'A comprehensive modeling framework for collaborative networked organizations', *Journal of Intelligent Manufacturing*, Vol. 18, No. 5, pp.529–542.

Cerami, E. (2002) *Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*, 1st ed., O'Reilly & Associates, Inc.

Chan, K. and Sterling, L. (2003) 'Specifying roles within agent-oriented software engineering', *Proceedings of the 10th Asia-Pacific Software Engineering Conference*, IEEE, pp.390–395.

Corchado, J.M., Bajo, J. and Abraham, A. (2008a) 'GERAmI: improving the delivery of health care', *IEEE Intelligent Systems*, Vol. 23, No. 2, pp.19–25.

Corchado, J.M., Bajo, J., De Paz, Y. and Tapia, D.I. (2008b) 'Intelligent environment for monitoring Alzheimer Patients, agent technology for health care', *Decision Support Systems*, Vol. 44, No. 2, pp.382–396.

DeLoach, S. (2001) 'Analysis and design using MaSE and agentTool', *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS)*.

De Paz, J.F., Rodríguez, S., Bajo, J. and Corchado, J.M. (2008) 'Dynamic case based planning', *Proceedings of the 8th International Conference on Computational and Mathematical Methods in Science and Engineering*, La Manga del Mar Menor, Spain, Vol. 1, pp.213–224.

Jayaputera, G.T., Zaslavsky, A.B. and Loke, S.W. (2007) 'Enabling run-time composition and support for heterogeneous pervasive multi-agent systems', *Journal of Systems and Software*, Vol. 80, No. 12, pp.2039–2062.

Li, Y., Shen, W. and Ghenniwa, H. (2004) 'Agent-based web services framework and development environment', *Computational Intelligence*, Vol. 20, No. 4, pp.678–692.

Libxml2-WEB (2008) *The XML C parser and toolkit of Gnome*. Available online at: http://xmlsoft.org/index.html

Liu, X. (2007) 'A multi-agent-based service-oriented architecture for inter-enterprise cooperation system', *Proceedings of the 2nd International Conference on Digital Telecommunications*, IEEE Computer Society, Washington, DC.

OASIS (2004) 'Advancing open standards for the information society', *Web Services Security: SOAP Message Security 1.0*.

OASIS (2008) *Reference Architecture for Service Oriented Architecture Version 1.0*, Public Review Draft 1.

Oren, E., Haller, A., Mesnage, C., Hauswirth, M., Heitmann, B. and Decker, S. (2007) 'A flexible integration framework for Semantic Web 2.0 applications', *IEEE Software*, Vol. 24, No. 5, pp.64–71.

Pavón, J., Gómez-Sanz, J.J. and Fuentes-Fernandez, R. (2005) *The INGENIAS Methodology and Tools*, Idea Group Publishing, pp.236–276.

Rahaman, M.A. and Schaad, A. (2007) 'SOAP-based secure conversation and collaboration', Web Services, 2007, ICWS 2007, IEEE International Conference, pp.471–480.

Ricci, A., Buda, C. and Zaghini, N. (2007) 'An agent-oriented programming model for SOA & web services', *Proceedings of the 5th IEEE International Conference on Industrial Informatics*, Vienna, Austria, pp.1059–1064.

Sanin, A. (2008) *XML Security Library Reference Manual*. Available online at: http://www.aleksey.com/xmlsec/api/index.html

Shafiq, M.O., Ding, Y. and Fensel, D. (2006) 'Bridging multi-agent systems and web services: towards interoperability between software agents and semantic web services', *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, IEEE Computer Society, Washington, DC, pp.85–96.

Snidaro, L. and Foresti, G.L. (2007) 'Knowledge representation for ambient security', *Expert Systems*, Vol. 24, No. 5, pp.321–333.

Wooldridge, M. and Jennings, N.R. (1995) 'Intelligent agents: theory and practice', *The Knowledge Engineering Review*, Vol. 10, No. 2, pp.115–152.

Wooldridge, M., Jennings, N.R. and Kinny, D. (2000) 'The Gaia methodology for agent-oriented analysis and design, autonomous agents and multi-agent systems', Vol. 3, No. 3, pp.285–312.

W3C (2002) *Web XML encryption syntax and processing*. Available online at: http://www.w3.org/TR/xmlenc-core/

W3C (2008) XML signature syntax and processing. Available online at: http://www.w3.org/TR/xmlenc-core/