International Journal of Innovative Computing, Information and Control
Volume **8**, Number **4**, April **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 2453–2476

A MULTI-AGENT ARCHITECTURE FOR DISTRIBUTED SERVICES AND APPLICATIONS

Juan M. Corchado[1], Dante I. Tapia[1] and Javier Bajo[2]

[1]Departament of Computer Science
University of Salamanca
Plaza de la Merceds/n, 37008, Salamanca, Spain
{ corchado; dantetapia }@usal.es

[2]Universidad Pontificia de Salamanca
Compañía 5, 37002, Salamanca, Spain
jbajope@upsa.es

Received December 2010; revised June 2011

Abstract. *Ambient Intelligence has acquired great importance in recent years and requires the development of new innovative solutions. This paper presents a novel architecture which facilitates the integration of multi-agent systems, distributed services and applications to optimize the construction of Ambient Intelligence environments. The architecture proposes a new and easier method to develop distributed intelligent ubiquitous systems, where applications and services can communicate in a distributed way with intelligent agents, even from mobile devices, independent of location restrictions. The core of the architecture is a group of deliberative agents acting as controllers and administrators for all applications and services. This approach provides the systems with a higher ability to recover from errors and a better flexibility to change their behavior at execution time. The architecture is founded on the Ambient Intelligence paradigm. A distributed multi-agent system has been developed to test this architecture. This system is aimed to improve health care and assistance to dependent persons in geriatric residences, and the preliminary results are presented in this paper.*
Keywords: Ambient intelligence, Multi-agent architecture, Distributed services, Services oriented architectures, Case-based reasoning, Case-based planning

1. **Introduction.** People are currently surrounded by technology which tries to increase our quality of life and facilitate our daily activities. However, there are situations where technology is difficult to handle or people lack knowledge to use it. For these reasons, Ambient Intelligence tries to adapt the technology to the people's needs by proposing three basic concepts: ubiquitous computing, ubiquitous communication and intelligent user interfaces [55]. To reach this objective, it is necessary to develop new functional architectures capable of providing adaptable and compatible frameworks, allowing access to services and applications regardless of location restrictions. A functional architecture defines the physical and logical structure of the components that make up a system, as well as the interactions between those components [19]. There are SOA [11, 18] and agents' frameworks and platforms which provide tools for developing distributed systems and multi-agent systems [23, 43, 48]. However, these tools do not solve by themselves the Ambient Intelligence based systems needs. For this reason, it is necessary to develop innovative solutions that integrate different approaches in order to create flexible and adaptable systems, especially for achieving higher levels of interaction with people in a ubiquitous and intelligent way.

This paper describes a Flexible User and ServIces Oriented multi-ageNt Architecture (FUSION@) and explains how this architecture has been designed and applied to a real case scenario. FUSION@ is a novel architecture which integrates a SOA approach with intelligent agents for building systems based on the Ambient Intelligence paradigm. The architecture focuses on distributing the majority of the systems' functionalities into remote and local services and applications. It also 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, but rather they are modeled as distributed services and applications which are invoked by the agents acting as controllers and coordinators. This model provides a flexible distribution of resources and facilitates the inclusion of new functionalities in highly dynamic environments.

FUSION@ is based on agents due to their characteristics, such as autonomy, reasoning, reactivity, social abilities, pro-activity, mobility and organization, which allow them to cover several needs for Ambient Intelligence environments, especially ubiquitous communication and computing and adaptable interfaces. Multi-agent systems have been successfully applied to several Ambient Intelligence scenarios, such as education, culture, entertainment, medicine and robotics [5, 13, 16, 36, 44, 55]. The characteristics of the agents make them appropriate for developing dynamic and distributed systems based on Ambient Intelligence, as they possess the capability of adapting themselves to the users and environmental characteristics [24]. The continuous advancement in mobile computing makes it possible to obtain information about the context and also to react physically to it in more innovative ways [24]. To facilitate ubiquitous computation capabilities, the agents in the FUSION@ architecture are based on the deliberative Belief, Desire, Intention (BDI) model [39, 45, 56], where the agents' internal structure and capabilities are based on mental aptitudes, using beliefs, desires and intentions [45]. Nevertheless, Ambient Intelligence developments need higher adaptation, learning and autonomy levels than pure BDI model [39], since it is necessary to incorporate memory management and reasoning mechanisms. This is achieved in FUSION@ by modeling the agents' characteristics [56] to provide them with innovative mechanisms that allow solving complex problems and autonomous learning. Some of these mechanisms are Case-Based Reasoning (CBR) (Montani and Jain 2010) and Case-Based Planning (CBP), where problems are solved by using solutions to similar past problems [13]. 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 modeled as external services. Deliberative agents use these services to learn from past experiences and to adapt their behavior according to the context. One of the advantages of these intelligent reasoning mechanisms is the effective distribution of computational tasks into services and applications.

FUSION@ was initially implemented to develop a distributed multi-agent system aimed at providing assistance to dependent people, such as the elderly and those with disabilities, but can be easily adapted to other kinds of problems or environments. Dependence is a permanent situation in which a person needs important assistance from others in order to perform basic daily life activities such as essential mobility, object and people recognition and domestic tasks [14]. There is an ever growing need to supply constant care and support to the disabled and elderly, and the drive to find more effective ways of providing such care has become a major challenge for the scientific community [35]. The World Health Organization has determined that in the year 2025 there will be 1 billion people in the world over the age of 60 and twice as many by 2050, with nearly 80% concentrated in developed countries [57]. Spain will be the third "oldest country" in the world, just behind Japan and Korea, with 35% of its citizens over 65 years of age [42]. In fact, people over 60 years old represent more than 21% of the European population [57], and

people over 65 are the fastest growing segment of the population in the United States of America [3]. Furthermore, over 20% of those people over 85 have a limited capacity for independent living, requiring continuous monitoring and daily assistance [17]. The importance of developing new and more reliable ways of providing care and support for the elderly is underscored by this trend, and the creation of secure, unobtrusive and adaptable environments for monitoring and optimizing health care will become vital. Some authors [35] consider that tomorrow's health care institutions will be equipped with intelligent systems capable of interacting with humans. Multi-agent systems and architectures based on intelligent devices have recently been explored as supervision systems for medical care for dependent people. These intelligent systems aim to support patients in all aspects of daily life, predicting potential hazardous situations and delivering physical and cognitive support.

FUSION@ combines multi-agent systems, services and applications to obtain an innovative architecture that presents important improvements in the area of Ambient Intelligence (AmI) [1, 55], facilitating ubiquitous computation and communication and high levels of human-system-environment interaction. It also provides an advanced flexibility and customization to easily add, modify or remove applications or services on demand, independently of the programming language. FUSION@ formalizes the integration of applications, services, communications and agents. It also facilitates the inclusion of context-aware technologies, such as radiofrequency location and identification, 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., all of which are concepts defined by Ambient Intelligence. Wireless networks are one of the top technologies that have been used to test this architecture in several systems. These networks provide an infrastructure capable of supporting the distributed communication needed for agents, applications and services, and of increasing mobility, flexibility and efficiency since resources can be accessed no matter their physical location [31]. The goal in FUSION@ is not only to distribute services and applications, but to also promote a new way of developing AmI-based systems focusing on ubiquity and simplicity. Given that developing novel solutions for dependent people is one of the priorities of AmI, FUSION@ has been used to construct an intelligent environment for dependent people, and compared with a previous existing solution.

In the next section, the specific problem description that essentially motivated the development of FUSION@ is presented. Section 3 describes the main characteristics of this architecture and briefly explains some of its components. Section 4 presents a case study consisting of a distributed multi-agent system for health care scenarios developed using this architecture. Finally, Section 5 presents the results and conclusion obtained.

2. Multi-Agent and Services Oriented Architectures.

The emergence of Ambient Intelligence [1] involves substantial changes in the design of functional architectures, since it is necessary to provide features which enable a ubiquitous computing and communication and also an intelligent interaction with users. This section discusses some of the most important problems of existent functional architectures, including their suitability for constructing intelligent environments according the AmI paradigm. This section also presents the strengths and weaknesses of existent platforms and analyzes the feasibility of a new alternative: FUSION@.

Excessive centralization of services negatively affects the systems' functionalities, overcharging or limiting their capabilities. Classical functional architectures are characterized by trying to find modularity and a structure oriented to the system itself. Modern functional architectures like Service-Oriented Architecture (SOA) [18] 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 [11, 50]. Services are linked by means of standard communication protocols that must be used by applications in order to share resources in the services network [4, 37]. 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 multi-agent systems which can help to distribute resources and reduce the central unit tasks [4, 53, 56]. There are several agents' frameworks and platforms, such as Open Agent Architecture (OAA) [12, 23, 30], RETSINA [20, 43, 49], JADE [6, 48] which provide a wide range of tools for developing distributed multi-agent systems. FUSION@ sets on top of these architectures by adding new layers for integrating a SOA approach and facilitating the distribution and management of resources (i.e., services). 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 centralized architectures [10]. Additionally, the programming effort is reduced because it is only necessary to specify global objectives so that agents cooperate in solving problems and reaching specific goals, thus giving the systems the ability to generate knowledge and experience. Moreover, the system control is reduced because the agents need more autonomy to solve complex problems. Unfortunately, the difficulty in developing a multi-agent architecture is higher and because there are no specialized programming tools to develop agents, the programmer needs to type a lot of code to create services and clients [41]. Furthermore, it is also necessary to have a more complex system analysis and design, which implies more time to reach the implementation stage. The development of agents is an essential piece in the analysis of data from distributed sensors and gives those sensors the ability to work together and analyze complex situations, thus achieving high levels of interaction with humans [38].

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 [4, 8, 51]. Some developments are centered on communication between these models, while others are centered on the integration of distributed services, especially Web Services, into the structure of the agents. Bonino da Silva et al. [7] propose merging multi-agent techniques with semantic web services to enable dynamic, context-aware service composition. They focus on SOA in designing a multi-agent service composition as an intelligent control layer, where agents discover services and adapt their behavior and capabilities according to semantic service descriptions. Ricci et al. [40] have developed a java-based framework to create SOA and Web Services compliant applications, which are modeled as agents. Communication between agents and services is performed by using what they call "artifacts" and WSDL (Web Service Definition Language) [11]. Shafiq et al. [46] propose a gateway that allows interoperability between Web Services and multi-agent systems. This gateway is an agent that integrates Foundation for Intelligent Physical Agents (FIPA) and The World Wide Web Consortium (W3C) specifications, translating Agent Communication Language (ACL), SOAP [11] and WSDL messages, and combines both directories from agents' platforms and web services. Li et al. [28] propose a similar approach, but focus on the representation of services. They use SOAP and WSDL messages to interact with agents. Liu [29] proposes

a multi-agent architecture to develop inter-enterprise cooperation systems using SOA and Web Services components and communication protocols. Walton [54] presents a technique to build multi-agent systems using Web Services, defining a language to represent the dialogs among agents. There are also frameworks, such as Sun's Jini and IBM's WebSphere, which provide several tools to develop SOA-based systems. Jini uses Java technology to develop distributed and adaptive systems over dynamic environments [22]. Rigole et al. [41] have used Jini to create agents on demand into a home automation system, where each agent is defined as a service in the network. WebSphere provides tools for several operating systems and programming languages [2]. However, the systems developed using these frameworks are not open at all because the framework is closed and services and applications must be programmed using a specific programming language that supports their respective proprietary APIs.

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. In addition, FUSION@ not only provides communication and integration between distributed agents, services and applications; but also proposes a new method to facilitate the development of distributed multi-agent systems by means of modeling the functionalities of the agents and the systems as services. Another feature in this architecture is security, which is managed by the agents. All communications must take place via the agents, so services cannot share their resources unless the agents allow it. Besides, services defined for each system must always be available, so they are not shared with other systems unless it is specified. The FUSION@ approach is presented in detail in the following section.

3. FUSION@: A Flexible User and Services Oriented Multi-ageNt Architecture.

FUSION@ is a novel architecture which facilitates the integration of multi-agent systems, distributed services and applications. The model of FUSION@ has been mainly designed to develop Ambient Intelligence based systems. These systems must be dynamic, flexible, robust, adaptable to changes in context, scalable and easy to use and maintain. However, the architecture can be used to develop any kind of complex system because it is capable of integrating almost any service and application desired, with no dependency on any specific programming language. Because the architecture acts as an interpreter, the users can run applications and services programmed in virtually any language, but have to follow a communication protocol that all applications and services must incorporate. Another important functionality is that, thanks to the agents' capabilities, the systems developed can make use of reasoning mechanisms or learning techniques to handle services and applications according to context characteristics, which can change dynamically over time. Agents, applications and services can communicate in a distributed way, even from mobile devices. This makes it possible to use resources no matter its location. It also allows the starting or stopping of agents, applications, services or devices separately, without affecting the rest of resources, so the system has an elevated adaptability and capacity for error recovery.

FUSION@ proposes a new perspective, where multi-agent systems and SOA services are integrated to provide ubiquitous computation, ubiquitous communication and intelligent interfaces facilities. Multi-agent architectures as Open Agent Architecture (OAA) [12], RETSINA [49], JADE [48], define agent-based structures to resolve distributed computational problems and facilitate user interactions. However, the communications capabilities are limited, not allowing easy integration with services and applications. On the other hand, the SOA-based architectures usually do not provide intelligent computational and

interactive mechanisms [50]. FUSION@ combines both paradigms, trying to take advantage of their strengths and avoid their weakness. Next, the structure of FUSION@ is described.

As can be seen in Figure 1, FUSION@ framework defines four basic blocks: Applications, Services, Agents Platform and Communication Protocol. This framework has been modeled following the SOA model, but adding the applications block which represents a fundamental part in Ambient Intelligence: the interaction with users. These blocks provide all the functionalities of the architecture:

- **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.
- **Agents Platform**. This is the core of FUSION@, integrating a set of agents, each one with special characteristics and behavior. 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 with distributed computation and coordination abilities. These BDI agents are implemented using the Jadex framework [39]. The agents modify their behavior according to the users' preferences, the knowledge acquired from previous interactions, as well as the choices available to respond to a given situation.
- **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 in a ubiquitous way. Services are designed to be invoked locally or remotely. Services can be organized 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.
- **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, facilitating ubiquitous communication capabilities. This protocol is based on SOAP specification to capture all messages between the platform and the services and applications [11, 18]. 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.

One of the advantages of FUSION@ is that the users can access the system through distributed applications, which run on different types of devices and interfaces (e.g., computers, cell phones, PDA). Figure 2 shows the basic schema of FUSION@, where all

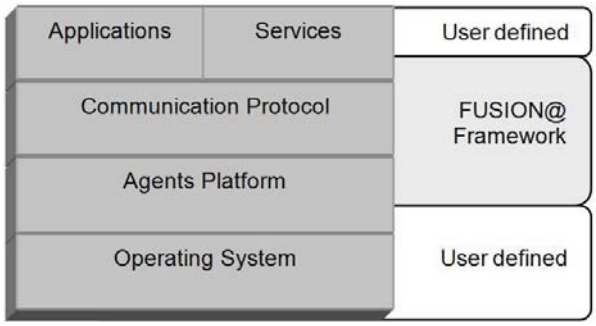

FIGURE 1. FUSION@ framework

requests and responses are handled by the agents in the platform. The agents analyze all requests and invoke the specified services either locally or remotely. Services process the requests and execute the specified tasks. Then, services send back a response with the result of the specific task.

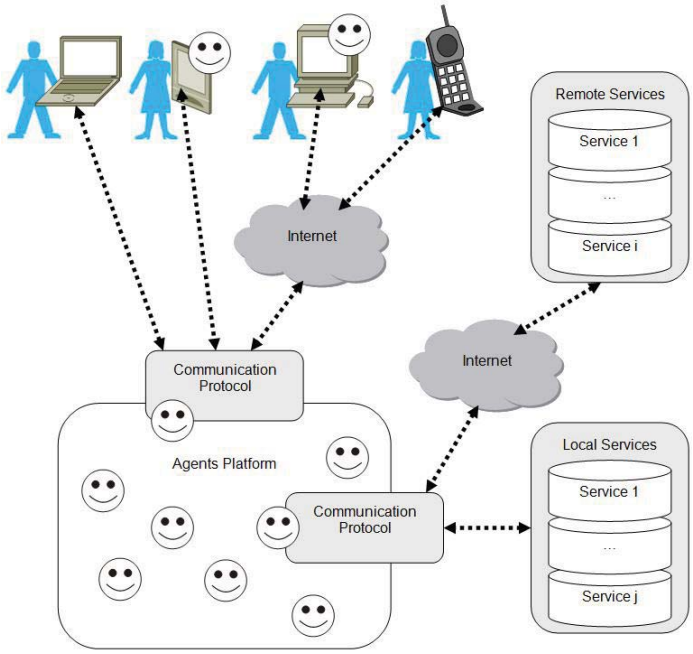

FIGURE 2. FUSION@ basic schema

A simple case illustrated in Figure 3 can better demonstrate the basic functioning of FUSION@ when it is requesting a service. *A user needs to calculate the sum of two numbers and wants to do it through a PDA remotely connected to the system. The user executes a mathematical toolkit which provides him or her with a large set of formulas from which he or she selects the sum function, introduces a set of values, and clicks a button to get the result. When the user clicks the button, the application sends a request to the platform to find a service that can process that request. The agents invoke the appropriate service and send the request. The service processes the request and sends the result back to the agents which in turn send it to the application.* It is obvious that invoking a remote service to execute a sum is not the best choice. However, imagine a large scale process that uses complex AI (Artificial Intelligence) techniques, such as genetic algorithms, data mining and neural networks. where the limited processing capacity of the PDA makes it impossible to calculate. In this case, the service may be in a powerful computer and

could be remotely invoked by the PDA. A real example of this situation is described in Section 4.1.

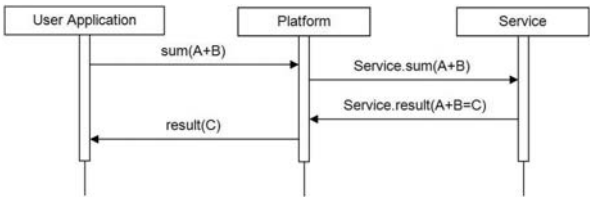

FIGURE 3. Basic behavior when requesting a service by means of FUSION@

As can be seen in Figure 4, the Web Services Architecture model uses an external directory, known as UDDI (Universal Description, Discovery and Integration) [11] to list all available services. Each service must send a WSDL file to the UDDI to be added to the directory. Applications consult the UDDI to find a specific service. Once the service is located, the application can start communication directly with the selected service. However, FUSION@ does not include a service discovery mechanism, so applications must use only the services listed in the platform. In addition, all communication is handled by the platform, so there is no way to interact directly between applications and services. Moreover, the platform makes use of deliberative agents to select the optimal option to perform a task, so users do not need to find and specify the service to be invoked by the application. These features have been introduced into FUSION@ to create a secure communication between applications and services. They also facilitate the inclusion of new services that users can make use regarding their location and application.

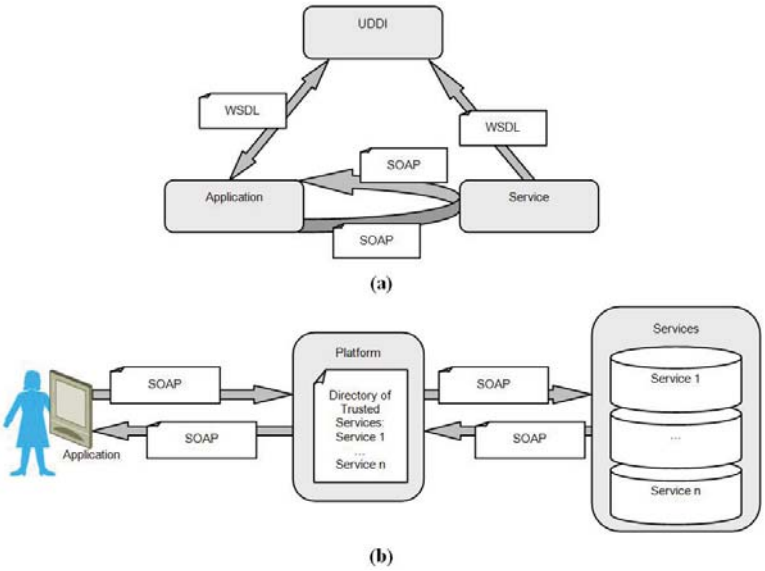

FIGURE 4. (a) Web services architecture; (b) FUSION@s

FUSION@ is a modular multi-agent architecture, where services and applications are managed and controlled by deliberative BDI (Belief, Desire, Intention) agents [39, 45, 56]. Deliberative BDI agents are able to cooperate, propose solutions on very dynamic environments, and face real problems, even when they have a limited description of the problem and few resources available. These agents depend on beliefs, desires, intentions and plan representations to solve problems [39]. Deliberative BDI agents are the core

of FUSION@. There are different kinds of agents in the architecture, each one with specific roles, capabilities and characteristics. This fact facilitates the flexibility of the architecture in incorporating new agents. However, there are pre-defined agents which provide the basic functionalities of the architecture:

- **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, then the agent creates a new thread to start communication by using sockets. The agent sends all requests to the Admin 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 analyzed.
- **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. Admin 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 analyzed. 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 [47], 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 and no service found) 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 Ambient Intelligence environments because the data processed is very sensitive and personal. Thus, security must be a major concern when developing AmI-based 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 analyzes 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.

- **Admin 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 Admin 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. Admin 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 Admin 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 into 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 analyzes the requests and sends them to the Admin 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. In fact, we have implemented this philosophy in the system described in Section 4.

FUSION@ is an open architecture that allows developers to modify the structure of the agents described before, so that agents are not defined in a static manner. Developers can add new agent types or extend the existing ones to conform to their projects needs. However, most of the agents' functionalities should be modeled as services, releasing them from tasks that could be performed by services. Services represent all functionalities that the architecture offers to users and uses itself. As previously mentioned, services can be invoked locally or remotely. All information related to services is stored into a directory which the platform uses in order to invoke them, i.e., the services. This directory is flexible and adaptable, so services can be modified, added or eliminated dynamically. Services are always on "listening mode" to receive any request from the platform. It is necessary to establish a permanent connection with the platform using sockets. Every service must have a permanent listening port open in order to receive requests from the platform. Services are requested by users through applications, but all requests are managed by the platform, not directly by applications. This provides more control and security when requesting a service because the agents can control and validate all messages sent to the services and applications. When the platform requests a service, the CommServ Agent sends an XML message to the specific service. The message is received by the service and creates a new thread to perform the task. The new thread has an associated socket which maintains communication open to the platform until the task is finished and the result is sent back to the platform. This method provides services the capability of managing multiple and simultaneous tasks, so services must be programmed to allow multi-threading. However, there could be situations where multi-tasks will not be permitted, for instance high demanding processes where multiple executions could significantly reduce the services performance. In these cases, the Admin Agent asks the CommServ Agent to consult the status of the service, which informs the platform that it is busy and cannot accept other requests until finished. The platform must then seek another service that can handle the request, or wait for the service to be idle. To add a new service, it is necessary to manually store its information into the directory list managed

by the Directory Agent. Then, CommServ Agent sends a ping message to the service. The service responds to the ping message and the service is added to the platform. A service can be virtually any program that performs a specific task and shares its resources with the platform. These programs can provide methods to access data bases, manage connections, analyze data, get information from external devices (e.g., sensors, readers and screens), publish information, or even make use of other services. Developers have are free to use any programming language. The only requirement is that they must follow the communication protocol based on transactions of XML (SOAP) messages [11].

In the next section, a case study is presented, where FUSION@ has helped to develop a completely functional multi-agent system aimed at improving several aspects of dependent people. This system introduces all functionalities in FUSION@.

4. **Using FUSION@ to Improve ALZ-MAS, a Multi-Agent System for Health Care.** Ambient Intelligence based systems aim to improve quality of life, offering more efficient and easy ways to use services and communication tools to interact with other people, systems and environments. Among the general population, those most likely to benefit from the development of these systems are the elderly and dependent persons, whose daily lives, with particular regard to health care, will be most enhanced [13, 52]. Dependent persons can suffer from degenerative diseases, dementia, or loss of cognitive ability [14]. In Spain, dependency is classified into three levels [14]: Level 1 (moderated dependence) refers to all people that need help to perform one or several basic daily life activities, at least once a day; Level 2 (severe dependence) consists of people who need help to perform several daily life activities two or three times a day, but who do not require the support of a permanent caregiver; and finally, Level 3 (great dependence) refers to all people who need support to perform several daily life activities numerous times a day and, because of their total loss of mental or physical autonomy, need the continuous and permanent presence of a caregiver.

Agents and multi-agent systems in dependency environments are becoming a reality, especially in health care. Most agents-based applications are related to the use of this technology in the monitoring of patients, treatment supervision and data mining. Lanzola et al. [26] present a methodology that facilitates the development of interoperable intelligent software agents for medical applications, and propose a generic computational model for implementing them. The model may be specialized in order to support all the different information and knowledge-related requirements of a hospital information system. Meunier [32] proposes the use of virtual machines to support mobile software agents by using a functional programming paradigm. This virtual machine provides the application developer with a rich and robust platform upon which to develop distributed mobile agent applications, specifically when targeting distributed medical information and distributed image processing. While an interesting proposal, it is not viable due to the security reasons that affect mobile agents, and there is no defined alternative for locating patients or generating planning strategies. There are also agents-based systems that help patients to get the best possible treatment, and that remind the patient about follow-up tests [33]. They assist the patient in managing continuing ambulatory conditions (chronic problems). They also provide health-related information by allowing the patient to interact with the on-line health care information network. Decker and Li [15] propose a system to increase hospital efficiency by using global planning and scheduling techniques. They propose a multi-agent solution that uses the generalized partial global planning approach which preserves the existing human organization and authority structures, while providing better system-level performance (increased hospital unit throughput and decreased inpatient length of stay time). To do this, they use resource constraint scheduling

to extend the proposed planning method with a coordination mechanism that handles mutually exclusive resource relationships. Other applications focus on home scenarios to provide assistance to elderly and dependent persons. RoboCare presents a multi-agent approach that covers several research areas, such as intelligent agents, visualization tools, robotics and data analysis techniques to support people with their daily life activities [38]. TeleCARE is another application that makes use of mobile agents and a generic platform in order to provide remote services and automate an entire home scenario for elderly people [10]. Although these applications expand the possibilities and stimulate research efforts to enhance the assistance and health care provided to elderly and dependent persons, none of them integrate intelligent agents, distributed and dynamic applications and services approach, or the use of reasoning and planning mechanisms into their model.

FUSION@ has been employed to develop a multi-agent system for dependent environments ALZ-MAS 2.0. ALZ-MAS 2.0 is an improved version of a previous existing solution ALZ-MAS (ALZheimer Multi-Agent System) [13], a multi-agent system aimed at enhancing the assistance and health care for Alzheimer patients living in geriatric residences. ALZ-MAS 1.0 [13] was a multi-agent system developed to monitor and supervise Alzheimer patients in geriatric residences. ALZ-MAS 1.0 used RFID technology to locate the patients and manage alarms. The main functionalities in this system included reasoning and planning mechanisms for scheduling the activities of the medical staff that are embedded into deliberative BDI agents. The main problem with ALZ-MAS 1.0 was the efficiency of the computational planning processes as well as the integration with other existing architectures. The main characteristics of the ALZ-MAS 1.0 are briefly described in the following paragraphs, followed by a description of the new ALZ-MAS 2.0 system developed by means of the FUSION@ architecture. The main components of ALZ-MAS 2.0 have been modeled as distributed and independent services by means of FUSION@, releasing the agents from high demanding computational processes.

As shown in Figure 5, ALZ-MAS 1.0 structure has five different deliberative agents based on the BDI model (BDI Agents), each one with specific roles and capabilities:

- **Patient Agent**. It manages the patients' personal data and behavior (monitoring, location, daily tasks and anomalies).
- **Mobile Agent**. It runs on mobile devices and executes the nurse and doctor roles. This agent corresponds to an Interface Agent in FUSION@. This agent inserts new tasks to be processed by a Case-Based Reasoning mechanism. There is one Mobile Agent for each doctor and nurse connected to the system.
- **Nurse Agent**. It is a BDI agent with a Case-Based Planning (CBP) mechanism embedded in its structure. It schedules the users' daily activities and obtains dynamic plans depending on the tasks needed for each user. It manages scheduled-users profiles (preferences, habits, holidays, etc.), tasks, available time and resources. Every agent generates personalized plans depending on the user profile. There is one Nurse Agent for each nurse connected to the system.
- **Manager Agent**. This agent plays two roles: the security role that monitors the users' location and physical building status (temperature, lights, alarms, etc.); and the manager role that handles the databases and the task assignment. It must provide security for the users and ensure the efficiency of the tasks assignments. These assignments are carried out through a Case-Based Reasoning (CBR) mechanism, which is incorporated within the Manager Agent. When a new assignment of tasks needs to be carried out, the agent recalls both past experiences and the needs of the current situation in order to allocate the appropriate task. There is just one Manager Agent running in the system.

- **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 (temperature, lights, door locks, alarms, etc.), and also checks the status of the wireless devices connected to the system (e.g., PDAs). The information obtained is sent to the Manager Agent for processing. There is just one Devices Agent running in the system.

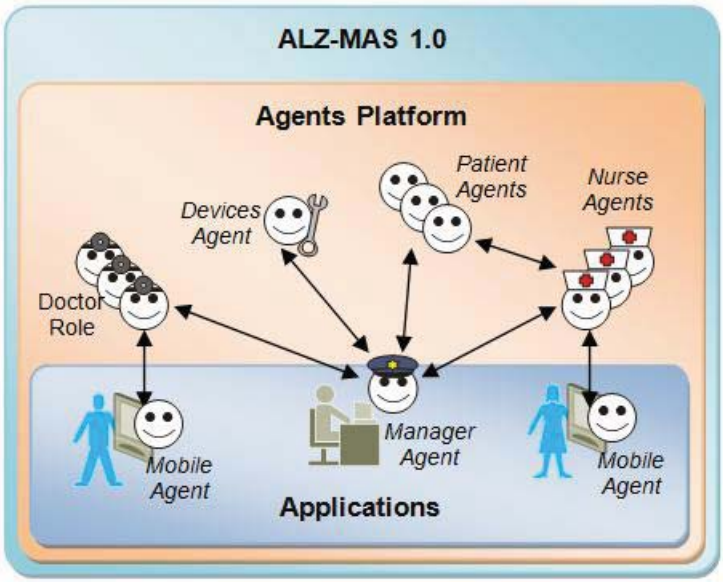

FIGURE 5. ALZ-MAS 1.0 basic structure

The main problem with ALZ-MAS 1.0 is the efficiency of the computational planning processes as well as the integration with other existing architectures. In ALZ-MAS 2.0, the main functionalities have been modeled as distributed and independent services by means of FUSION@, releasing the agents from high demanding computational processes. In ALZ-MAS 1.0, each agent integrates 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 is an important issue in ALZ-MAS 1.0, since agents running on mobile devices (i.e., 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 planning mechanisms are integrated into the agents. These mechanisms are busy almost all the time, overloading the respective agents. Because these 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 planning mechanisms into deliberative BDI agents makes these agents complex and unable to be executed on mobile devices.

As can be seen in Figure 6, the entire ALZ-MAS 2.0 structure has been modified according to FUSION@ model, separating most of the agents' functionalities from those to be modeled 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. As an example showing the differences between both

approaches, the next sub-section describes the CBP mechanism that has been extracted from the Nurse Agent structure and modeled as a service.

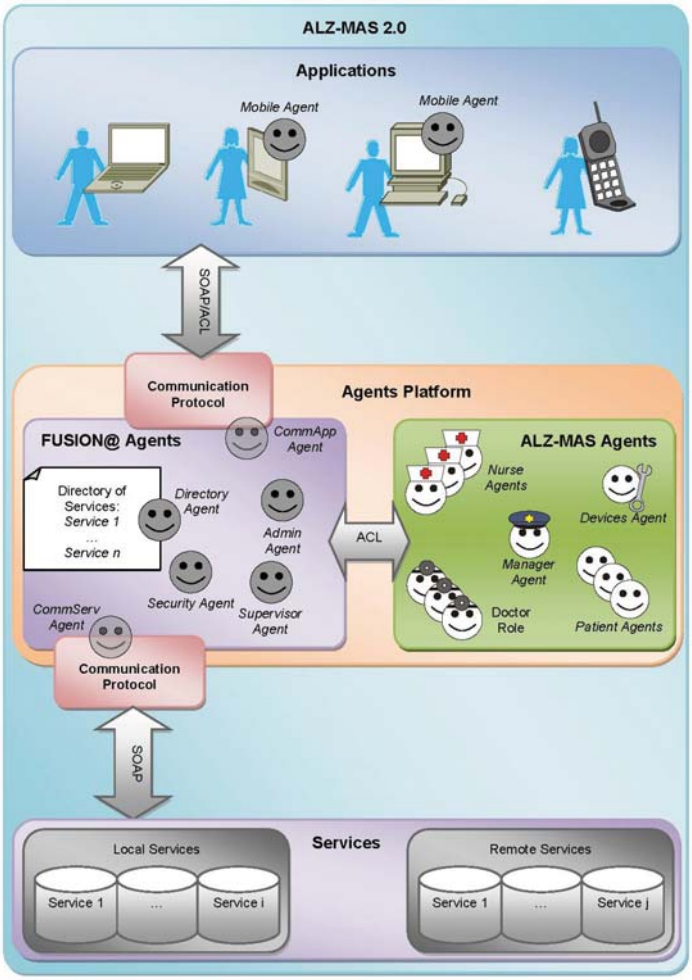

FIGURE 6. ALZ-MAS 2.0 basic structure

4.1. **Case-based planning service.** As previously mentioned, some agents in ALZ-MAS integrate Case-Based Reasoning (CBR) and Case-Based Planning (CBP) mechanisms, 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. To demonstrate the capabilities of FUSION@ in integrating complex services, the most complex and resources demanding task, in this case the CBP mechanism, has been modeled as a service. The main characteristics of this mechanism are described in the remainder of this section.

Case-based Reasoning (CBR) is a type of reasoning based on past experiences [34]. CBR systems 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 systems 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.

CBP comes from CBR, but is specially designed to generate plans (sequence of actions) [13]. 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 [21]. 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 case-based reasoning, the case representation, the plans memory organization, and the algorithms used in every stage of the case-based planning cycle are essential in defining an efficient planner.

FUSION@ can take advantage of these characteristics in order to enhance the services organization. In ALZ-MAS 1.0, the CBR and CBP mechanisms are deeply integrated into the agents' structure. Following FUSION@ model, these mechanisms have been modeled as services linked to agents, thus increasing the system's overall performance. To generate a new plan, an Interface Agent (running on a mobile device) sends an XML to the platform. The message is processed by the Admin Agent which decides to use the planner service (PlanServ). The platform invokes the service sending an XML message. PlanServ 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 Nurse Agents and the corresponding Interface Agents running.

The CBP service used in ALZ-MAS creates optimal paths and scheduling in order to facilitate the completion of all tasks defined for the nurses connected to the system. Figure 7 shows the mains steps to generate a new plan. First, an application (e.g., Interface Agent) sends a SOAP message to the platform. The message is processed by the Admin Agent which decides to use the planner service. The platform invokes the service sending a SOAP message. The location of each nurse and patient in the residence is also sent to the planner service. The service receives the message and starts to generate a new plan by means of the CBP cycle. Then, the solution is sent to the platform which delivers the new plan to all nurses connected to the system. 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. A task is a Java object that contains a set of parameters: *TaskId,*

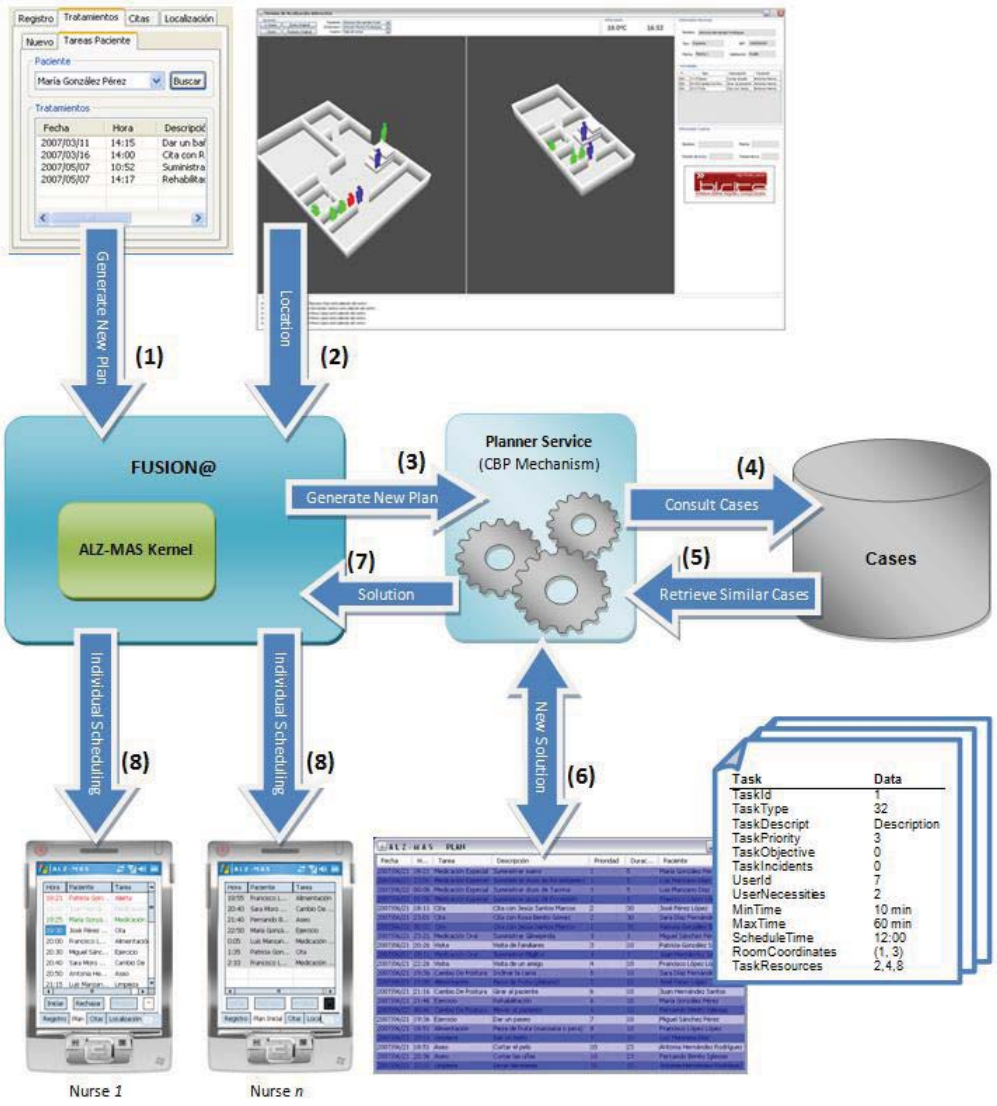

FIGURE 7. Steps to generate a new plan

TaskType, TaskDescript, TaskPrority, TaskObjective, TaskIncidents, UserId, UserNecessities, MinTime, MaxTime, ScheduleTime, RoomCoordinates and TaskResources. These parameters are processed by the CBP mechanism in order to create optimal plans. For example, a task with a maximum priority ($TaskPrority = 1$) must be completed before, even if other tasks have the same time scheduled.

In this case, a plan is the name given to a sequence of tasks that defines the best path for each nurse. A problem description (belief) will be formed by the tasks that the nurses need to execute, the location, the resources available, and the times assigned for their shift. In the retrieve stage, those problem descriptions found within a range of similarity close to the original problem description are recovered from the beliefs base. In our case, a tolerance of 20% has been permitted. In order to do this, the CBP service allows the application of different similarity algorithms (cosine, clustering, etc.). Once the most similar problem descriptions have been selected, the solutions associated with them are recovered. One solution contains all the plans (sequences of tasks) carried out in order to achieve the objectives for a problem description (assuming that re-planning is possible) in the past, as well as the efficiency of the solution being supplied. The chosen solutions are

combined in the reuse stage to construct a plan. The reuse is focused on the objectives and resources needed by each task, as well as on the objectives that the nurses need to perform and the resources available in order to carry out the global plan (compilation of all tasks for all nurses). The objectives that each nurse has are aimed to attend the patients and not exceed eight working hours. The time available is a problem restriction. The revision of the plans is made by the nurses. If the evaluation of the plan is at least a 90% similar, the case is stored in the cases memory.

The variation of the plans will essentially be induced by: the changes that occur in the environment and that force the initial plan to be modified; and the knowledge from the success and failure of the plans that were used in the past, and which are favored or punished via learning.

If there are no similar cases, the entire planning process must be performed. As the objective of this paper is not describing in detail the CBP mechanism but the distributed approach used to model it as a service, this mechanism is briefly introduced. The planning is carried out through a neural network based on the Kohonen network [9, 27]. The basic Kohonen network [25] cannot be used to resolve our problem since it attempts to minimize distances without taking into account any other type of restriction, such as time limits. In our case, the planner is based on Kohonen networks but with a number of improvements that allow us to reach a solution far more rapidly. Furthermore, once a solution has been reached, it is re-modified in order to take restrictions into account. As such, by modifying the basic algorithm, we are aiming to make the solution search more flexible. In order to achieve this, the basic vicinity function used in the Kohonen network is modified and the number of neurons in the output layer corresponds to the places that the subject wishes to visit. The case study could be easily generalized to similar Ambient Intelligence environments, such as routes surveillance for security guards and shopping centers. Next, the results obtained after applying a distributed approach in ALZ-MAS by means of FUSION@ are presented.

5. **Results and Conclusion.** FUSION@ facilitates the development of Ambient Intelligence based multi-agent systems. Its model is based on a SOA approach, by formalizing services, applications, communications and deliberative agents. The architecture proposes an alternative where agents are based on the BDI (Belief, Desire, Intention) model and act as controllers and coordinators. FUSION@ exploits the agents' characteristics to provide a robust, flexible, modular and adaptable solution that can cover most requirements of a wide diversity of Ambient Intelligence projects. All functionalities, including those of the agents, are modeled as distributed services and applications. FUSION@ sets on top of existing agents' frameworks by adding new layers for integrating a SOA approach and facilitating the distribution and management of resources. Figure 8 shows how FUSION@ adds these features to common agents' frameworks, such as JADE, OAA and RETSINA and improves the services provided by these previous architectures. JADE implements a "container" philosophy. Each instance of the JADE run-time is called container. Each agent has a container (Container Agent) and there is a Main Container which encloses all containers. Communication between agents is done through FIPA ACL [6]. In OAA, the communication and cooperation between agents are negotiated by "facilitators" using ICL (Interagent Communication Language) [12]. Facilitators manage all requests from users and agents. RETSINA implements a Hierarchical Task Network (HTN) which is a planning methodology that creates plans (i.e., sequence of actions) by task decomposition. RETSINA includes a Communicator for managing and communicating with the

agents using KQML (Knowledge Query and Manipulation Language) [49]. These previous architectures have limited communication abilities and are not compatible with SOA architectures.

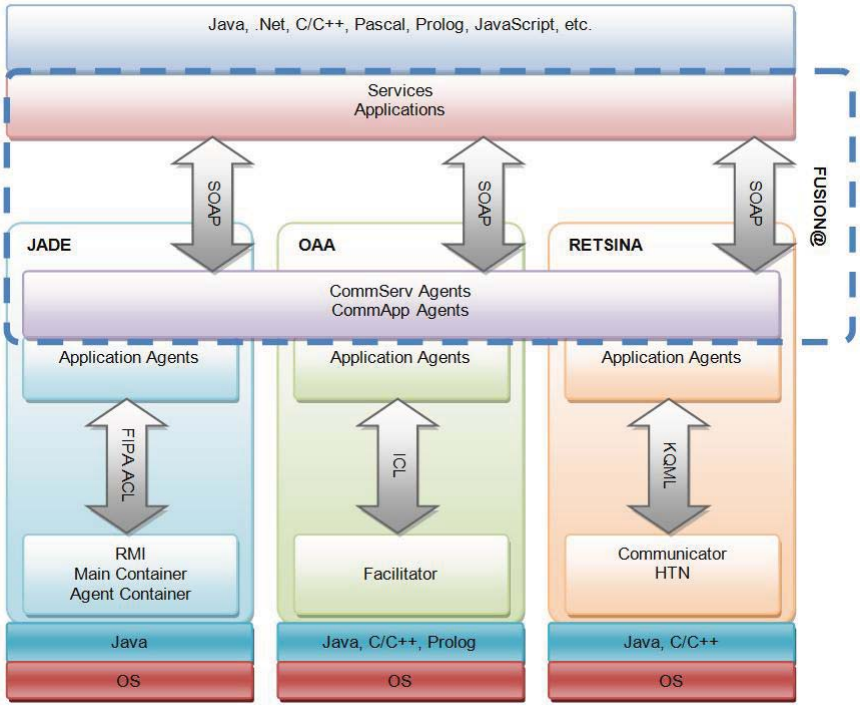

FIGURE 8. FUSION@ over JADE, OAA and RETSINA frameworks

Initially, FUSION@ was mainly designed to develop Ambient Intelligence based systems, especially for scenarios involving dependent and elderly persons. In fact, it has been employed to develop an improved version of ALZ-MAS 2.0, a system aimed at enhancing assistance and health care for Alzheimer patients in geriatric residences, improving the previous ALZ-MAS 1.0 solution [13].

Figure 9 shows the main user interface of ALZ-MAS 1.0 (a) and ALZ-MAS 2.0 (b), which 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 the planning mechanism has been modeled as a Service. Table 1 shows the main differences between ALZ-MAS 1.0 and ALZ-MAS 2.0 in terms of architectural functionalities. The characteristics of the architecture used to develop the ALZ-MAS system can be comparable to those provided for multi-agent architectures as Open Agent Architecture or RETSINA. ALZ-MAS 2.0 incorporates functional improvements compared with ALZ-MAS 1.0, such as a better communication capacity, improved integration with SOA-based applications and distribution of resources, as well as more scalability and robustness.

Several tests have been done to compare the overall performance of ALZ-MAS 1.0 and ALZ-MAS 2.0, the latter making use of FUSION@. The test scenario was established in the same residence as the previous ALZ-MAS 1.0 solution [13]. It was used a HP Pavilion a1319.pt (P4 517 2.93GHz, 512MB RAM) as central server and several HP iPAQ 210 Enterprise Handhelds (624 MHz Marvell XScale PXA310, 128MB RAM, 256MB flash).

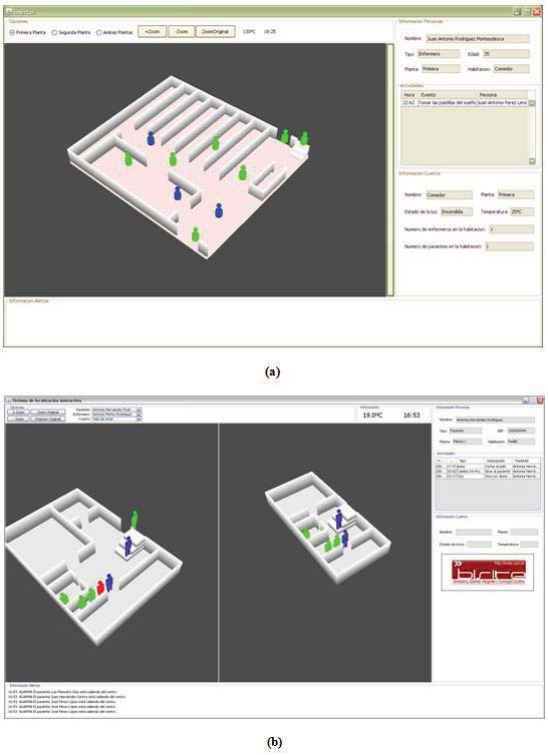

FIGURE 9. (a) ALZ-MAS 1.0 main user interface; (b) ALZ-MAS 2.0 main user interface

TABLE 1. ALZ-MAS 1.0 and ALZ-MAS 2.0 comparison

	ALZ-MAS 1.0	ALZ-MAS 2.0
Architecture	Agents	Agents + SOA
Framework	JADE	JADE + FUSION@
Communication protocol	ACL	ACL, SOAP
Types of Agents	5	12
Operating System	Linux, Windows 2000/XP/2003 Server/Vista	Linux, Windows 2000/XP/2003 Server/Vista
Programming Language	Java	Java/C++/.Net
Distributed resources	Agents	Agents, Services
Scalability	Agents replication	Agents and services replication
Reusability	Functionalities are deeply integrated into the agents' structure.	Functionalities are modeled as independent services outside of the agents' structure. Services can be used in other developments.

The network infrastructure consisted of a 100-Mbps Ethernet local area network (LAN) and a Wi-Fi wireless local area network (WLAN) working at 2.4 GHz. The Manager and Patient agents run on the central server, while Mobile agents run on each PDA. Finally, services were located in five different HP Pavilion Workstations (Intel Core 2 Duo E7200; 4GB RAM). The tests consisted of a set of requests delivered to the planning

mechanism (CBP) described in Section 4.1 which in turn had to generate paths for each set of tasks. 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. 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 CBR mechanism increases or decreases ScheduleTime and MaxTime according to the priority of the task:

$$ScheduleTime = ScheduleTime - 5min * TaskPriority \quad (1)$$

$$MaxTime = MaxTime + 5min * TaskPriority \quad (2)$$

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 7 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 and with a max QoS value (1) were replicated.

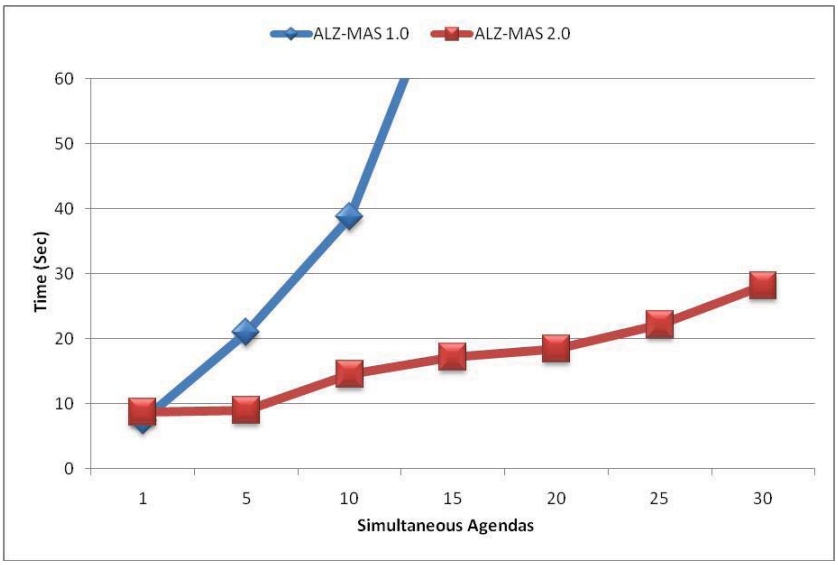

FIGURE 10. Time needed for both systems to generate paths for a group of simultaneous agendas

Figure 10 shows the average time needed by both systems to generate the paths for a fixed number of simultaneous agendas. As can be seen in Figure 10, ALZ-MAS 1.0 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 5 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 ALZ-MAS 1.0 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. As shown in Figure 10, we have compared the approach presented in this paper with the previous existing approach. It

is difficult to compare our approach with other existing architectures because it was not possible to find resource distribution mechanisms as the proposed in this work. Figure 10 shows how one of the main problems of ALZ-MAS 1.0 has been solved: the performance of ALZ-MAS 1.0 was notably reduced when multiple simultaneous agendas (more than 5) were used, which sometimes drove the system to collapse. In this sense, we can observe how ALZ-MAS 2.0 provides the expected behavior, that is, a more stable and constant distribution of the work load, and a more reasonable response time.

Figure 11 shows the number of crashed agents and services for both versions of ALZ-MAS during tests. None of the tests where agents or services crashed were taken into account to calculate data, so these tests were repeated. As can be seen, ALZ-MAS 1.0 is far more unstable than ALZ-MAS 2.0, especially when comparing the number of crashed agents. These data demonstrate that this approach provides a higher ability to recover from errors. As shown in Figure 11, the use of FUSION@ provides a more effective use of resources and reduces the number of crashes. Thus, the system can support multiple simultaneous agendas with a reasonable number of agent and service crashes. ALZ-MAS 2.0 is more effective than ALZ-MAS 1.0 taking into account the reliability of the system and the completion of tasks.

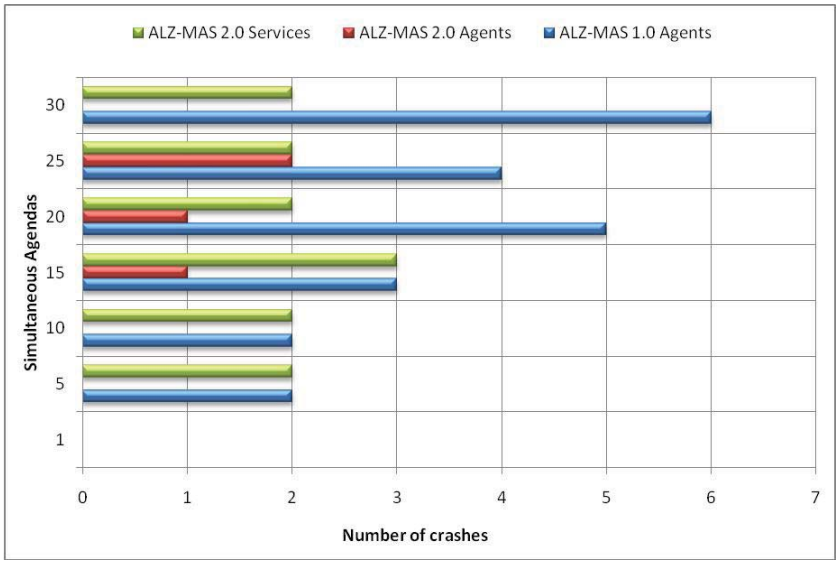

FIGURE 11. Number of agents and services crashed at both versions of ALZ-MAS

As a conclusion we can say that although FUSION@ is still under development, preliminary results demonstrate that it is adequate for building complex systems and exploiting composite services, in this case ALZ-MAS 2.0 and the CBP mechanism presented in this paper. The CBP mechanism is just an example that shows how to deploy distributed services using FUSION@. However, services can be any functionality (mechanisms, algorithms, routines, etc.) designed and deployed by developers. FUSION@ has laid the groundwork to boost and optimize the development of future projects and systems based on Ambient Intelligence. This architecture facilitates the development of systems based on the paradigm of Ambient Intelligence because it is able to solve problems at execution time over highly dynamic and distributed environments. FUSION@ makes it easier for developers to integrate independent services and applications because they are not restricted to programming languages supported by the agents' frameworks used (e.g., JADE, OAA, RETSINA). The distributed approach of FUSION@ optimizes usability and performance

because it can be obtained lighter agents by modeling the systems' functionalities as independent services outside of the agents' structure, thus services may be used in other developments.

FUSION@ allows interoperating with as many services and applications as developers and users require. CommServ or CommApp Agents may be cloned is there are needed more communication resources. Nevertheless, all these CommServ and CommApp Agents must communicate with an only Admin Agent, thus provoking an undesired bottleneck when working at a high work load. In this sense, the BISITE Research Group is currently working on the design and development of mechanisms aimed at distributing and balancing the work load directly inside FUSION@. Therefore, in the future FUSION@ will allow the existence of multiple Admin Agents running on the same agent platform, thus balancing the work load and increasing the efficiency of the systems implemented using this architecture.

Acknowledgment. This work has been supported by the MICINN PET2008_0036 project.

REFERENCES

[1] E. Aarts and B. de Ruyter, New research perspectives on ambient intelligence, *Journal of Ambient Intelligence and Smart Environments*, vol.1, no.1, pp.5-14, 2009.

[2] S. Ahuja and S. Chandrasekaran, *Application Development for IBM WebSphere Process Server 7 and Enterprise Service Bus 7*, Packt Publishing, 2010.

[3] R. N. Anderson, A method for constructing complete annual U.S. life tables, vital health statistics, *National Center for Health Statistics*, vol.2, no.129, pp.1-28, 1999.

[4] L. Ardissono, G. Petrone and M. Segnan, A conversational approach to the interaction with web services, *Computational Intelligence*, vol.20, pp.693-709, 2004.

[5] I. N. Athanasiadis, M. Milis, P. A. Mitkas and S. C. Michaelides, A multi-agent system for meteorological radar data management and decision support, *Environmental Modelling & Software*, vol.24, no.11, pp.1264-1273, 2009.

[6] F. Bellifemine, A. Poggi and G. Rimassa, Jade: A FIPA-compliant agent framework, *Proc. of PAAM 1999*, pp.97-108, 1999.

[7] L. O. B. da Silva, F. Ramparany, P. Dockhorn, P. Vink, R. Etter and T. Broens, A service architecture for context awareness and reaction provisioning, *IEEE Congress on Services*, pp.25-32, 2007.

[8] F. M. T. Brazier, J. O. Kephart, H. Van D. Parunak and M. N. Huhns, Agents and service-oriented computing for autonomic computing: A research agenda, *IEEE Internet Computing*, vol.13, no.3, pp.82-87, 2009.

[9] R. Z. Cabada, M. L. B. Estrada and C. A. R. Garca, EDUCA: A web 2.0 authoring tool for developing adaptive and intelligent tutoring systems using a Kohonen network, *Expert Systems with Applications*, vol.38, no.8, pp.9522-9529, 2011.

[10] L. M. Camarinha-Matos and H. Afsarmanesh, A comprehensive modeling framework for collaborative networked organizations, *Journal of Intelligent Manufacturing*, vol.18, no.5, pp.529-542, 2007.

[11] E. Cerami, *Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*, 1st Edition, O'Reilly & Associates, Inc., 2002.

[12] A. Cheyer and D. Martin, The open agent architecture, *Autonomous Agents and Multi-Agent Systems*, vol.4, no.1, pp.143-148, 2001.

[13] J. M. Corchado, J. Bajo, Y. de Paz and D. I. Tapia, Intelligent environment for monitoring alzheimer patients, agent technology for health care, *Decision Support Systems*, vol.44, no.2, pp.382-396, 2008.

[14] J. Costa-Font and C. Patxot, The design of the long-term care system in Spain: Policy and financial constraints, *Social Policy and Society*, vol.4, no.1, pp.11-20, 2005.

[15] K. Decker and J. Li, Coordinated hospital patient scheduling, *Proc. of the 3rd International Conference on Multi-Agent Systems*, pp.104-111, 1998.

[16] D. V. Dimarogonas and K. J. Kyriakopoulos, A connection between formation infeasibility and velocity alignment in kinematic multi-agent systems, *Automatica*, vol.44, no.10, pp.2648-2654, 2008.

[17] P. Erickson, R. Wilson and I. Shannon, Years of healthy life, US department of health and human services, CDC, national center for health statistics, Hyattsville, Maryland, *Statistical Notes*, no.7, 1995.
[18] T. Erl, *SOA Design Patterns*, 1st Edition, Prentice Hall PTR, 2009.
[19] S. Franklin and A. Graesser, Is it an agent, or just a program?: A taxonomy for autonomous agents, *Proc. of the 3rd International Workshop on Agent Theories, Architectures and Languages, LNCS*, vol.1193, pp.21-35, 1996.
[20] J. A. Giampapa, K. P. Sycara, S. R. Owens et al., An agent-based C4ISR testbed, *The 8th International Conference on Information Fusion*, vol.2, 2005.
[21] M. Glez-Bedia and J. M. Corchado, A planning strategy based on variational calculus for deliberative agents, *Computing and Information Systems Journal*, vol.10, pp.2-14, 2002.
[22] J. González, M. Ruiz, E. Barrera, J. M. López, G. de Arcas and J. Vega, Configuration and supervision of advanced distributed data acquisition and processing systems for long pulse experiments using JINI technology, *Fusion Engineering and Design*, vol.84, no.2-6, pp.832-836, 2009.
[23] N. Hawes, J. Wyatt and A. Sloman, Exploring design space for an integrated intelligent system, *Knowledge-Based Systems*, vol.22, no.7, pp.509-515, 2009.
[24] G. T. Jayaputera, A. B. Zaslavsky and S. W. Loke, Enabling run-time composition and support for heterogeneous pervasive multi-agent systems, *Journal of Systems and Software*, vol.80, no.12, pp.2039-2062, 2007.
[25] T. Kohonen, *Self-Organization and Associative Memory*, 3rd Edition, Springer-Verlag, New York, NY, USA, 1989.
[26] G. Lanzola, L. Gatti, S. Falasconi and M. Stefanelli, A framework for building cooperative software agents in medical applications, *Artificial Intelligence in Medicine*, vol.16, no.3, pp.223-249, 1999.
[27] K. S. Leung, H. D. Jin and Z. B. Xu, An expanding self-organizing neural network for the traveling salesman problem, *Neurocomputing*, vol.62, pp.267-292, 2004.
[28] Y. Li, W. Shen and H. Ghenniwa, Agent-based web services framework and development environment, *Computational Intelligence*, vol.20, no.4, pp.678-692, 2004.
[29] X. Liu, A multi-agent-based service-oriented architecture for inter-enterprise cooperation system, *Proc. of the 2nd International Conference on Digital Telecommunications*, pp.22, 2007.
[30] D. Martin, A. Cheyer and D. Moran, The open agent architecture: A framework for building distributed software systems, *Applied Artificial Intelligence*, vol.13, no.1/2, pp.91-128, 1999.
[31] G. Matthew, *802.11 Wireless Networks: The Definitive Guide*, O'Reilly & Associates Inc, 2005.
[32] J. A. Meunier, A virtual machine for a functional mobile agent architecture supporting distributed medical information, *Proc. of the 12th IEEE Symposium on Computer-Based Medical Systems*, pp.177-182, 1999.
[33] S. Miksch, Plan management in the medical domain, *AI Communications*, vol.12, no.4, pp.209-235, 1999.
[34] S. Montani and L. Jain, Innovations in case-based reasoning applications, *Successful Case-based Reasoning Applications – I*, vol.305, pp.1-5, 2010.
[35] J. Nealon and A. Moreno, Applications of software agent technology in the health care domain, *Birkhauser, Whitestein Series in Software Agent Technologies*, 2003.
[36] R. R. Negenborn, B. D. Schutter and J. Hellendoorn, Multi-agent model predictive control for transportation networks: Serial versus parallel schemes, *Engineering Applications of Artificial Intelligence*, vol.21, no.3, pp.353-366, 2008.
[37] S. Otón, A. Ortiz, J. R. Hilera, R. Barchino, J. M. Gutiérrez, J. J. Martínez, J. A. Gutiérrez, L. De Marcos and L. Jiménez, Service oriented architecture for the implementation of distributed repositories of learning objects, *International Journal of Innovative Computing, Information and Control*, vol.6, no.3(A), pp.843-854, 2010.
[38] F. Pecora and A. Cesta, Dcop for smart homes: A case study, *Computational Intelligence*, vol.23, no.4, pp.395-419, 2007.
[39] A. Pokahr, L. Braubach and W. Lamersdorf, Jadex: A BDI reasoning engine, *Multi-Agent Programming. Multiagent Systems, Artificial Societies, and Simulated Organizations*, vol.15, pp.149-174, 2005.
[40] A. Ricci, C. Buda and N. Zaghini, An agent-oriented programming model for SOA & web services, *Proc. of the 5th IEEE International Conference on Industrial Informatics*, pp.1059-1064, 2007.
[41] P. Rigole, T. Holvoet and Y. Berbers, Using Jini to integrate home automation in a distributed software-system, *The 4th International Workshop on Distributed Communities on the Web, LNCS*, vol.2468, pp.291-304, 2002.

[42] M. Sancho, A. Abellán, L. Prez and J. A. Miguel, Ageing in Spain: Second world assembly on ageing, *IMSERSO*, Madrid, Spain, 2002.
[43] B. Sasikumar and V. Vasudevan, Improving the performance of TCP/IP over wireless networks with a RETSINA agent, *The 1st International Conference on Emerging Trends in Engineering and Technology*, pp.151-156, 2008.
[44] B. Schon, G. M. P. O'Hare, B. R. Duffy, A. N. Martin and J. F. Bradley, Agent assistance for 3D world navigation, *LNCS*, vol.3661, pp.499-499, 2005.
[45] K. T. Seow and K. M. Sim, Collaborative assignment using belief-desire-intention agent modeling and negotiation with speedup strategies, *Information Sciences*, vol.178, no.4, pp.1110-1132, 2008.
[46] M. O. Shafiq, Y. Ding and D. Fensel, Bridging multi agent systems and web services: Towards interoperability between software agents and semantic web services, *Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference*, pp.85-96, 2006.
[47] L. Snidaro and G. L. Foresti, Knowledge representation for ambient security, *Expert Systems*, vol.24, no.5, pp.321-333, 2007.
[48] C. J. Su and C. Y. Wu, JADE implemented mobile multi-agent based, distributed information platform for pervasive health care monitoring, *Applied Soft Computing*, vol.11, no.1, pp.315-325, 2011.
[49] K. Sycara, M. Paolucci, M. Van Velsen and J. Giampapa, The RETSINA MAS infrastructure, *Autonomous Agents and Multi-Agent Systems*, vol.7, no.1-2, pp.29-48, 2003.
[50] P.-C. Tseng, C.-Y. Chen, W.-S. Hwang, J.-S. Pan and B.-Y. Liao, QoS-aware residential gateway supporting ZigBee-related services based on a service-oriented architecture, *International Journal of Innovative Computing, Information and Control*, vol.6, no.6, pp.2803-2816, 2010.
[51] R. Vaculin, R. Neruda and K. Sycara, The process mediation framework for semantic web services, *International Journal of Agent-Oriented Software Engineering*, vol.3, no.1, pp.27-58, 2009.
[52] K. van Woerden, Mainstream developments in ICT: Why are they important for assistive technology? *Technology and Disability*, vol.18, no.1, pp.15-18, 2006.
[53] H. Voos, Agent-based distributed resource allocation in technical dynamic systems, *Proc. of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, pp.157-162, 2006.
[54] C. Walton, *Agency and the Semantic Web*, Oxford University Press, 2006.
[55] W. Weber, J. M. Rabaey and E. Aarts, *Ambient Intelligence*, Springer-Verlag, New York, 2005.
[56] M. Woolridge and M. J. Wooldridge, *Introduction to Multiagent Systems*, John Wiley & Sons, New York, USA, 2002.
[57] *Global Age-friendly Cities: A Guide*, World Health Organization, 2007.