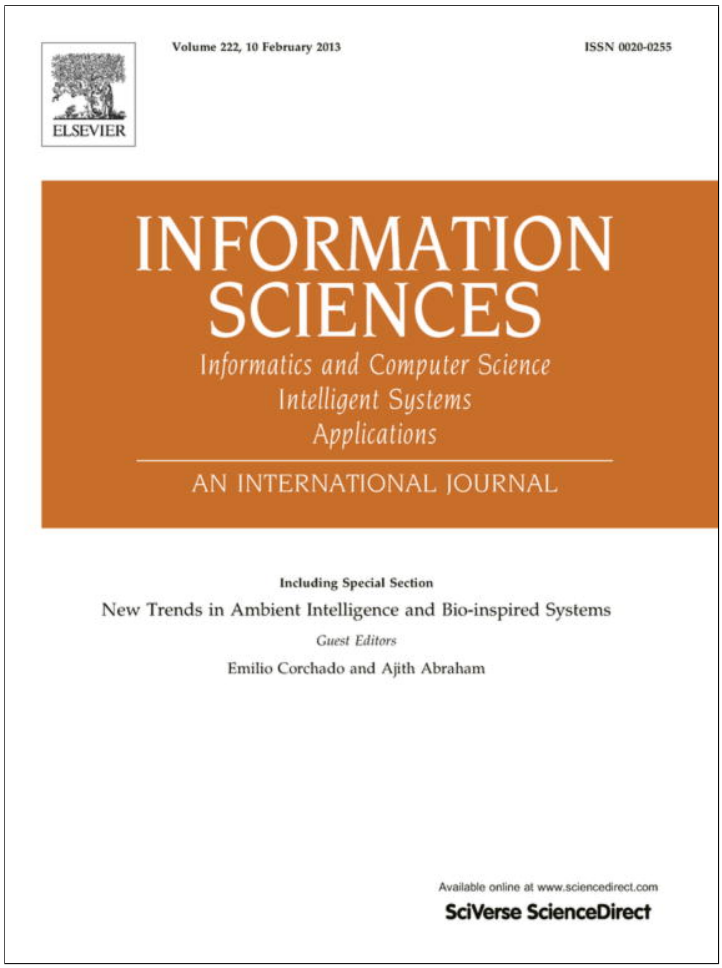
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Information Sciences 222 (2013) 47–65

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Integrating hardware agents into an enhanced multi-agent architecture for Ambient Intelligence systems

Dante I. Tapia [a,*], Juan A. Fraile [b], Sara Rodríguez [a], Ricardo S. Alonso [a], Juan M. Corchado [a]

[a] Department of Computers and Automation, University of Salamanca, Plaza de la Merced s/n, 37008 Salamanca, Spain
[b] Faculty of Informatics, Pontifical University of Salamanca, Compañía, 5, 37002 Salamanca, Spain

ARTICLE INFO

Article history:
Available online 14 May 2011

Keywords:
Ambient Intelligence
Distributed architectures
Multi-agent systems
Service-oriented architecture
Wireless sensor networks

ABSTRACT

Ambient Intelligence (AmI) systems require the integration of complex and innovative solutions. In this sense, agents and multi-agent systems have characteristics such as autonomy, reasoning, reactivity, social abilities and pro-activity which make them appropriate for developing distributed systems based on Ambient Intelligence. In addition, the use of context-aware technologies is an essential aspect in these developments in order to perceive stimuli from the context and react to it autonomously. This paper presents the integration of the *Hardware-Embedded Reactive Agents* (HERA) Platform into the *Flexible and User Services Oriented Multi-agent Architecture* (FUSION@), a multi-agent architecture for developing AmI systems that integrates intelligent agents with a service-oriented architecture approach. Because of this integration, FUSION@ has the ability to manage both software and hardware agents by using self-adaptable heterogeneous wireless sensor networks. Preliminary results presented in this paper demonstrate the feasibility of FUSION@ as a future alternative for developing Ambient Intelligence systems where users and systems can use both software and hardware agents in a transparent way, achieving a higher level of ubiquitous computing and communication.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

The search for software capable of better adapting to people's needs and particular situations has led to the development of Ambient Intelligent (AmI) systems [43]. AmI tries to adapt technology to people's needs by incorporating omnipresent computing elements that communicate ubiquitously among themselves [29]. These systems capture and manage relevant information that surrounds them and constitutes the context [19]. Individuals are also becoming increasingly accustomed to living with more and more technology in the hope of increasing their quality of life and facilitating their daily activities. Nowadays, there is a wide range of small, portable and non-intrusive devices intended to help users with their daily life [31]. These devices allow agents to gather context-information in a dynamic and distributed way [8]. However, the integration of such devices is not an easy task. Therefore, 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.

Along these lines, the implementation of distributed architectures has been presented as a solution to such problems [23]. 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) consider integration and performance to be aspects that must be taken into account when functionalities are created outside the system [8]. The development of AmI-based

* Corresponding author. Tel.: +34 923 294400x1525; fax: +34 923 294514.
E-mail addresses: dantetapia@usal.es (D.I. Tapia), jafraileni@upsa.es (J.A. Fraile), srg@usal.es (S. Rodríguez), ralorin@usal.es (R.S. Alonso), corchado@usal.es (J.M. Corchado).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.05.002

Author's personal copy

systems that integrate different subsystems demands the creation of complex and flexible applications. As the complexity of an application increases, it needs to be divided into modules with different functionalities. Since different applications could require similar functionalities, there is a trend towards the reutilization of resources that can be implemented as part of other systems. One of the most prevalent alternatives in distributed architectures is Multi-Agent Systems (MAS) which can help to distribute resources and reduce the central unit tasks [2]. A distributed agent-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]. Multi-agent systems have been successfully applied to several scenarios, such as education, culture, entertainment, medicine, commerce, robotics, etc. [15,21,40].

AmI-based developments require the use of sensors strategically distributed over the environment. In this sense, sensors expand the agents' context-aware capabilities in order to change dynamically their behavior and personalize their reactions [19]. Sensor networks are used for gathering the information needed by intelligent environments, whether in home automation, industrial applications, etc. [42]. Sensor networks need to be fast and easy to install and maintain [30]. It is possible to distinguish between two types of sensor networks: wired and wireless. Wireless sensor networks (WSN) are more flexible and require less infrastructural support than wired sensor networks. Although there are plenty of technologies for implementing WSNs (e.g., ZigBee, Wi-Fi or Bluetooth), it is not easy to integrate devices from different technologies into a single network [31]. The lack of a common architecture may lead to additional costs due to the necessity of deploying non-transparent interconnection elements among the different networks [33]. Moreover, the developed elements are dependent on the application to which they belong, thus complicating their reutilization.

The main objective of the research presented in this paper is to design, build and deploy an innovative platform that addresses the requirements of Ambient Intelligence paradigm, such as context-awareness and ubiquitous communication, allowing the use of heterogeneous WSNs and taking advantage of the use of intelligent agents directly embedded on wireless nodes. In this sense, this paper describes the integration of the HERA (*Hardware-Embedded Reactive Agents*) platform into FUSION@ (*Flexible and User Services Oriented Multi-agent Architecture*) [48]. On the one hand, FUSION@ is an architecture aimed at facilitating the development of AmI-based systems with advanced context-aware capabilities. FUSION@ exploits the use of intelligent agents as the main components in employing a service-oriented approach, focusing on distributing the most of the systems' features into remote and local services and applications. On the other hand, HERA is an evolution of SYLPH *(Services laYers over Light PHysical devices)* [17]. SYLPH has the ability to use dynamic and self-adaptable heterogeneous WSNs and has been already tested to work with FUSION@ [1]. In HERA, agents are directly embedded into wireless nodes. This way, through the integration of HERA and FUSION@, there is no difference between a software and a hardware agent. That is, FUSION@ can run both software and hardware agents that offer services to other agents and applications, regardless of whether the agent is a piece of code or a wireless sensor. In effect, FUSION@ combines these two approaches and formalizes the integration of agents, services, communications and wireless technologies 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 AmI.

The next section presents the specific problem description that essentially motivated the development of FUSION@ and HERA and their integration. Section 3 describes the main characteristics of FUSION@ and HERA and briefly explains some of their components, including the integration of HERA into the FUSION@ architecture. Finally, Section 4 presents the results and conclusions obtained.

2. Motivation and related approaches

One of the key aspects for the construction of AmI-based systems is obtaining information about the people and their environment through sensor networks. We have developed several systems based on AmI [15,16,21,46]. However, these developments have caused significant problems when integrating sensors from different network technologies with agents, services and applications (e.g., hardware incompatibilities, reduced performance, etc.). This section discusses some of the most important problems of existing functional architectures, including their suitability for constructing intelligent environments according to the AmI paradigm. This section also presents the strengths and weaknesses of existing platforms and analyzes the feasibility of a new alternative: the integration of HERA into FUSION@.

Ambient Intelligence proposes three essential concepts: ubiquitous computing, ubiquitous communication and intelligent user interfaces [33]. AmI-based systems must be dynamic, flexible, robust, adaptable to changes in context, scalable and easy to use and maintain. In addition, dependability [3] and resilience [52] are attributes pursued by this kind of systems. The development of AmI-based systems that integrate different subsystems demands the creation of complex and flexible applications. As the complexity of an application increases, so does its need to be divided into modules with different functionalities. Since different applications could require similar functionalities, there is a trend towards the reutilization of resources that can be implemented as part of other systems. This trend is the best solution in the long-term and can be accomplished by using a common platform. However, it is difficult to carry out because the systems in which those functionalities are implemented are not always compatible with other systems.

An alternative to such an approach is the reimplementation of the required functionalities. Although it implies more development time, it is generally the easiest and safest solution. However, reimplementation can lead to duplicated functions and a more difficult system migration. A distributed architecture provides more flexible ways to move functions to

Author's personal copy

where actions are needed, thus obtaining better responses, autonomy, services continuity, and superior levels of flexibility and scalability than centralized architectures [29]. In addition, excessive centralization negatively affects system functionalities, overcharging or limiting their capabilities. For this reason, it is difficult for the system to dynamically adapt its behavior to the changes in the infrastructure. Thus, distributed architectures look for the interoperability among different systems, the distribution of resources and the independence of programming languages [29].

As said before, one of the most prevalent alternatives in distributed architectures is multi-agent systems. An agent can be defined as a computational system situated in an environment and able to act autonomously in this environment to achieve its design goals [50]. Expanding this definition, an agent is anything with the ability to perceive its environment through sensors, and to respond in the same environment through actuators, assuming that each agent may perceive its own actions and learn from the experience [37]. A multi-agent system is defined as any system composed of multiple autonomous agents incapable of solving a global problem, where there is no global control system, the data is decentralized and the computing is asynchronous [25,50]. Clearly, these definitions are closely related to Ambient Intelligence. There are several agent frameworks and platforms [7,32,45] that provide a wide range of tools for developing distributed multi-agent systems. The development of agents is an essential component 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 [34]. Multi-agent systems have been successfully applied to several Ambient Intelligence scenarios, such as education, culture, entertainment, medicine, commerce, industry, robotics, etc. [6,11,21,22,38–40,43,48]. Furthermore, agents can use reasoning mechanisms and methods in order to learn from past experiences and to adapt their behavior according to the context [5,16].

Nevertheless, multi-agent systems do not always cover the actual necessities of distributed systems. Thus, several developments consider the integration between agents and modern functional architectures, such as service-oriented architecture (SOA) [2,12,18]. These developments try to improve the distribution of the available resources, facilitate the reutilization of functionalities and optimize the compatibility among different platforms. In classic functional architectures, the modularity and structure are oriented towards the systems themselves. Modern functional architectures such as SOA allow functionalities to be created outside the system, as external services linked to the architecture. The term "service" can be defined as a mechanism that facilitates the access to one or more features (e.g., functions, network capabilities, etc.) [12]. Services are integrated through communication protocols that have to be used by applications to share resources in the network.

Another alternative to integrate subsystems in different environments is that known as Enterprise Application Integration (EAI) [27], widely used in business environments. The EAI consists of the use of software to integrate a set of applications and information taking into account basic principles of computer systems architectures. The success achieved by the effective and efficient application of the information technology and the knowledge engineering onto EAI has encouraged the analysis of the performance of this framework on other environments. However, its use on Ambient Intelligence scenarios makes it harder the integration of the distributed information throughout the environment. Even though there are many solutions provided by the EAI, most of them have failed from a practical point of view [27].

Any AmI-based scenario has to take into account the information about the context, which can be gathered by sensor networks. The context includes information about the people and their environment. The information may consist of many different parameters such as location, the building status (e.g., temperature), vital signs (e.g., heart rhythm), etc. Sensor networks need to be fast and easy to install and maintain. Each element that forms part of a sensor network is called a node. In an AmI scenario, nodes must communicate directly with one another in a distributed way [33]. In a centralized architecture, most of the intelligence is located in a central node. That is, the central node is responsible for managing most of the functionalities and knowing the existence of all nodes in a specific WSN. That means that a node belonging to a certain WSN does not know about the existence of another node forming part of a different WSN, even though this WSN is also part of the system. Nonetheless, this model can be improved using a common distributed architecture where all nodes in the system can know about the existence of any other node in the same system regardless of the technology or interface they use or the sub-network to which they belong.

2.1. Comparison of existing approaches that integrate multi-agent systems and WSNs

Both FUSION@ and HERA have stemmed from the necessity to more efficiently cover the challenges produced by the AmI-based systems. The fusion of the multi-agent technology and wireless sensor networks is not easy due to the difficulty in developing, debugging and testing distributed applications for devices with limited resources. The interfaces developed for these distributed applications are either too simple or, in some cases, do not even exist, which even further complicates their maintenance. Therefore, there are researches [49] that develop methodologies for the systematic development of multi-agent systems for WSNs. Some researches that relate multi-agent technologies with WSNs [28] insist that the combination of such technologies extends the life of wireless sensor nodes trough the reduction of the power consumption. However, these researches have not analyzed the capacities achieved by agents after analyzing the data obtained by WSNs. ActorNet [26] is other study that describes a mobile agent platform for WSNs. Implementing agent programs over WSNs is complicated due to the limitations of the sensor nodes, their limited memory, small bandwidth and low energy autonomy. ActorNet facilitates the development, providing an abstract environment for mobile code oriented to light objects over WSNs. ActorNet platform defines as its top layer an actor language interpreter. Likewise, the platform provides services such as virtual memory management and blocking input–output operations. Thus, ActorNet allows a wide range of dynamic

Author's personal copy

applications, including customized queries and aggregation functions, in the sensor network platform. However, each mobile agent is only centered on a sensor node. On the contrary, with the integration of HERA into FUSION@ the multi-agent system manages all data obtained by the WSNs to find effective and fast solutions. In addition, there are researches [41] that detail the design of practical applications over WSNs to control light, temperature and occupation levels in the physical environment. Sandhu [41] proposes the use of multi-agent systems to process information received by sensors, thus facilitating the decision-making process. Moreover, distributed learning through the multi-agent system provides a robust method of auto-configuration and adaption to the environment. Nevertheless, Sandhu [41] does not present any result or a detailed system evaluation. On the other hand, the integration of HERA into FUSION@ has produced some experiments with very promising results. Baker et al. [4] present the integration of an agent-based WSN within an existing MAS focused on condition monitoring. In this research, it is used SubSense, a multi-agent middleware platform developed to allow condition monitoring agents to be deployed onto a WSN. The architecture of the SubSense platform is based on the model defined by FIPA (Foundation for Intelligent Physical Agents), but customized so that agents are embedded into sensor nodes. SubSense platform is implemented over 512KB RAM SunSPOT sensor nodes using the Java Mobile Edition (J2ME). HERA platform can run on lightweight sensor nodes with just 8KB RAM. In addition, SubSense platform is not aimed to solve Ambient Intelligence challenges and it is not focused on working with heterogeneous WSNs. There are other studies [24] that supervise the transport through software agents, RFID (*Radio Frequency IDentification*) readers and WSNs. In these studies, software agents preprocess the information that they receive from sensors, and only communicate significant changes in the status of the freight to its owner. Jedermann et al. [24] present a system for tracking and monitoring elements applied to fruit logistics. In this case, software agents make autonomous decisions but do not have any language capacity or reasoning for improving their functioning. However, when integrating HERA into FUSION@, the agents coordinate their own functions and have learning and reasoning capacities that offer the best results and continually improve the given solutions. Furthermore, most of the works that relate multi-agent systems and WSNs talk about *Mobile Agents based on WSN* (MAWSN). For instance, the study presented by Qi and Wang [35] deal with planning routes to mobile agents based on WSN that consume the minimum amount of resources. The analysis of such a study is centered on the optimal design of the itinerary of the mobile agent. Fok et al. [20] also describe their system, Agilla, as a mobile agent that facilitates the fast development of applications on WSNs and apply it to fire tracking. Agilla allows users to create and integrate special programs called mobile agents that coordinate among themselves through local spaces and relate among themselves through WSNs to develop tasks specific to a given application. In a similar way, Zboril et al. [51] propose WSageNt, a platform that is implemented through mobile agents running on wireless sensor nodes. One key feature of this platform is a module for an agent control language interpretation. This language is presented as an original low-level control language known as Agent Low Level Language (ALLL). This research poses that in WSN-based agent platforms the resources limitations of sensor motes do not allow affording its development as an ordinary agent platform that should accomplish the FIPA specifications. Opposite to Agilla, WSageNt is supposed to be fault-tolerant and not to be only focused on WSNs. However, it has not context-awareness features and, as Agilla, does not seem to contemplate the interconnection of heterogeneous WSNs. These three studies [20,35,51] relating mobile agents and WSNs are very specific, while the integration of HERA into FUSION@ is not. The integration of HERA into FUSION@ produces a multi-agent architecture that relates a multi-agent system with WSNs, allowing it to be applied to different scopes and environments. Other studies, as this one described by Chen et al. [14] try to reduce the redundancy of the data gathered by sensors from different types of networks by using mobile agents that pack the data. This achieves faster data delivering and energy savings in the reception and delivery of the data. Other research presented by Chen et al. [13] proposes a MAWSN system that improves the communication of the nodes and mobile agents with regards to the client–server model. It also packs the data gathered by the sensor network, reduces the reception and delivery times and saves energy in devices with a low autonomy capacity. Along the lines of fusing or packing data gathered by sensors, Rajagopalan et al. [36] use a mobile routing agents' model to try to state the quality of the fused data and the cost of communication between the transmitter and the receiver. With their model, the authors explain the problem of multiple optimization goals, maximizing the detected signal energy and reducing the maximum energy consumption lost in the process. These three studies [13,14,36] focus on the fusing and optimizing the data communication process in a WSN through mobile agents.

The integration of HERA into FUSION@ provides a hybrid architecture that takes advantage of the capacities and particular features of each of them. Both FUSION@ and HERA are models that successfully solve the problems they set out to resolve. FUSION@ is a multi-agent architecture specially designed for developing AmI-based systems. FUSION@ is mainly centered on distributing systems functionalities onto applications and services, which are managed by a set of intelligent agents. HERA is a platform specially designed to implement hardware agents. Because HERA is based on SYLPH, it allows devices from different radio and networks technologies to coexist in the same distributed network. The combination of FUSION@ and HERA facilitates and speeds up the integration between agents and sensors for reusing resources in the context. This approach allows the development of multi-agent systems with increased scalability. It also expands the agents' capabilities to obtain information about the context and to automatically react over the environment. A totally distributed approach and the use of heterogeneous WSNs provides an architecture that is better capable of recovering from errors, and more flexible to adjust its behavior in execution time.

A comparison between some of these approaches that integrate multi-agent systems and wireless sensor networks is depicted in Table 1. As can be seen, one of the main contribution of the platforms based on SYLPH, such as the integration of SYLPH with FUSION@, the HERA platform and the new integration of HERA into FUSION@ presented in this paper, is that these allow the integration of multi-agent systems with heterogeneous WSNs. Other approaches do not take into account

Author's personal copy

Table 1
Comparison between existing approaches focused on integrating multi-agent systems and wireless sensor networks.

	Heterogeneous WSNs	Hardware-embedded agents	Context-awareness	AmI-oriented
actorNet	No	Yes	Partly	No
SubSense	No	Yes	Yes	No
Agilla	No	Yes	Partly	No
WSageNt	No	Yes	No	No
FUSION@ + SYLPH	Yes	No	Yes	Yes
HERA (stand-alone)	Yes	Yes	Yes	Yes
FUSION@ + HERA	Yes	Both	Yes	Yes

the use of such heterogeneous WSNs and they are focused on working with sensor nodes that use just an only radio technology. Furthermore, in the design of both FUSION@ and HERA, they have been mainly aimed to address the necessities of Ambient Intelligence, such as context-awareness and ubiquitous computing, while other existing approaches are not specially centered on dealing with these requirements. The real implementation of hardware-embedded agents into heterogeneous wireless sensor networks focused on solving the necessities that Ambient Intelligence systems require is which make the integration of HERA into FUSION@ so innovative, opposite to other approaches. HERA has been designed to have agents directly embedded into sensor nodes, giving a step over SYLPH. However, the integration of HERA into FUSION@ provides an architecture where software and hardware-embedded agents can interact each other, not being distinction between them, while other existing approaches do not take this feature into account.

3. FUSION@ and HERA

This section presents the integration of the new HERA (*Hardware-Embedded Reactive Agents*) platform into FUSION@ (*Flexible and User Services Oriented Multi-agent Architecture*). First, the FUSION@ Architecture and its components and agents are described. Next, the new HERA platform and the WSN-SOA platform on which it is based, SYLPH (*Services laYers over Light PHysical devices*), are explained. And finally, after presenting former approaches of combining SYLPH and FUSION@, the integration of HERA into FUSION@ is finally described.

3.1. The FUSION@ architecture

FUSION@ is a SOA-based [47] multi-agent architecture that facilitates the integration of distributed multi-agent systems [48]. The FUSION@ model has also been designed to develop systems based on the Ambient Intelligence paradigm. Multi-agent systems help to distribute resources and reduce the central unit tasks [2]. There are several agent frameworks and platforms, such as Open Agent Architecture (OAA) [32], RETSINA [45] and JADE [7], which provide a wide range of tools for developing distributed multi-agent systems. FUSION@ exceeds these architectures and frameworks by adding new layers, and facilitating the distribution and management of resources (i.e., services). A distributed agent-based architecture provides more flexible ways of moving 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. FUSION@ can be used to develop any kind of complex systems because it is capable of integrating almost any service and application desired, without depending on any specific programming language. Because the architecture acts as an interpreter, the users can run applications and services that can be programmed in virtually any language, but have to follow a communication protocol that all applications and services must incorporate. Another important functionality is that, because of the agents' capabilities, the newly developed systems can use 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 regardless of their location. It also makes is possible to start or stop agents, applications, services or devices separately, without affecting the other resources, providing the system with an elevated adaptability and capacity for error recovery.

FUSION@ sits on top of existing agent frameworks by adding new layers to integrate a service-oriented approach and facilitate the distribution and management of resources. Therefore, the FUSION@ framework was modeled after the SOA model, but has added the applications blocks that represent the interaction with users. These blocks provide all the functionalities of the architecture. FUSION@ adds new features to common agent frameworks, such as OAA, RETSINA and JADE and improves the services provided by these previous architectures. These other architectures have limited communication abilities and are not compatible with SOA architectures. Fig. 1 shows the UML deployment diagram of FUSION@, where can be seen the four basic blocks of FUSION@:

– **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 invoked services. They can be executed locally or remotely, even on mobile devices with limited processing capabilities, because computing tasks are primarily delegated to the agents and services.

Author's personal copy

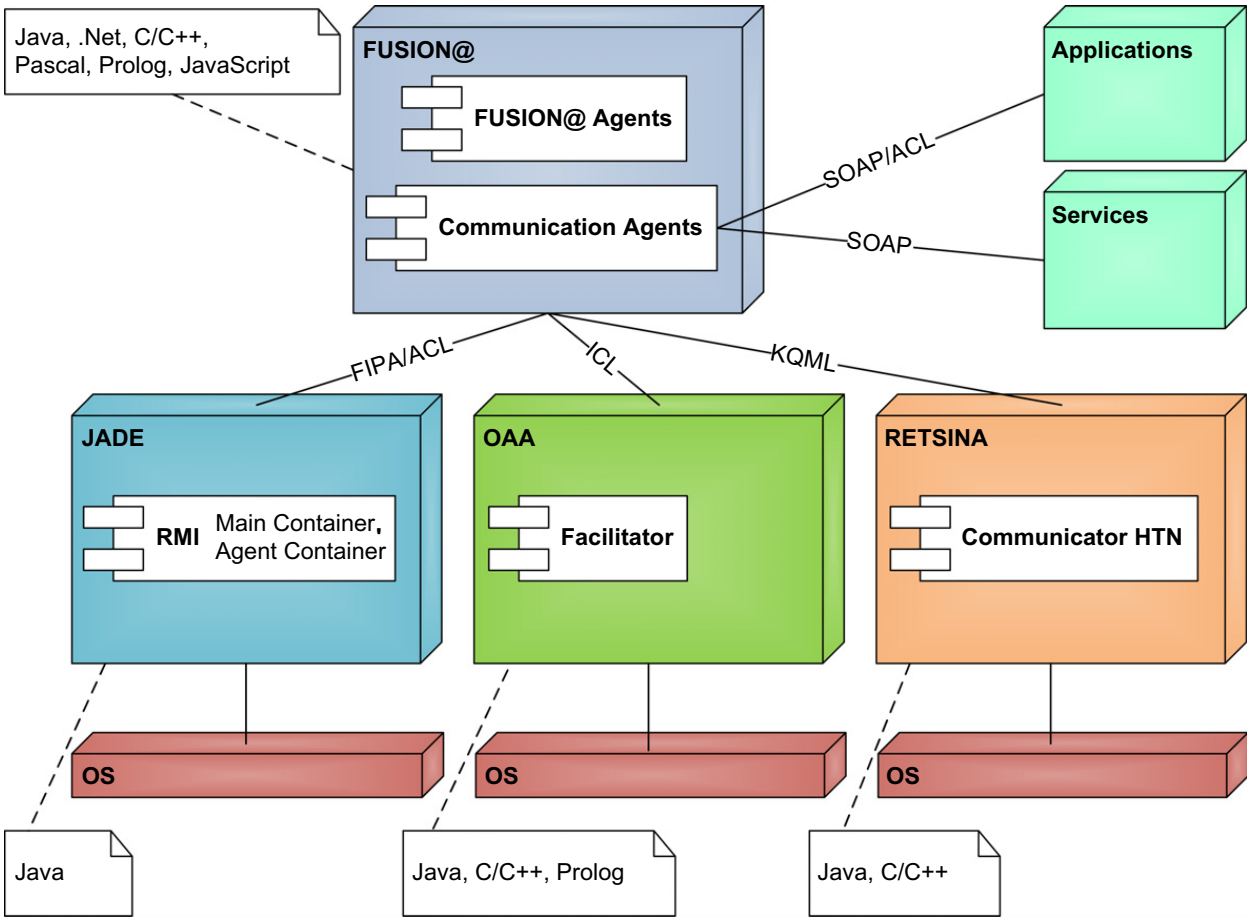


Fig. 1. UML deployment diagram of the basic schema of the FUSION@ architecture. The association between FUSION@ and each agent platform is defined according to its respective communication protocol. In a similar way, communication protocols are also used to associate the FUSION@ architecture itself with both services and applications.

– **Agent platform.** This is the core of FUSION@, integrating a set of agents, each one with special characteristics and behaviors. 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. 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. 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 functions that users require. FUSION@ has a flexible and scalable directory of services that 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 agent 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 [18]. Services and applications communicate with the agent 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 amongst 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 use agent 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, PDAs). All requests and responses are handled by the

Author's personal copy

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.

FUSION@ is a modular multi-agent architecture, where services and applications are managed and controlled by deliberative BDI (Belief, Desire, Intention) agents [9,50]. 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, and intentions, and plan representations to solve problems [9]. Deliberative BDI agents are the core of FUSION@. Therefore 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 to incorporate new agents. However, there are pre-defined agents that 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. The *CommApp Agent* is always on "listening mode". Applications send XML messages to the agent requesting a service; this agent then starts communication by using sockets to create a new thread. 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 the *Security Agent* for their structure and syntax to be analyzed.

– **CommServ Agent.** It is responsible for all communications between services and the platform. Its functionalities are similar to *CommApp Agent* but in reverse. This agent is always on "listening mode", waiting for responses from services. The *Admin Agent* indicates to the *CommServ Agent* which service must be invoked. Then, the *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 the *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, the list of services is static and can only be manually modified; however, services can be dynamically added, erased or modified. The list contains the information of all trusted available services. The name and description of the service, the required parameters 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 (QoS). 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 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 assigned tasks. Information management is especially important in environments where the data processed is very sensitive and personal (e.g., healthcare applications). 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 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 is undeliverable. This agent also directs the problem to the *Directory Agent*, which modifies the QoS of the service where the message was sent.

– **Admin Agent.** The *Admin Agent* decides which service must be called by taking into account the QoS and users' preferences. Users can explicitly invoke a service, or they can let the *Admin Agent* decide which service is the best for accomplishing 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 a 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, the *CommServ Agent* informs the *Admin Agent*, which tries to find an alternative service, and informs the *Directory Agent* to modify the respective QoS. Furthermore, the *Admin Agent* is responsible for checking if the *Supervisor Agent* is alive. If not, the *Admin Agent* initiates a new instance of the *Supervisor Agent* so that the system can better recover from errors.

– **Interface Agent.** This kind of agent was designed to be embedded in user 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 directly sent to the *CommApp Agent*, which forwards them to the *Security Agent* so that it analyzes the requests. If they are correct, the *CommServ Agent* 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.

FUSION@ is an open architecture that allows developers to modify the structure of the agents, previously described, so that they are not defined in a static manner. Developers can add new agent types or extend the existing ones to conform

Author's personal copy

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 both 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. 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 by 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 that include managing multiple and simultaneous tasks, so services must be programmed to allow multithreading. However, there could be situations where multi-tasks are 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 it has finished.

3.2. The HERA platform

HERA (*Hardware-Embedded Reactive Agents*) facilitates agents, applications and services communication in FUSION@ through the use of dynamic and self-adaptable heterogeneous WSNs. There have been several attempts to integrate WSNs and a SOA approach [28,44,49]. Unlike those approaches, the agents in HERA are directly embedded on the WSN nodes and their services can be invoked from other nodes in the same network or another network connected to the original one. HERA is an evolution of SYLPH (*Services laYers over Light PHysical devices*) [17]. The SYLPH platform follows a SOA model [12] for integrating heterogeneous WSNs in AmI-based systems. As SYLPH, HERA focuses specifically on devices with small resources in order to save CPU time, memory size and energy consumption. In SYLPH, services are directly offered by the wireless sensor nodes that are part of the platform. In the same way, any node in the platform can directly invoke a SYLPH service offered by other node in the platform, no matter if both nodes are in the same physical wireless network or not.

Each element that forms part of a sensor network is called a node. In an AmI scenario, nodes must communicate directly with one another in a distributed way. In a centralized architecture, most of the intelligence is located in a central node. That is, the central node is responsible for managing most of the functionalities and knowing the existence of all nodes in a specific WSN. That means that a node belonging to a certain WSN does not know about the existence of another node forming part of a different WSN, even though this WSN is also part of the system. For this reason, it is difficult for the system to dynamically adapt its behavior to the changes in the infrastructure. In addition, excessive centralization negatively affects system functionalities, overcharging or limiting their capabilities. Nonetheless, this model can be improved using a common distributed architecture where all nodes in the system can know about the existence of any other node in the same system regardless of the technology or interface they use or the sub-network to which they belong. This can be achieved by adding a middleware logical layer over the existing application layers on the nodes. A distributed architecture provides more flexible ways to move functions to where actions are needed, thus obtaining better responses, autonomy, services continuity, and superior levels of flexibility and scalability than centralized architectures.

The HERA platform has been designed to enable an extensive integration of WSNs and optimize the distribution, management and reutilization of the available resources and functionalities in its networks. As a result of SYLPH, HERA provides the possibility of connecting wireless sensor networks based on different radio and link technologies, whereas other approaches do not. That is, HERA allows the agents embedded into nodes to work in a distributed way and does not depend on the lower stack layers related to the WSN formation (i.e., network layer) or the radio transmission among the nodes that form part of the network (i.e., data link and physical layers). Likewise, HERA can be executed over multiple wireless devices independently of their microcontroller or the programming language they use.

In SYLPH, the main objective is to distribute resources over multiple WSNs by modeling the functionalities as independent services. As explained below, the information gathered by SYLPH nodes can be managed by intelligent agents by using the integration of SYLPH with the FUSION@ multi-agent architecture. Thus, the agents running on FUSION@ can use reasoning mechanisms to adapt their behavior to the context information obtained through SYLPH nodes. However, HERA goes a step further than SYLPH. In HERA, agents are directly embedded on the wireless nodes. Therefore, each wireless node is an agent and works inside FUSION@ as another software agent running on platforms such as RETSINA or JADE. This way, through the integration of HERA into FUSION@, there is no difference between a software agent and a hardware agent.

SYLPH covers aspects related to services such as registration, discovering and addressing. Additionally, a node can invoke functionalities offered by any other node in the system, regardless of whether they are in the same WSN or not. Some nodes in the system can integrate service directories for distributing registration and discovering services. Node registration is done in the corresponding WSN (i.e., specific network) and service registration is maintained by multiple services directories. Thus, the process of connecting new nodes to the system is performed in a dynamic way. A node can know about the existence of other nodes and the services they offer. Therefore, it can directly communicate with other nodes to perform

Author's personal copy

a specific service. A SOA model was chosen in SYLPH because architectures based on this model are asynchronous and non-dependent on context (i.e., previous states of the system, which must not be confused with context-aware environments) [12]. Thus, devices working on them do not continuously take up processing time, consume less energy and are free to perform other tasks.

SYLPH can be executed over multiple wireless devices independently of their microcontroller or the programming language they use. SYLPH works in a distributed way so that the application code does not have to reside almost completely on a single central node. SYLPH allows the interconnection of several networks from different wireless technologies, such as ZigBee or Bluetooth. Thus, a node designed over a specific technology can be connected to a node from a different technology. In this case, both WSNs are interconnected by a set of intermediate gateways simultaneously connected to several wireless interfaces. SYLPH allows applications to work in a distributed way and independently of the lower layers related to the WSNs formation (i.e., network layer) and the radio transmission among the nodes that comprise them (i.e., data link and physical layers). The services can be executed from multiple wireless devices. Given that neither developers nor users have to worry about what kind of technology each node in the system uses, the experience is transparent for everybody involved. This facilitates the inclusion of context-aware capabilities into AmI-based systems because developers can dynamically integrate and remove nodes on demand.

3.2.1. Layers and components of the HERA platform

SYLPH implements an organization based on a stack of layers [42]. Each layer in one node communicates with its peer in another node through an established protocol. In addition, each layer offers specific functionalities to the immediately upper layer in the stack. These functionalities are usually called *interlayer services*, which must not be confused with the services invoked from node to node. These interlayer services are abstract functions and independent of the implementation of the platform. The SYLPH layers are added over the existent application layer of each WSN stack, allowing the platform to be reutilized over different technologies. Fig. 2 shows the SYLPH layers and protocol stacks over two ZigBee nodes through an UML deployment diagram. The structure of SYLPH will now be described.

– **SYLPH Message Layer (SML).** The SML offers the upper layers the possibility of sending asynchronous messages between two nodes through the SYLPH Services Protocol (SSP). These messages specify the source and destination nodes and the service invocation in a SYLPH Services Definition Language (SSDL) format. The SSDL describes the service itself and the parameters to be invoked. This SML not only transports the services invocations over the network, but also the services registration and search functions.

– **SYLPH Application Layer (SAL).** The SAL allows different nodes to directly communicate with each other using SSDL requests and responses that will be delivered in encapsulated SML messages following the SSP. SAL implements the service code (i.e., firmware) from within each node, allowing each one to communicate with the SYLPH platform and invoke services located in other nodes. Moreover, there are other interlayer services for registering services or finding services offered by other nodes. In fact, these interlayer services for registering and searching services call other interlayer services offered by the SYLPH Services Directory Sub-layer (SSDS). Therefore, the SAL can use the interlayer services of the SML either directly or through the SSDS.

– **SYLPH Services Protocol (SSP).** The SSP is the internetworking protocol of the SYLPH platform. SSP has functionalities similar to those of the Internet Protocol (IP). That is, it allows sending packets of data from one node to another node regardless of the WSN to which each one belongs. Every node has a unique SSP 32-bit address in the SYLPH network. Therefore, an SSP packet includes a header that describes the SSP addresses of the source node and the destination node, as well as information for managing transmissions that involve multiple SSP packets (i.e., number of SSP packet and remaining bytes).

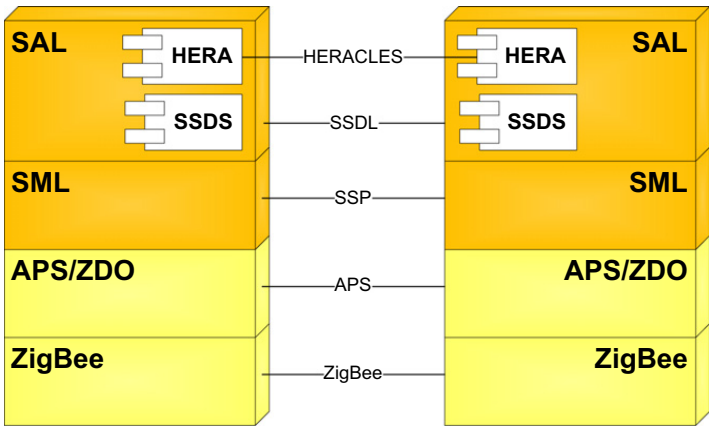


Fig. 2. UML deployment diagram of the HERA platform running over SYLPH. This diagram shows the HERA Agents layer running as a component on the SYLPH Application Layer. Likewise, it can be seen the different layers of SYLPH and HERA, as well as the different protocols that communicate each layer on different ZigBee nodes.

Author's personal copy

– **SYLPH Services Definition Language (SSDL).** The SSDL is the IDL (Interface Definition Language) used by SYLPH. Unlike other IDLs such as WSDL (*Web Services Definition Language*) [18], SSDL does not use as many intermediate separating tags, and the order of its elements is fixed. SSDL has been specifically designed to work with limited computational resources nodes. Nodes can request the SSDS for the location of services and their specifications using SSDL.
– **SYLPH Services Directory Sub-layer (SSDS).** The SSDS creates dynamic services tables to locate and register services in the network. A node that stores and maintains services tables is called SYLPH Directory Node (SDN). These tables are made up of a list of service entries, each of which includes the description of a service in SSDL format and the SSP address of the node that offers the service. In addition, each entry stores additional information about the service whose location and description is maintained in the network. Such information includes, for instance, a Quality of Service (QoS) rate and the last time the SDN checked if the service was available. A node in the network can make a request to the SDN to know the location (i.e., network address) of a certain service. Requests are packed in SML messages and must follow the SSP. SSDS is also used by the SAL when registering a new service.

The HERA Agent platform adds its own agents layer over the SYLPH stack of layers, as shown in Fig. 2. As the HERA platform is based on the existing layers of SYLPH platform, the applications and systems built over HERA take advantage of one of the main features of SYLPH: it can be run over any wireless sensor node regardless of its radio technology or the programming language used for development. In addition, as HERA is designed over SYLPH communication layers, HERA Agents running on WSNs with different radio technology can communicate among themselves through one or more SYLPH Gateways, as explained below. Consequently, the main components added by HERA to the SYLPH's stack of layers are:

– **The HERA Agents Layer (or just HERA).** HERA Agents are specifically intended to run on devices with reduced resources, precisely what SYLPH was designed for. To communicate with each other, HERA Agents use HERACLES, the agent communication language designed for being used under the HERA platform. Each HERA Agent is an intelligent piece of code running over the SYLPH Application Layer. As explained below, there must be at least one *facilitator agent* in every agent platform. This agent is the first created in the platform and acts as a directory for searching agents. In HERA, the equivalent of these agents is the HERA-SDN (*HERA Spanned Directory Node*).
– **HERA Communication Language Emphasized to Simplicity (HERACLES).** The HERACLES language is directly based on the SSDL language. As with SSDL, HERACLES does not use intermediate tags and the order of its elements is fixed to constrain the resource necessities of the nodes. This makes its human-readable representation, used by developers for coding, very similar to SSDL. When HERACLES is translated to HERACLES frames, the actual data transmitted among nodes, they are encapsulated into simple SSDL frames using "HERA" as their *service id* field.

3.2.2. HERA basic operation and HERA Spanned Directory Nodes (HERA-SDNs)

The behavior of SYLPH is essentially similar to that of any other service oriented architecture. However, SYLPH has several characteristics and functionalities that make it different from other models. First, a service registers itself on the SDN and informs the network of its location, the parameters it requires, and the type of returned value after its execution. In order to do this, the service uses SSDL, which was created to work with limited resources nodes. Once the service has been registered in the SDN, it can be invoked by any application using SYLPH. Both the SDN and the services can be stored in any node of the WSN or in other subsystem connected to the WSN. This system can be, for instance, a simple personal computer connected through a USB port to a wireless interface. Thus, developers decide which nodes or subsystems will implement each part of the distributed application. Any node in the network can ask the SDN for the location of a particular service and its specification using SSDL. Because the aim of the architecture is to be as distributed as possible, it is possible to have more than one SDN in the same network, which makes it possible for redundancies to exist or services to be organized in different directories. The SDN can be stored in one of the network nodes, with a memory external to the microcontroller if necessary, or it can be contained on a computationally higher machine connected to the WSN, as is the case of a data server or a personal computer with wireless connection.

The UML sequence diagram depicted in Fig. 3a shows how a node can discover services in the network. For example, SYLPH node #1, belonging to WSN "A" registers itself in the SYLPH platform. For this, it sends a broadcast message searching for existing SDNs in the network. At this moment, only SDN #0 is active, so after receiving the broadcast message it sends a message to node #1 informing of its situation (i.e., SSP address) and its setup parameters. An example of a setup parameter is whether the SDN will periodically inform of its presence or if the nodes themselves have to ask it. After this, node #1 is able to communicate with SDN #0 in order to obtain information about the possible services existing in the network. Later, node #2 registers itself on the platform. It belongs to a different WSN, called WSN "B" and perhaps uses a radio technology different than that used by WSN "A". This is possible because of the SYLPH Gateways, described below. As node #2 has SDN functionalities, it informs the rest of the nodes with a broadcast message. SDN #1 stores this information on its SSDS entry list and informs node #2 about its role as SDN. Any node in the network can not only offer or invoke SYLPH services, but also include SDN functionalities in order to provide services descriptions to other network nodes. SDNs include additional information about services, for example, a Quality of Service rate and a timestamp that represents the last time the SDN checked if the service was available. A SDN can be configured to check the services periodically or to allow service broadcasts.

Author's personal copy

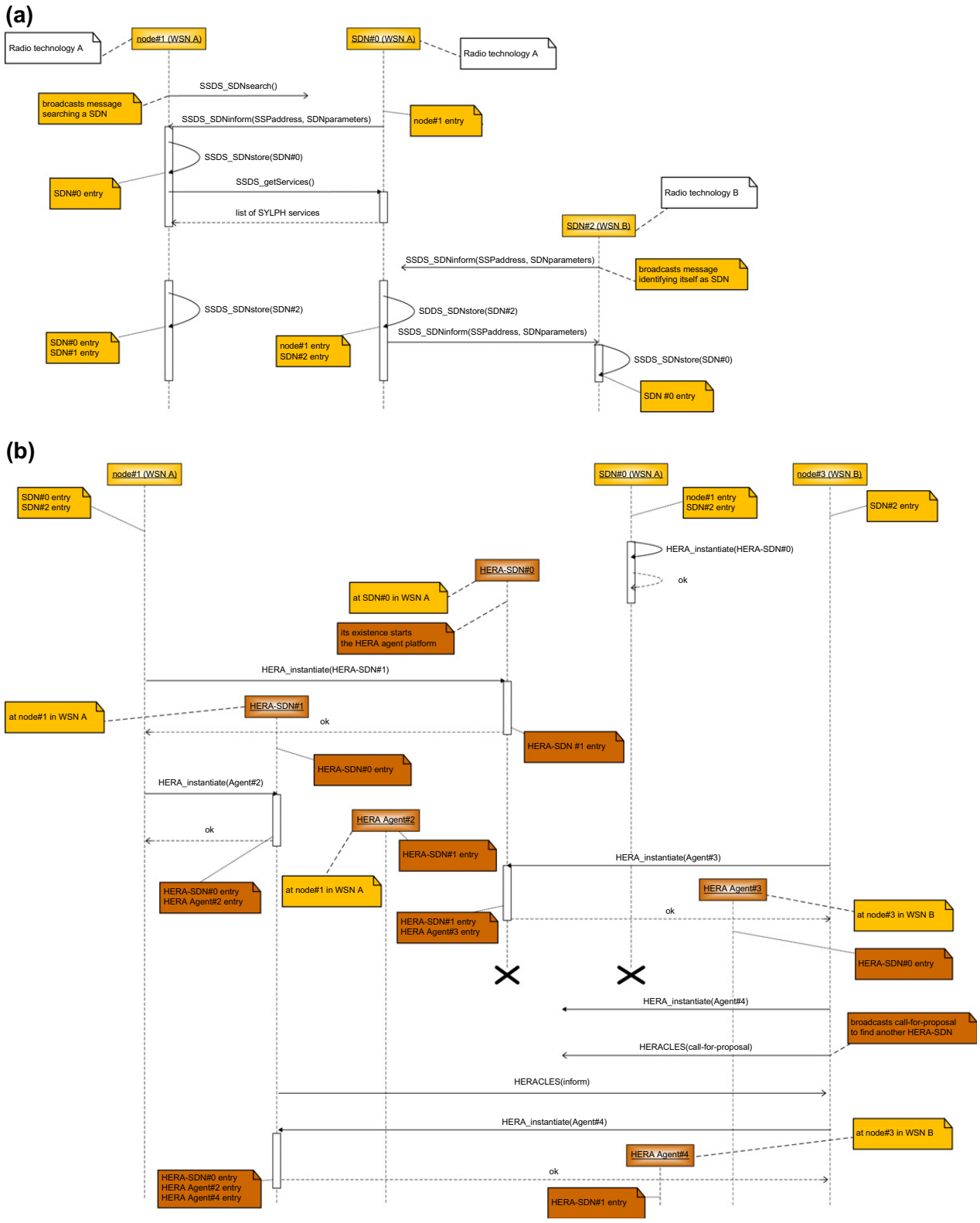


Fig. 3. UML sequence diagrams showing examples of the basic operation of SYLPH and HERA platforms when registering services or agents on the SYLPH Directory Nodes (a) and the HERA Spanned Directory Nodes (b), respectively.

Every agent platform needs some kind of *facilitator agent* that needs to be created before other agents are instantiated in the platform [7,32,45]. Facilitator agents act as agent directories. This way, every time an agent is created, it is registered on one of the existing facilitator agents. This allows other agents to request one of the facilitator agents in order to know where an agent with certain functionalities is and how to invoke such functionalities. As HERA is intended to run on machines that are not more complex than sensor nodes themselves are, it was necessary to design some hardware facilitator agents that do

Author's personal copy

not need more CPU complexity and memory size than what a regular sensor node has. In order to do this, HERA's facilitator agents, called HERA-SDNs (HERA Spanned Directory Nodes) are based on the SYLPH Directory Nodes (SDN), described above. This way, any HERA node can perform as a HERA-SDN, just as SDNs do in the SYLPH platform. However, a HERA-SDN does not also have to be a SDN. HERA-SDN instances itself and starts the HERA platform by registering a special SYLPH service called "HERA" on a SDN stored on any node of the SYLPH network. When a new HERA Agent wants to instantiate itself through a HERA-SDN, it looks for the "HERA" service on the SYLPH network, using a primitive service of the SSDL/SSP layers. When a HERA Agent is correctly instantiated, the HERA layer also registers a "HERA" service for the agent in a SDN. In this way HERA Agents can send HERACLES messages to each other over SYLPH, referring to the service of each node with HERA Agents, including HERA-SDNs, as "HERA".

Fig. 3b shows the basic operation of HERA Agents and HERA-SDNs through another UML sequence diagram. In order to start the HERA platform, an initial HERA-SDN must be created. This will be HERA-SDN #0 running on SDN #0. At that moment, other SYLPH nodes with HERA running on them can instantiate more HERA Agents or even more HERA-SDNs. Because HERA is designed to run on devices with low resources that are usually connected wirelessly, it is very important that the platform does not have to depend on only one HERA-SDN (i.e., one facilitator agent). This way, if the HERA-SDN crashes (e.g., power failure or problems with radio transmission), the HERA platform will not fail and will not need to be started again. After the creation of the HERA-SDN #0, the SYLPH node #1 uses the HERA-SDN #0 to instantiate a new HERA-SDN, the HERA-SDN #1, thus increasing the redundancy of the HERA-SDNs and the robustness of the platform. The SYLPH node #1 also instantiates the HERA Agent #2, this time through the HERA-SDN #1. SYLPH node #3, in the other WSN, instantiates HERA Agent #3 through the HERA-SDN #0, even if they are in distinct WSNs. With SYLPH, this is no longer a problem. At a specific moment, SDN #0 is powered off. After that, SYLPH node #3 looks for HERA-SDN #0. As HERA-SDN #0 does not reply, SYLPH node #3 sends a broadcast *call-for-proposal* HERACLES frame in order to find a live HERA-SDN. As HERA-SDN #1 replies, HERA Agent #4 is created through HERA-SDN #1. As shown in Fig. 3b, there can be several HERA Agents in a single SYLPH node. Moreover, there can be SYLPH nodes with no HERA implementation. A SYLPH Gateway is a clear example of this, as explained below. If, at a certain moment, HERA Agent #2 wants to look for an agent, but HERA-SDN #0 is not alive again, then HERA Agent #2 also has to look for an existing HERA-SDN in the platform, thus storing entries for the two HERA-SDNs. When HERA-SDN #0 is alive again, it will be useful for HERA Agent #2 to have this redundancy on HERA-SDNs entries.

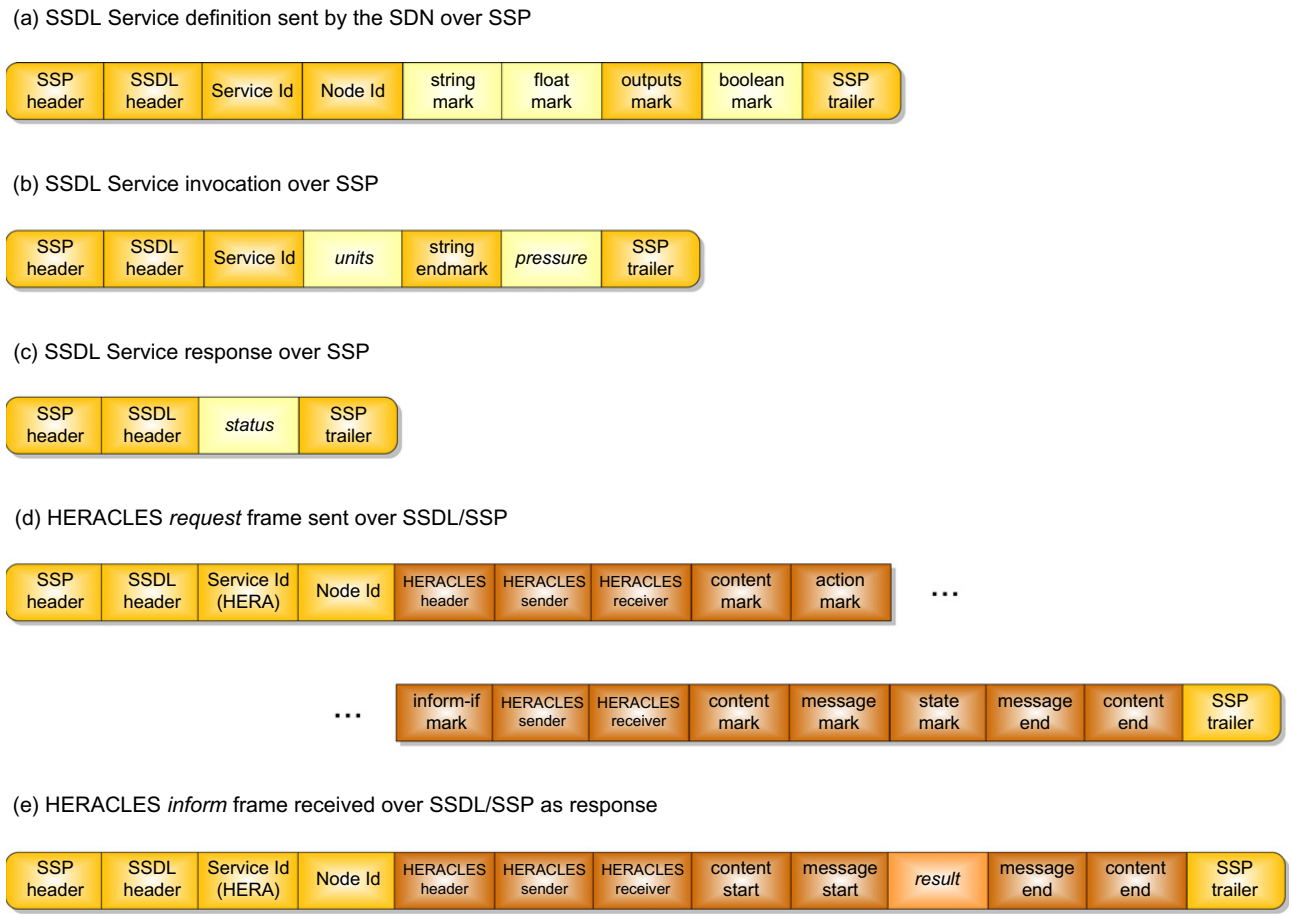


Fig. 4. Examples of SYLPH's SSDL frames over SSP and HERA's HERACLES frames over SSP/SSDL.

Author's personal copy

3.2.3. The HERA Communication Language Emphasized to Simplicity (HERACLES)

SSDL (SYLPH Services Definition Language) is the IDL (Interface Definition Language) used by SYLPH. Unlike other IDLs such as WSDL (*Web Services Definition Language*) [18], SSDL does not use as many intermediate separating tags, and the order of its elements is fixed. SSDL has been specifically designed to work with limited computational resource nodes. Nodes can request the SSDS for the location of services and their specifications using SSDL. The reason for these constraints is to reduce processing in the devices microcontrollers. Consequently, using a simple IDL makes it possible to use nodes with fewer resources, less power consumption and at a lower cost. In most cases a few float point data for informing the status of a sensor is sufficient. Thus, most service definitions require only a few bytes. SSDL considers the basic types of data (e.g., integer, float or Boolean), allowing the use of more complex data structures, such as variable length arrays or character strings. This makes SSDL flexible enough to specify more complex services if required.

In the List 1 in the appendix we demonstrate a simple example of the use of SSDL in order to define a SYLPH service. This representation of SSDL is the human-readable version, used by developers when defining services, not the version actually transmitted. It is clear that its syntax is slightly similar to that of C language. Assembler and C are the most used programming languages for coding microcontroller firmware. However, C language is more human-readable than assembler. The example defines a simple service called `registerServiceOnFireAlarm`. This service is stored in a sensor device that belongs to a wireless network with the SYLPH platform running over it.

In fact, this representation of the SSDL syntax is the one used by developers to specify the services in the firmware running on the devices attached to SYLPH architecture. After specifying the service by means of SSDL human-readable syntax, developers translate definitions to specific code for the target language (e.g., C or nesC) and the microcontroller where the service will run. When the node registers its service in a SDN, SYLPH layers do not transmit the human-readable SSDL message, but a more compact array of bytes that describe the service and how to invoke it from other nodes. Fig. 4 shows the SSDL frames involved in the `registerServiceOnFireAlarm` service definition (a), invocation (b) and response (c) when transmitted over SSP. When a node asks a SDN for the service definition, the SDN answers with a frame as shown in Fig. 4a. This frame describes the service identification, the address of the node that stores the service, the definition of the input and output parameters, and the QoS offered by the service. It has a SSDL header that specifies the SSDL data length and the type of frame it is (registration, definition, invocation or response). We can see that there is an *outputs mark* that denotes the input parameters and the output parameters that follow it. In the example of the `registerServiceOnFire-Alarm` service, a service *callback* is described as an input parameter. To specify that issue, there are some *service start* and *end marks*. As the service callback has no input parameters, the *outputs mark* is the first field inside its definition in the parameters description. Once the invoker node knows the service definition, it can call the service by sending a SSP frame to the node that stores the service. This frame (Fig. 4b) does not need any mark inside it, because the input parameters have to follow the specified order. Thus, the SSDL combines ease of parsing with flexibility in the type and size of the used parameters. The SSP header includes the SSP address of the destination node. The response frame contains only the output parameter (Fig. 4c).

In HERA, the hardware agents communicate with each other through the *HERA Communication Language Emphasized to Simplicity* (HERACLES). This language is an extension of the SSDL used in SYLPH. As explained above, SSDL has two distinct representations [17]: one that is human-readable, similar to C language and used for services development proposals, and one embedded on frames that SYLPH nodes understand. This is done in this way because in nodes with reduced resources (memory and CPU time) it is not convenient to overload the microcontroller and the memory space with a heavy parsing method. When developing a program, programmers use the human-readable representation to define agents' functionalities, similar to that shown on List 2 in the appendix.

However, similar to SYLPH, HERA Agents transmit the more compact representation of HERACLES as frames. The compact frames corresponding to the previous example are also represented in Fig. 4. This kind of compact frames is what HERA Agents transmit in a heterogeneous WSN based on HERA-SYLPH over the SSDL/SSP protocols. Fig. 4d shows the frame corresponding to the HERACLES *request*, whereas Fig. 4e depicts the frame corresponding to the HERACLES *inform* of the previous example.

3.2.4. Operation of HERA over heterogeneous WSNs using SYLPH Gateways

As previously mentioned, with SYLPH, a node in a specific type of WSN (e.g., ZigBee) can directly communicate with a node in another type of WSN (e.g., Bluetooth). Therefore, several heterogeneous WSNs can be interconnected through a SYLPH Gateway. A SYLPH Gateway is a device with several hardware network interfaces (e.g., a Wi-Fi network card), each of which is connected to a distinct WSN. As with an IP gateway, a SYLPH Gateway does not need to implement the layers over the SML. The SYLPH Gateway stores routing tables for forwarding SSP packets among the different WSNs with which it is interconnected. The information transported in the SSP header is enough to route the packets to the corresponding WSN. If several WSNs belong to the SYLPH network, there is no difference between invoking a service stored either in a node in the same WSN or in a node in a different WSN. For example, if a source node invokes a service stored in a destination node located in a different WSN, the source node looks for the service in a SDN present in the WSN to which it belongs. In fact, the entry stored in the services table of that SDN points to the SSP address of the SYLPH Gateway. When the source node invokes the service in the destination node, the SYLPH Gateway forwards the call message to the destination node through its hardware interface connected to the WSN where the destination node is located. Fig. 5 shows an UML deployment diagram representing a ZigBee and a Bluetooth network working together using SYLPH.

Author's personal copy

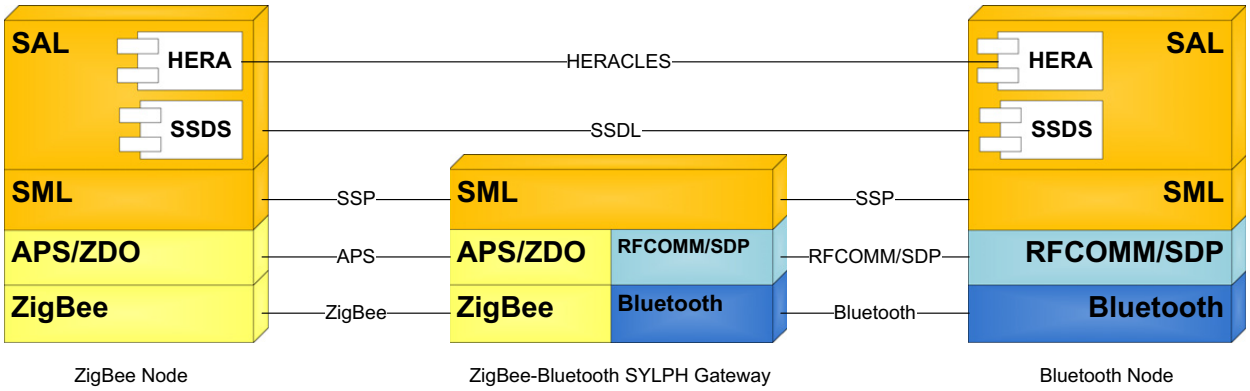


Fig. 5. UML deployment diagram of SYLPH and HERA over a ZigBee and a Bluetooth network through a SYLPH Gateway. As can be seen, HERA Agents on nodes from different radio technologies communicate each other in a transparent way thanks to SYLPH.

Because of HERA is implemented over SYLPH through the addition of new layers and protocols (HERA Agents and HERACLES), it can be used over several heterogeneous WSNs in a transparent way. HERA Agents are implemented over the SAL layer, so HERA does not mind how many intermediate SYLPH Gateways and different WSNs there are between the location of one HERA Agent and another. This is demonstrated in Fig. 5. Both HERA Agents and HERA-SDNs communicate with each other directly through HERACLES. HERA Agents use SAL's service points to deliver HERACLES frames between agents. Since HERACLES frames are transported as other SSDL frames over SSP between SYLPH nodes, HERA Agents do not need to know which nodes other HERA Agents are stored on, or if such nodes are in remote WSNs.

3.3. Integration of HERA into FUSION@

Before the development of HERA, the use of FUSION@ was proposed [1] to interact with a SYLPH network from a system that is not made up of WSNs. Consequently, two agents in SYLPH were designed to interact with FUSION@: *SylphInterface Agent* and *SylphMonitor Agent*. The *SylphInterface Agent* allows the rest of the agents to discover and invoke services offered by SYLPH WSN nodes. Moreover, the *SylphInterface Agent* can offer services to the wireless nodes. In order to do this, the WSN node in the FUSION@–SYLPH Gateway stores service entries on its SSDS table. As shown in the UML deployment diagram depicted in Fig. 6, the *SylphInterface Agent* performs as a broker between the SYLPH network and the FUSION@ architecture.

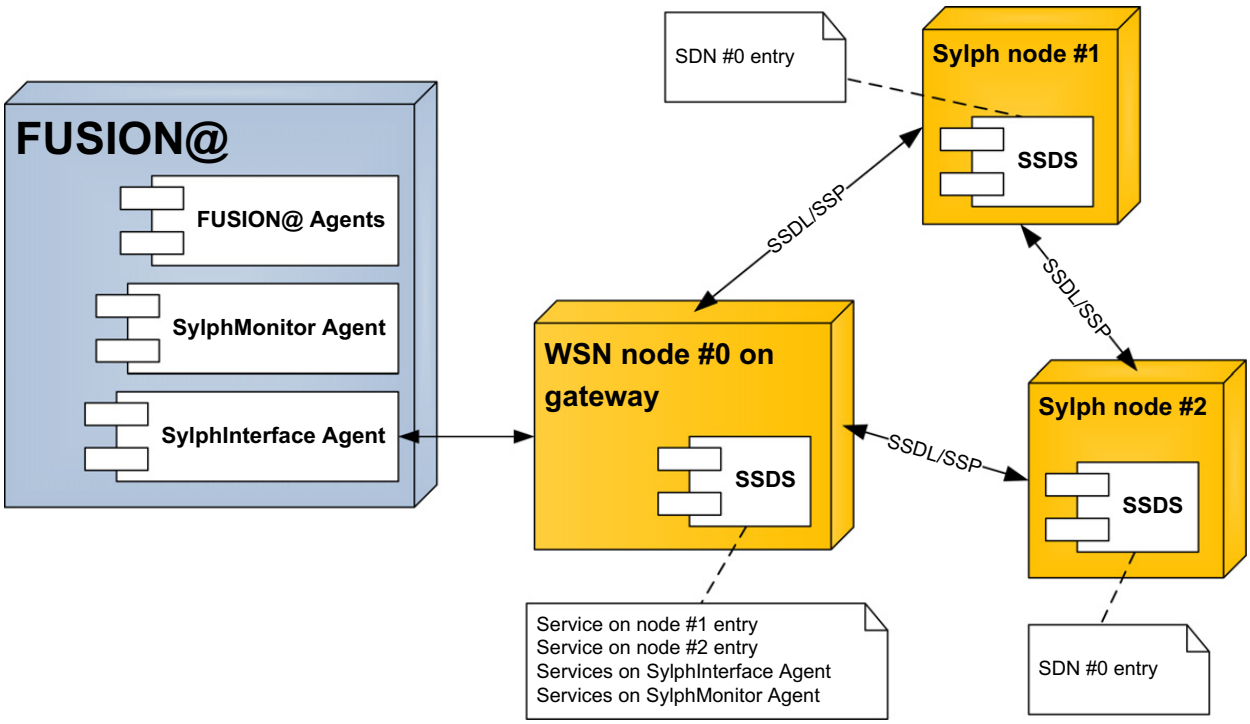


Fig. 6. UML deployment diagram of the interaction between SYLPH and FUSION@ where HERA is not yet present. Even though the features of FUSION@ provide an easy way to access SYLPH services as other services and applications, the integration of HERA into FUSION@ improves this architecture embedding agents directly on wireless sensor nodes.

Author's personal copy

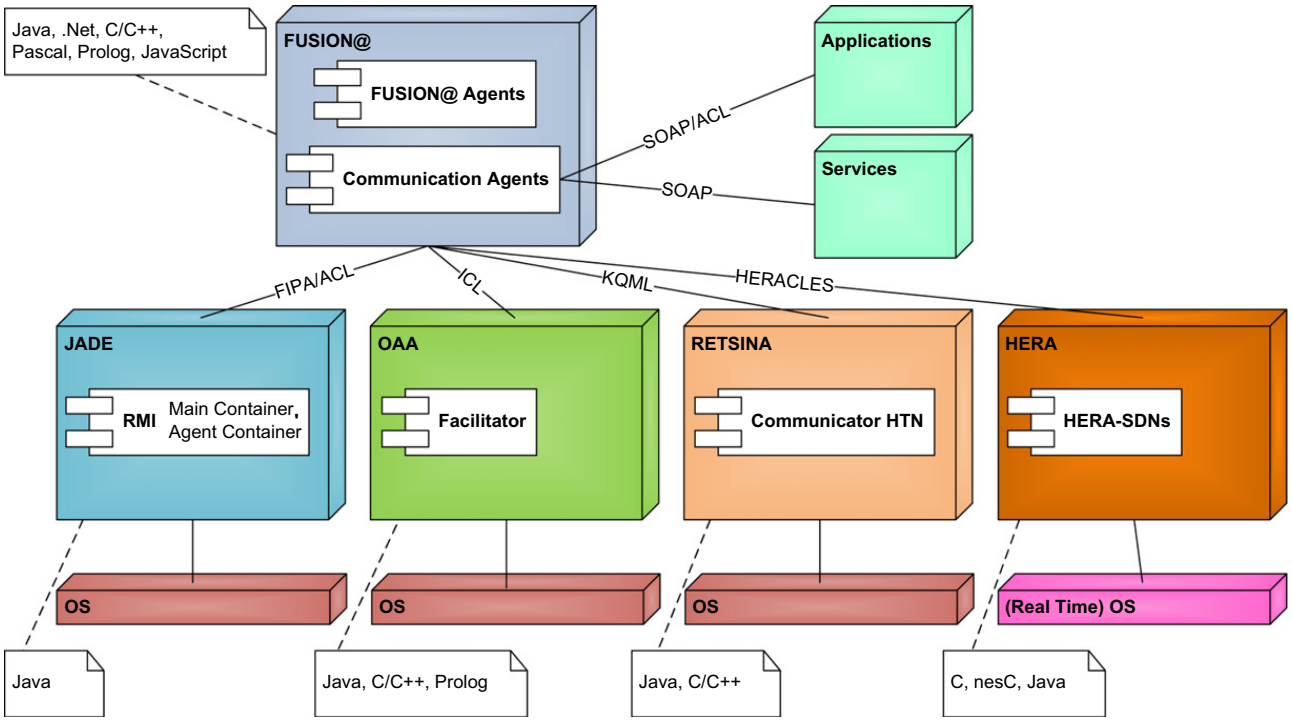


Fig. 7. UML deployment diagram representing the integration of HERA platform into FUSION@ architecture. As can be seen, HERA interact with FUSION@ as other agent platform, no matter if HERA Agents are, in fact, hardware-embedded agents running on wireless sensor nodes from different radio technologies.

The *SylphMonitor Agent* allows the agent platform to monitor the state and operation of the SYLPH network. Thus, the *SylphMonitor Agent* monitors all the traffic (i.e., service invocations, responses, registrations or searches) in the SYLPH network. It is necessary for the nodes to operate in a special debug mode, so that every time a node invokes a service it also invokes a monitoring service on a node connected to the SYLPH WSN node in the gateway. The node gathers all the invocations and forwards them to the *SylphMonitor Agent* running on the agent platform. The same process is done for service responses, searches and registrations. The *SylphMonitor Agent* makes it possible to observe when a node is searching for a certain service in the network, the services offered by the nodes, and the contents of the SSDS entries tables stored in the SDNs.

In the new proposal, with the integration of the HERA platform into the FUSION@ architecture, hardware agents of HERA and software agents of FUSION@ can communicate with each other in a transparent way. Even though HERA can run on just light hardware nodes, its integration into FUSION@ allows building systems and applications where hardware reactive agents can interact directly with heavier software deliberative agents. This integration can be seen on the UML deployment diagram shown in Fig. 7. FUSION@ was enhanced in order to support HERACLES for both *Services* and *Applications* blocks. To do this, the *CommServ Agent* and the *CommApp Agent* were modified in FUSION@ to support the HERACLES language. This way, the HERA-SDNs can communicate with the *CommServ Agent* and the *CommApp Agent* through the HERACLES language.

The manner in which several heterogeneous HERA/SYLPH WSNs are connected with the pre-defined agents of FUSION@ is done through a HERA-SDN running on a SYLPH Gateway. Therefore, predefined-agents of FUSION@ run, for example, in a JADE platform on a complex machine, as a server or a workstation. Any platform that understands ACL is valid for running pre-defined FUSION@ agents. There are SYLPH Gateways that interconnect the server and one of the SYLPH WSNs. This SYLPH Gateway can be, for example, a ZigBee node connected through a serial port (e.g., USB or RS-232) to the server and some code in the server where SYLPH layers and protocols are implemented. In addition, a HERA-SDN on that same SYLPH Gateway is instantiated in the HERA platform running over the heterogeneous SYLPH WSNs. This way, the HERA-SDN is a HERA Agent that can communicate through HERACLES both with the HERA Agents in the WSN nodes and with the *CommServ Agent* and the *CommApp Agent* in the FUSION@ platform on the server, as now they understand the HERACLES language.

4. Experiments and results

In order to test the HERA platform, its integration into FUSION@ and its feasibility in AmI-based systems, we deployed a distributed WSN infrastructure with HERA running over it. The infrastructure consists of a ZigBee network with 31 nodes, one acting as ZigBee coordinator and the rest as ZigBee routers. Each ZigBee node includes an 8-bit C51-based microcontroller with 8448 bytes of RAM and 128 KB of Flash memory and an IEEE 802.15.4/ZigBee transceiver. These devices include temperature and light sensors, some buttons and some LEDs, which allows them to be used as sensors and actuators. In fact,

Author's personal copy

these devices were selected because we had already used them in previous developments and we intend to integrate them in future projects as well. The ZigBee nodes are distributed in a short-range simple mesh, with less than 10 m between any router and the coordinator. Each time the ZigBee network is formed, nodes are powered on different random times, so that the mesh topology is different each time. This means that the number of neighbors and children nodes of any node and their addresses is different every time. However, they are some constraints: the maximum depth of the network (i.e., the maximum number of hops between the coordinator and any node in the network) is 5, the maximum number of neighbors of any node is 8 and the maximum number of children of any node in the network is also 8. Each time the ZigBee network is formed, the SYLPH platform is started over it. This way, the ZigBee coordinator has SDN capabilities and also SYLPH Gateway capabilities for connecting the ZigBee network with a server through a USB port, if that were ever required.

We have also developed an application for monitoring the state and operation of the HERA network. This application is based on a previous one we had developed for monitoring SYLPH networks. As with the previous one, this application monitors all the traffic (i.e., service invocations, responses, registrations or searches) in the SYLPH network. It is necessary for the nodes to operate in debug mode, so that every time a node invokes a service it also invokes a monitoring service on a node connected to a computer (e.g., via a USB port). The node gathers all the invocations and forwards them to the monitoring application running on the computer. The same process is done for service responses, searches and registrations. The monitoring application makes it possible to observe when a node is searching for a certain service in the network, the services offered by the nodes, and the contents of the SSDS entry tables stored in the SDNs. In addition, with the newly developed application it is possible to monitor the HERA Agent instances in the HERA-SDNs of the HERA platform and also the HERACLES *request*, *inform* and other frames.

Several experiments were carried out to evaluate the performance of both HERA alone (i.e., without FUSION@) and HERA with FUSION@, mainly to test how it handled the instances of HERA-SDN and HERA Agents, as well as the exchange of frames between agents through HERACLES. The first experiment consisted of trying to start a platform with just HERA over a ZigBee SYLPH network. As previously described, the network was made up of 31 ZigBee nodes, one of them acting as SDN and 30 acting as SYLPH nodes. After the entire SYLPH network is correctly created the coordinator and SDN try to instance a HERA-SDN. HERA-SDN instances itself and starts the HERA platform registering a special SYLPH service called ''HERA'' on the SDN stored on the same ZigBee coordinator node. Then, 10 of the 30 SYLPH nodes try to instance one HERA Agent, each of them in the HERA platform. Once the HERA-SDN and the 10 HERA Agents were successfully instantiated, the HERA-SDN started to ''ping'' every of the 10 HERA Agents with a HERACLES *request* frame including an *inform-if* command and waiting for a *inform* frame as a ''pong'' response. Each HERA Agent is pinged by the HERA-SDN one time every 5 s during 1 h. This experiment was performed to measure the success ratio of the platform start and the agent instantiation. The experiment was run until the platform and the agents were successfully started and instantiated 50 times. When the SYLPH network could not be correctly created the run was discarded and was not taken into account in the 50 runs. Furthermore, if the HERA platform could not be completely started and created (i.e., all 10 HERA Agents correctly instantiated), these runs were also discarded and not taken into account as forming part of the 50 runs. This way, the experiment was run 55 times. If any HERA Agent crashed (e.g., it got blocked or the node fell from the WSN) it was immediately restarted. Through the HERA monitoring application previously described, HERACLES messages were registered in order to measure when a ping-pong failed and if a HERA Agent had to be restarted. The results are shown in Table 2 which indicates that it is necessary to improve SYLPH creation and the instantiation of HERA Agents. In the first case, a better ARQ (*Automatic Repeat Request*) mechanism could increase SSP-over-WSN transmissions. In the second case, it is necessary to debug the implementation of the Agents and fix errors. In addition, the robustness of the HERA Agents should be improved by introducing a mechanism to ping and keep running the HERA Agents and the HERA-SDNs.

The second experiment consisted of the same 21 ZigBee SYLPH nodes, but this time the ZigBee coordinator was both the SDN and SYLPH Gateway between the WSN and a computer through a USB port. This experiment is very similar to the

Table 2
Results of the HERA-alone and HERA into FUSION@ experiments.

HERA-alone experiments	
Total runs	55
SYLPH not created correctly (% of total runs)	2 (3.6%)
HERA not started correctly (% of SYLPH correctly created)	3 (5.6%)
All 10 HERA Agents correctly instantiated	50
Total pings tried	7200
Ping-pongs not completed (% of total tried)	15 (0.2%)
Total restarted HERA Agents in an hour	8
HERA into FUSION@ experiments	
Total tries	57
SYLPH not created correctly (% of total runs)	3 (5.2%)
HERA not started correctly (% of SYLPH correctly created)	4 (8.0%)
All 10 HERA Agents correctly instantiated	50
Total pings tried	7200
Ping-pongs not completed (% of total tried)	27 (0.3%)
Total restarted HERA Agents in an hour	6

Author's personal copy

previous one. However, this time HERA is integrated into FUSION@. Pre-defined agents of FUSION@ are created in a JADE platform running on the computer. It is only possible to try to create the SYLPH network, and then the HERA Agents, if the JADE platform and all FUSION@ pre-defined agents are correctly instantiated. In this case, the HERA-SDN is also a SYLPH Gateway between the ZigBee network and the computer. Moreover, the responsibility for sending the pings to the HERA Agents belongs to the *Admin Agent* of FUSION@. The *Admin Agent* uses ACL requests to the *CommServ Agent*, which communicates with the HERA-SDN in the SYLPH SDN/Gateway through HERACLES. As shown in Table 2 the SYLPH creation and HERA start is a little worse when HERA is integrated into FUSION@ than when HERA is running alone. Moreover, when running with FUSION@, HERACLES ping-pongs fail quite a bit more. This is because the use of the SYLPH Gateway through the USB serial port introduces some delay in the transmission and overloads the SYLPH SDN/Gateway node. Nevertheless, the number of HERA Agents that should be restarted was very similar when running with FUSION@ which indicates that FUSION@ does not affect the robustness of the HERA platform.

The results of these experiments demonstrate the feasibility of the integration of the HERA platform into FUSION@. These results show that the performance of HERA is hardly affected by its integration into FUSION@. This way, this integration enhances the latter with some features that FUSION@ do not provide by itself. These features include the possibility of working with heterogeneous wireless sensor networks to build Ambient Intelligence based systems and applications that can perform an improved context-awareness. Furthermore, the context of the environment and users is maintained by means of directly Hardware-Embedded Reactive Agents that can interact with other existing software agents in a transparent way.

5. Conclusions and future work

The integration of the well-proven FUSION@ architecture and the new HERA platform is presented in this paper. FUSION@ (*Flexible and User Services Oriented Multi-agent Architecture*) facilitates the development of dynamic and intelligent multi-agent systems. Its model is based on a service-oriented approach, by formalizing services, applications, communications and deliberative agents. The architecture proposes an alternative where agents act as controllers and coordinators. FUSION@ exploits the agents' characteristics to provide a robust, flexible, modular and adaptable solution that can cover most requirements that can be found in a wide array of distributed systems. All functionalities, including those of the agents, are modeled as distributed services and applications.

The HERA (*Hardware-Embedded Reactive Agents*) platform allows wireless devices from different technologies to work together in a distributed way. These devices do not require large memory chips or fast microprocessors to exploit their functionalities. The HERA model was designed to be implemented on AmI-based systems. However, it can be used by any kind of complex systems as it is capable of integrating almost any desired functionality. HERA Agents can communicate in a distributed way, even from devices with reduced computational resources. Because of SYLPH, HERA Agents can communicate with each other regardless of the technology or the programming language they use. Furthermore, an important new feature provided by the HERA platform is that HERA Agents are light enough to be run on WSN nodes with limited resources.

The HERA Agents are reactive because they act on devices with critical response times. Applications and services of FUSION@ can communicate with the HERA Agents in order to obtain information from the context and modify the environment. HERA Agents are also integrated in the multi-agent system of FUSION@. Therefore, the results obtained by BDI agents directly influence the actions of the reactive agents.

These features make the integration of FUSION@ and HERA be an ideal approach to build applications and systems based on Ambient Intelligence that take advantage of the benefits of both Multi-Agent Systems and heterogeneous Wireless Sensor Networks. This way, it is possible to use this innovative proposal to implement any AmI-based application that requires the use of sensor nodes belonging to different wireless technologies. Moreover, the integration of HERA into FUSION@ allows such applications to use light reactive agents directly embedded into the sensor nodes, which are not required to have many computational and memory resources to accomplish their tasks in the system. Furthermore, FUSION@ provides these applications with the integration of software agents that interact with HERA Agents in a transparent way.

There is a wide range of AmI-based applications that could be benefitted from these features. Some examples of these are several applications that have been developed by the BISITE Research Group, such as a healthcare Tele-monitoring application where SYLPH and FUSION@ have already been tested [1]. This healthcare application is focused on performing a remote tele-monitoring of patients at their own homes.

As future work, this healthcare application will be enhanced by the introduction of the integration of HERA into FUSION@. In the new design based on FUSION@ and HERA, there will be HERA Agents running on each sensor node. As in the former design, there will be WSNs from different technologies, such as ZigBee and Bluetooth. On the one hand, ZigBee networks will be used for implement presence detectors, panic button actuators carried by each patient and home automation sensor and actuators. On the other hand, Bluetooth nodes will be used as biomedical sensors to control the patients' vital signs, such as temperature, breath and preventing possible fails. The features of FUSION@ will be used for control the sensors and actuators deployed by the patient's home, accessing them as services by the FUSION@ agents and, thus, performing monitoring and management tasks using reasoning mechanisms. Nevertheless, it is also intended to test the use of the integration of HERA into FUSION@ on other real AmI scenarios, such as surveillance routes, healthcare for Alzheimer patients [46] or even educational environments.

Future work also includes the improvement of the overall performance of the HERA platform. This way, the underlying SYLPH platform will be also improved, especially in the network formation and the SYLPH Gateways. Furthermore, we

Author's personal copy

are working in the design of an efficient mechanism that allows HERA Agents to move throughout different nodes, no matter the WSN technology they use. This way, we will get, for example, HERA Agents to move from a ZigBee node to a Bluetooth node through HERA, allowing the use of different programming languages and operating systems.

Acknowledgment

This work has been supported by Project PET2008_0036 and FEDER funds.

Appendix A. Appendix

List 1. Sample of the use of the human-readable version of SSDL to define a SYLPH service called `registerServiceOnFireAlarm`.

```
service registerServiceOnFireAlarm {
  input {
    uint16 threshold;
    servicepoint callback {
      output {
        boolean status; }; }; };
  output {
    boolean status; }; };
```

List 2. Sample of the human-readable representation of HERACLES language to send messages through HERA.

```
request {
  sender agent1;
  receiver agent2;
  content {
    action(agent2) {
      inform-if {
        sender agent1;
        receiver agent2;
        content {
          message {
          state result; }; };
        language HERACLES;
        ontology HERA_ONTOLOGY; }; }; };
  language HERACLES;
  ontology HERA_ONTOLOGY; };
inform {
  sender agent2;
  receiver agent1;
  content {
    message {
      state result; }; };
  language HERACLES;
  ontology HERA_ONTOLOGY; };
```

References

[1] R.S. Alonso, O. García, C. Zato, O. Gil, F. De la Prieta, Intelligent agents and wireless sensor networks: a healthcare telemonitoring system, in: Y. Demazeau, F. Dignum, J.M. Corchado, J. Bajo, R. Corchuelo, E. Corchado, et al. (Eds.), Trends in Practical Applications of Agents and Multiagent Systems, Springer, Berlin/Heidelberg, 2010, pp. 429–436.
[2] L. Ardissono, G. Petrone, M. Segnan, A conversational approach to the interaction with Web services, Computational Intelligence 20 (2004) 693–709.
[3] A. Avizienis, J. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing 1 (2004) 11–33.
[4] P. Baker, V. Catterson, S. McArthur, Integrating an agent-based wireless sensor network within an existing multi-agent condition monitoring system, in: 15th International Conference On Intelligent System Applications to Power Systems, 2009, ISAP '09, 2009, pp. 1–6.
[5] B. Baruque, E. Corchado, A. Mata, J.M. Corchado, A forecasting solution to the oil spill problem based on a hybrid intelligent system, Information Sciences 180 (2010) 2029–2043.
[6] A.L. Bauer, C.A. Beauchemin, A.S. Perelson, Agent-based modeling of host-pathogen systems: the successes and challenges, Information Sciences 179 (2009) 1379–1389.
[7] F. Bellifemine, A. Poggi, G. Rimassa, Developing multi-agent systems with a FIPA-compliant agent framework, Software: Practice and Experience 31 (2001) 103–128.

Author's personal copy

[8] M.L. Borrajo, J.M. Corchado, E.S. Corchado, M.A. Pellicer, J. Bajo, Multi-agent neural business control system, Information Sciences 180 (2010) 911–927.
[9] M.E. Bratman, D. Israel, M.E. Pollack, Plans and resource-bounded practical reasoning, Computational Intelligence 4 (1988) 349–355.
[10] L.M. Camarinha-Matos, H. Afsarmanesh, A comprehensive modeling framework for collaborative networked organizations, Journal of Intelligent Manufacturing 18 (2007) 529–542.
[11] C. Carrascosa, J. Bajo, V. Julian, J.M. Corchado, V. Botti, Hybrid multi-agent architecture as a real-time problem-solving model, Expert Systems with Applications 34 (2008) 2–17.
[12] E. Cerami, Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, first ed., O'Reilly Media, Inc., 2002.
[13] M. Chen, T. Kwon, Y. Yuan, V.C. Leung, Mobile agent based wireless sensor networks, Journal of Computers 1 (2006) 14–21.
[14] M. Chen, T. Kwon, Y. Yuan, Y. Choi, V.C.M. Leung, Mobile agent-based directed diffusion in wireless sensor networks, EURASIP Journal on Applied Signal Processing (2007) 219.
[15] J.M. Corchado, J. Bajo, A. Abraham, GerAmi: improving healthcare delivery in geriatric residences, IEEE Transactions on Intelligent Systems 23 (2008) 19–25.
[16] J.M. Corchado, J. Bajo, Y. de Paz, D.I. Tapia, Intelligent environment for monitoring Alzheimer patients, agent technology for health care, Decision Support Systems 44 (2008) 382–396.
[17] J.M. Corchado, J. Bajo, D.I. Tapia, A. Abraham, Using heterogeneous wireless sensor networks in a telemonitoring system for healthcare, IEEE Transactions on Information Technology in Biomedicine 14 (2010) 234–240.
[18] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana, Unraveling the Web services web: an introduction to SOAP, WSDLand UDDI, IEEE Internet Computing 6 (2002) 86–93.
[19] A.K. Dey, G.D. Abowd, D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human–Computer Interaction 16 (2001) 97–166.
[20] C. Fok, G. Roman, C. Lu, Mobile agent middleware for sensor networks: an application case study, in: Fourth International Symposium on Information Processing in Sensor Networks 2005, IPSN 2005, 2005, pp. 382–387.
[21] J.A. Fraile, J. Bajo, J.M. Corchado. AMADE: developing a multi-agent architecture for home care environments, in: 7th Ibero-American Workshop in Multi-Agent Systems, Lisbon, Portugal, 2008, pp. 193–202.
[22] Q. Guo, M. Zhang, A novel approach for multi-agent-based intelligent manufacturing system, Information Sciences 179 (2009) 3079–3090.
[23] S. Ilarri, E. Mena, A. Illarramendi, Using cooperative mobile agents to monitor distributed and dynamic environments, Information Sciences 178 (2008) 2105–2127.
[24] R. Jedermann, C. Behrens, D. Westphal, W. Lang, Applying autonomous sensor systems in logistics – combining sensor networks, RFIDs and software agents, Sensors and Actuators A: Physical 132 (2006) 370–375.
[25] N.R. Jennings, K. Sycara, M. Wooldridge, A roadmap of agent research and development, Autonomous Agents and Multi-Agent Systems 1 (1998) 7–38.
[26] Y. Kwon, S. Sundresh, K. Mechitov, G. Agha, ActorNet: an actor platform for wireless sensor networks, in: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, Hakodate, Japan, 2006, pp. 1297–1300.
[27] X. Liu, W.J. Zhang, R. Radhakrishnan, Y.L. Tu, Manufacturing perspective of enterprise application integration: the state of the art review, International Journal of Production Research 46 (2008) 4567–4596.
[28] Y. Liu, C. Zhou, K. Wang, D. Li, D. Guo, Multi-agent ERA model based on belief interaction solves wireless sensor networks routing problem, in: Hybrid Artificial Intelligence Systems, Springer, Berlin, Heidelberg, 2008, pp. 30–37.
[29] K. Lyytinen, Y. Yoo, Issues and challenges in ubiquitous computing – introduction, Communications of the ACM 45 (2002) 62–65.
[30] S.S. Manvi, M.S. Kakkasageri, Multicast routing in mobile ad hoc networks by using a multiagent system, Information Sciences 178 (2008) 1611–1628.
[31] M. Marin-Perianu, N. Meratnia, P. Havinga, L. de Souza, J. Muller, P. Spiess, S. Haller, T. Riedel, C. Decker, G. Stromberg, Decentralized enterprise systems: a multiplatform wireless sensor network approach, IEEE Transactions on Wireless Communications 14 (2007) 57–66.
[32] D.L. Martin, A.J. Cheyer, D.B. Moran, The open agent architecture: a framework for building distributed software systems, Applied Artificial Intelligence 13 (1999) 91–128.
[33] S. Mukherjee, E. Aarts, R. Roovers, F. Widdershoven, M. Ouwerkerk, Amiware: Hardware Technology Drivers of Ambient Intelligence, illustrated edition., Springer, 2006.
[34] F. Pecora, A. Cesta, DCOP for smart homes: a case study, Computational Intelligence 23 (2007) 395–419.
[35] H. Qi, F. Wang, Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks, in: Proceedings of the IEEE ICC'01 Helsinki, Finland 2001, pp. 147–153.
[36] R. Rajagopalan, C. Mohan, P. Varshney, K. Mehrotra, Multi-objective mobile agent routing in wireless sensor networks, in: The 2005 IEEE Congress On Evolutionary Computation, voll. 2, 2005, pp. 1730–1737.
[37] S.J. Russell, P. Norvig, J.F. Canny, J. Malik, D.D. Edwards, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, NJ, 1995.
[38] N.M. Sadeh, F.L. Gardon, O.B. Kwon, Ambient Intelligence: The My Campus Experience, Technical Report CMU-ISRI-05-123, ISRI, 2005.
[39] N. Sánchez-Pi, J. Carbó, J. Molina, JADE/LEAP Agents in an Aml Domain, in: Hybrid Artificial Intelligence Systems, Springer, Berlin, Heidelberg, 2008, pp. 62–69.
[40] N. Sanchez-Pi, J.M. Molina, A multi-agent approach for provisioning of e-services in u-commerce environments, Internet Research 20 (2010) 276–295.
[41] J. Sandhu, Wireless sensor networks for commercial lighting control decision making with multi-agent systems, in: AAAI Workshop on Sensor Networks, 2004, pp. 131–140.
[42] J. Sarangapani, Wireless Ad hoc and Sensor Networks: Protocols, Performance, and Control, first ed., CRC, 2007.
[43] E. Serrano, J. Botia, Validating ambient intelligence based ubiquitous computing systems by means of artificial societies, Information Sciences, doi:10.1016/j.ins.2010.11.012.
[44] E.Y. Song, K.B. Lee, STWS: a unified web service for IEEE 1451 smart transducers, IEEE Transactions on Instrumentation and Measurement 57 (2008) 1749–1756.
[45] K. Sycara, M. Paolucci, M. Van Velsen, J. Giampapa, The RETSINA MAS infrastructure, Autonomous Agents and Multi-Agent Systems 7 (2003) 29–48.
[46] D.I. Tapia, A. Abraham, J.M. Corchad, R.S. Alonso, Agents and ambient intelligence: case studies, Journal of Ambient Intelligence and Humanized Computing 1 (2010) 85–93.
[47] D.I. Tapia, J. Bajo, J.M. Corchado, Distributing functionalities in a SOA-based multi-agent architecture, in: 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009), Springer, Berlin/Heidelberg, 2009, pp. 20–29.
[48] D.I. Tapia, J.M. Corchado, An ambient intelligence based multi-agent system for Alzheimer health care, International Journal of Ambient Computing and Intelligence (IJACI) 1 (2009) 15–26.
[49] R. Tynan, G. O'Hare, A. Ruzzelli, Multi-agent system methodology for wireless sensor networks, Multiagent and Grid Systems 2 (2006) 491–503.
[50] M. Wooldridge, An Introduction to MultiAgent Systems, second ed., Wiley, 2009.
[51] F. Zboril, J. Horacek, P. Spacil, Intelligent agent platform and control language for wireless sensor networks, In: Third UKSim European Symposium on Computer Modeling and Simulation, 2009, EMS '09, 2009, pp. 482–487.
[52] W.J. Zhang, Y. Lin, On the principle of design of resilient systems – application to enterprise information systems, Enterprise Information Systems 4 (2010) 99–110.