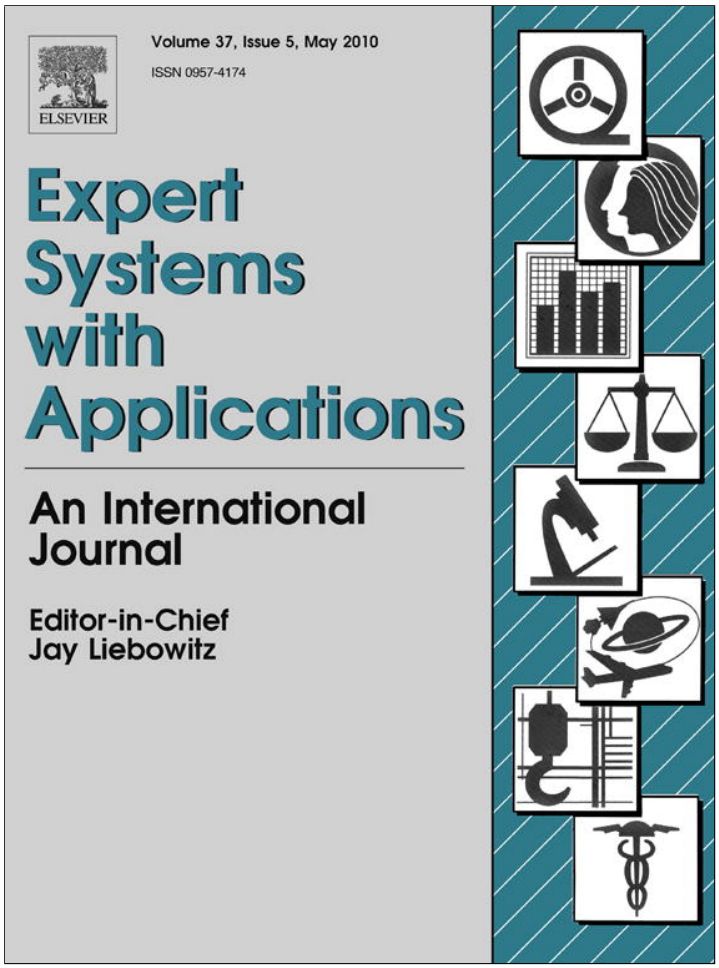

Expert Systems with Applications 37 (2010) 3986–3999

Contents lists available at ScienceDirect

Expert Systems with Applications

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

The THOMAS architecture in Home Care scenarios: A case study

Javier Bajo [a,*], Juan A. Fraile [a], Belén Pérez-Lancho [b], Juan M. Corchado [b]

[a] Universidad Pontificia de Salamanca, Compañía 5, 37002 Salamanca, Spain
[b] Departamento Informática y Automática Universidad de Salamanca, Plaza de la Merced s/n, 37008 Salamanca, Spain

ARTICLE INFO

Keywords:
Dependent environments
Multi-agent systems
Home Care
Virtual organizations

ABSTRACT

Today, the need for architectures and computational models for large scale open multi-agent systems is considered a key issue for the success of agent technology in real world scenarios. The main goal of this paper is to describe a case study in Home Care scenarios applying an abstract architecture and a computational model for large scale open multi-agent systems based on a service-oriented approach. The architecture used is THOMAS, which specifically addresses the design of Home Care systems. This paper presents services examples for the management of a dependent home environment, and demonstrates the new features of the proposal.

1. Introduction

Home Care is one of the objectives of pervasive computing, and dependent people require new solutions that can take advantage of technological advances which provide novel and fundamental services (Angulo & Tellez, 2004). The vision of pervasive computing is to improve the quality, access, equity and continuity of health care (Angulo & Tellez, 2004). In this sense, intelligent environments can improve health care services and can have a high social impact, especially in Home Care services for chronically dependent patients (Augusto & McCullagh, 2007). Because Home Care requires effective communication as well as distributed problem solving (Augusto & McCullagh, 2007), multi-agent systems can facilitate the development of pervasive Home Care environments. Moreover, agent-oriented methodologies provide mechanisms for modelling distributed, inter-operable and secure systems by taking social and organizational considerations into account. Agents are autonomous software entities (Camarinha-Matos & Afsarmanesh, 2004) able to interact with their surroundings. They are highly capable of adapting to changes, and can allow for integration with multiple devices, sensors and humans.

The continuous growth of the dependent people sector has dramatically increased the need for new Home Care solutions (Anastasopoulos, Niebuhr, Bartelt, Koch, & Rausch, 2005; Corchado & Laza, 2003). Furthermore, commitments to meet the needs of this sector suggest that the current systems are in need of modernization. Multi-agent systems (Want, Pering, Borriello, & Farkas, 2002) and intelligent device-based architectures have been recently explored as supervisor systems in health care scenarios (Angulo & Tellez, 2004) for elderly people and for Alzheimer patients (Corchado & Laza, 2003). These systems are capable of providing constant care in the daily life of dependent patients (Carrascosa, Bajo, Julian, Corchado, & Botti, 2008), predicting potentially dangerous situations, and facilitating a cognitive and physical support for the dependent patient (Augusto & McCullagh, 2007). Taking these solutions into account, it is possible to assume that multi-agent systems can further facilitate the design and development of pervasive environments (Corchado, Bajo, de Paz, & Tapia, 2008) and improve the services currently available by incorporating new functionalities. Multi-agent systems add a high level of abstraction with respect to the traditional distributed computing solutions. They also facilitate the analysis and design of the problem in terms of artificial intelligence systems. This allows a greater flexibility for incorporating human behaviours into the agent's structure. Multi-agent system technology makes it possible to cover a broad area of problems. Typical problems are systems in which there are several entities (Requesters) which may require one or more elements or services from other different entities (Bidders). In the area of Home Care, for example, Requesters would be patients and Bidders would be companies which provide services, such as identification, localization, home automation services, or warnings and alerts. Obviously, the development of these types of systems is complex and, therefore, it is necessary to analyze the intrinsic characteristics of these typical application environments in detail.

The aim of this research project is to present a case study applying the THOMAS (MeTHods, Techniques and Tools for Open Multi-Agent Systems) multi-agent architecture. THOMAS has been used to develop a case study for supervising and monitoring dependent patients at home. This multi-agent system offers a series of functionalities including an automatic reasoning and planning mechanism for scheduling the medical staff working day, an alert system,

* Corresponding author. Tel.: +34 639771985; fax: +34 923277101.
E-mail addresses: jbajope@upsa.es (J. Bajo), jafraileni@upsa.es (J.A. Fraile), lancho@usal.es (B. Pérez-Lancho), corchado@usal.es (J.M. Corchado).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.11.017

a location and tracking system, and an identification system. The medical staff has been provided with PDAs and mobile phones, as well as with Java Card tags, and the home environments have been equipped with presence detection sensors, access control mechanisms, door opening devices and video cameras. The multi-agent system monitors the daily routine of the patient and detects dangerous situations. If any anomalous situation is detected, the alert system is used to obtain medical assistance.

The remainder of this paper is structured as follows: Section 2 provides an analysis of related studies; Section 3 presents the proposed architecture model as well as a description of the services offered by each one of the modules that make up the reference model; Section 4 shows an example of an implementation, highlighting the new possibilities provided by this type of architecture, and presents a specific approach for Home Care management; finally, some conclusions of this study are shown in Section 5.

2. Related works

Dependence is a permanent situation in which a person needs important assistance from others in order to perform basic daily life activities such as essential mobility, object and people recognition, and domestic tasks (Costa-Font & Patxot, 2005). There is an ever growing need to supply constant care and support to the disabled and elderly, and the drive to find more effective ways of providing such care has become a major challenge for the scientific community (Nealon & Moreno, 2003). The importance of developing new and more reliable ways of providing care and support for the elderly is underscored by this situation, and the creation of secure, unobtrusive and adaptable environments for monitoring and optimizing health care will become vital. Some authors (Nealon & Moreno, 2003) consider that tomorrow's health care institutions will be equipped with intelligent systems capable of interacting with humans. The intelligent systems aim to support patients in all aspects of daily life (Cesta, Bahadori, Cortellesa, Grisetti, & Giuliani, 2003), predicting potential hazardous situations and delivering physical and cognitive support (Bahadori et al., 2003).

Home Care systems aim to improve quality of life, offering more efficient and easy ways to use services and communication tools to interact with other people, systems and environments. Among the general population, those most likely to benefit from the development of these systems are the elderly and dependent persons (i.e., those suffering from degenerative diseases, dementia or loss of cognitive ability (Costa-Font & Patxot, 2005)), whose daily lives, with particular regard to health care, will be most enhanced (Corchado et al., 2008; Van Woerden, 2006).

Agents and multi-agent systems in dependency environments are becoming a reality, especially in health care. Most agent-based applications are related to the use of this technology in the monitoring of patients, treatment supervision and data mining. Lanzola, Gatti, Falasconi, and Stefanelli (1999) present a methodology that facilitates the development of inter-operable intelligent software agents for medical applications, and propose a generic computational model for implementing them. The model may be specialized in order to support all the different information and knowledge-related requirements of a hospital information system. Meunier (1999) proposes the use of virtual machines to support mobile software agents by using a functional programming paradigm. This virtual machine provides the application developer with a rich and robust platform upon which to develop distributed mobile agent applications, specifically when targeting distributed medical information and distributed image processing. While an interesting proposal, it is not viable due to the security reasons that affect mobile agents, and there is no defined alternative for locating patients or generating planning strategies. There are also agent-based systems that help patients get the best possible treatment, and that remind the patient about follow-up tests (Miksch, Cheng, & Hayes-Roth, 1997). They assist the patient in managing continuing ambulatory conditions (chronic problems). They also provide health-related information by allowing the patient to interact with the on-line health care information network. Decker & Li propose (Decker & Li, 1998) a system to increase hospital efficiency by using global planning and scheduling techniques. They propose a multi-agent solution that uses the generalized partial global planning approach which preserves the existing human organization and authority structures, while providing better system-level performance (increased hospital unit throughput and decreased impatient length of stay time). To do this, they use resource constraint scheduling to extend the proposed planning method with a coordination mechanism that handles mutually exclusive resource relationships. Other applications focus on home scenarios to provide assistance to elderly and dependent persons. RoboCare presents a multi-agent approach that covers several research areas, such as intelligent agents, visualization tools, robotics, and data analysis techniques to support people with their daily life activities (Pecora & Cesta, 2007). TeleCARE is another application that makes use of mobile agents and a generic platform in order to provide remote services and automate an entire home scenario for elderly people (Camarinha-Matos & Afsarmanesh, 2004). Although these applications expand the possibilities and stimulate research efforts to enhance the assistance and health care provided to elderly and dependent persons, none of them integrate intelligent agents, distributed and dynamic applications and services approach into their model.

In multi-agent systems (MAS) one of the most important goals is to build systems capable of autonomous and flexible decision-making. Moreover, these systems must cooperate with others within a "society". Due to the technological advances of recent years, the term "society", in which the multi-agent system participates, needs to meet several requirements such as: distribution, constant evolution, flexibility to allow members to enter or exit the society, appropriate management of the organizational structure that defines the society, multi-device agent execution including devices with limited resources, and so on. All of these requirements define a set of features that can be addressed through an open system paradigm and virtual organizations. Despite the large number of agent platforms in existence, the majority are lacking in the management of virtual organizations for dynamic, open and large-scale environments. For example, the most well-known agent platforms (like *JADE*) (Argente et al., 2007) offer basic functionalities to agents, such as AMS and DF services; but designers must implement nearly all of the organizational features by themselves, namely organization representation, control mechanisms, organization descriptions, AMS and DF extensions, communication layer, monitoring, organization modelling support and organizational API.

The state-of-the-art in this field shows research interest in the integration of mobile agents and services, where agents are complex entities that can handle the problem of service discovery and composition in dynamic and changing open environments. Agents are not organized into plain societies, but into structured organizations that enclose the real world with the society representation and ease the development of open and heterogeneous systems. Current agent platforms must integrate these concepts to allow designers to employ higher abstractions when modelling and implementing these complex systems. All of these concerns can be found in the THOMAS architecture.

3. THOMAS architecture model

THOMAS architecture basically consists of a set of modular services. Though THOMAS feeds initially on the FIPA architecture, it

expands its capabilities to deal with organizations and to boost its services abilities. In this way, a new module in charge of managing organizations is introduced into the architecture, along with a redefinition of the FIPA Directory Facilitator that is able to deal with services in a more elaborate way, following Service Oriented Architectures guidelines. As previously stated, services are very important in this architecture. In fact, agents have access to the THOMAS infrastructure through a range of services included in different modules or components. The main components of THOMAS are shown in Fig. 1.

- Service Facilitator (SF): this component offers simple and complex services to the active agents and organizations. Basically, it is both a yellow pages service, and a green pages service descriptor.
- Organization Management System (OMS): mainly responsible for the management of the organizations and their entities, thus allowing for the creation and management of any organization.
- Platform Kernel (PK): maintains basic management services for an agent platform.

The following sections describe the different components of the THOMAS architecture in greater detail.

3.1. Service Facilitator

The Service Facilitator (SF) is a mechanism and support by which organizations and agents can offer and find services. The SF provides a place in which the autonomous entities can register service descriptions as directory entries.

The SF acts as a gateway to access the THOMAS platform. It manages this access transparently, by means of security techniques and access rights management. The SF can find services by searching for a given service profile or searching for the goals that can be fulfilled when executing the service. This is done by using the matchmaking (Sycara, Widoffand, Klusch, & Lu, 1982) and service composition mechanisms (Corchado & Laza, 2003) which are provided by the SF. The SF also acts as a yellow pages manager and in this way it can find which entities provide a given service.

A service represents an interaction of two entities, which is modelled as communication among independent processes. Services can be described as offering capabilities, each of which enables the fulfilment of a given goal. The service may have some preconditions, which need to be true for the service execution. Moreover, both service client and provider exchange one or more input and output messages during the service execution, which affects their environment. There could also be additional parameters in a service description, which are independent of the service functionality (non-functional parameters), such as quality of service, deadlines and security protocols. Finally, the service results can be enhanced using automatic service composition mechanisms (Klusch, Fries, & Sycara, 2006; Brogi, Corfini, & Popescu, 2005) (for example, partial matchmaking). To do this the SF stores the description of the internal processes that are executed when the service is running.

In our case, multi-agent technology provides us with FIPA communication protocols which are well established mechanisms for standardizing the interactions. Thus, every service has an associated protocol. In cases where the service requires the execution of a chain of protocols, the service is marked as "complex". Given that THOMAS works with semantic services, another important piece of data is the ontology used in the service. Thus, when the service description is accessed, any entity will have all of the information necessary in order to interact with the service and make an application that can use this service. Such a description can also be employed for pre-compiled services, in which the process model of the service is composed of the sequence of the elementary services to be executed, instead of the internal processes of this service.

Normally, a service can be supplied by more than one provider in the system. Therefore, a service has an associated list of providers. All providers can offer exact copies of the service, given that they share a common implementation of the service. Or they may share only the interface and each provider may implement the service in a different way. This is easily achieved in THOMAS

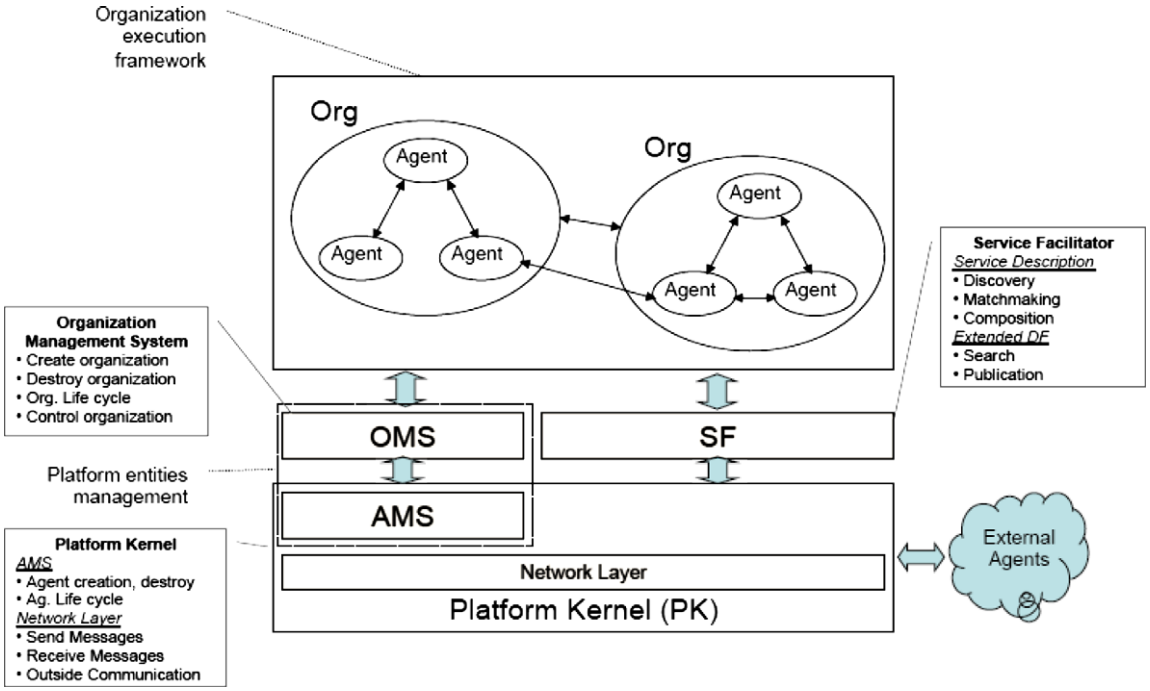

Fig. 1. THOMAS architecture.

because the general service profile is separate from the service process.

A service is defined as a tuple (sID, goal, prof, proc, ground, ont):

- **sID** is a unique service identifier;
- **goal** is the final purpose of the service and provides the first abstraction level for service composition;
- **prof** is the service profile which describes the service in terms of its IOPEs (Inputs, Outputs, Preconditions and Effects) and non-functional attributes, in a readable way for those agents searching for information (or matchmaking agents which act as searching service agents). This type of representation includes a description of what the service fulfils, the constraints to its applicability and quality of service, and the requirements that clients have to satisfy in order to use the service.
- **proc** describes how a client has to use the service; it specifies the semantic content for using the service, the situations in which it is obtained, and, whenever required, the step-by-step processes to get these results. In other words, it specifies how to call a service and what should happen when the service is executed.
- **ground** specifies in detail how an agent can access the service. Grounding specifies a communication protocol, the message formats, the contact port and other specific details of the service. It is specified using the OWL-S standard extended with FIPA protocols.
- **ont** is the ontology that gives meaning to all of the elements of the service. OWL-DL is the chosen language.

This proposal is based on OWL-S specification for semantic web services, extended when it is necessary to empower its functionality. Goals, preconditions and effects (or post-conditions) are logical formulas.

The tuple defined above for service specification is implemented in two parts: the abstract service, general for all providers; and the concrete service, with the implementation details. As such, services are stored within the system and split into two parts: the service profile (which represents the abstract service specification) and a set of service process specifications (which detail the specific service). Thus, in THOMAS services are implemented as the following tuple, in which its elements are OWL-S extended specifications:

<ServiceID, Providers, ServGoal, ServProfile>
Providers::= **<ProvIDList, ServImpID, ServProcess, ServGround>**
ProvIDList::= **ProviderID**

where

- Providers are a set of tuples composed of a Providers identifier list (ProvIDList), the service process model specification (ServProcess), and its particular instantiation (ServGround).
- ProvIDList stores a list of service provider identifiers.

The SF supplies a set of standard services (meta-services) to manage the services provided by organizations or individual agents (see Table 1). These meta-services also have to be used by the remaining THOMAS components (OMS and PK) to advertise their own services. SF meta-services can be classified in three types:

- Registration: allowing the addition, modification and removal of services from the SF directory.
- Affordability: for managing the association between providers and their services.
- Discovery: for searching and composing services as an answer to user requirements.

Their functionality is so complex that they can be delegated to a specialized component.

3.2. Organization Management System

The Organization Management System (OMS) is in charge of organization life-cycle management, including specification and administration of both its structural components (roles, units and norms) and its execution components (participant agents and roles they play, and active organizational units). Organizations are structured by means of organizational units, which represent groups of entities (agents or other units), which coordinate in order to pursue a common goal. These organizational units have an internal topology (i.e., hierarchical, team, plain), which imposes restrictions on agent relationships and control (e.g., supervision or information relationships).

In THOMAS, a "virtual" unit has been defined in order to represent the "world" system in which agents participate by default. The OMS creates organizations within this "virtual" unit by registering organizational units which can in turn be composed of more units. Moreover, roles are defined in each unit. They represent all of the functionality needed to achieve the unit goal. They might also have associated norms for controlling role actions (i.e., which of the services agents playing that role are allowed to request, offer or serve; permissions for accessing resources). As a result, agents can dynamically adopt roles within units. As such, the OMS controls this role adoption process and determines which entities play each role through time.

The OMS component makes use of the following information:

- UnitList: stores existing units, together with their objectives, topology and parent unit.
- RoleList: stores the list of roles defined in each unit and their attributes (accessibility, visibility, position and inheritance). Accessibility indicates whether a role can be adopted by an agent on demand; visibility indicates whether agents can obtain

Table 1
SF meta-services.

Type	Meta-service	Description
Registration	RegisterProfile	Creates a new service description (profile)
	RegisterProcess	Creates a particular implementation (process) for a service
	ModifyProfile	Modifies an existing service profile
	ModifyProcess	Modifies an existing service process
	DeregisterProfile	Removes a service description
Affordability	AddProvider	Adds a new provider to an existing service process
	RemoveProvider	Removes a provider from a service process
Discovery	SearchService	Searches a service (or a composition of services) that satisfies the user requirements
	GetProfile	Gets the description (profile) of a specific a service
	GetProcess	Gets the implementation (process) of a specific a service

information from this role on demand; position indicates whether it is a supervisor, subordinate or member of the unit; and inheritance indicates its parent role.
- NormList: stores norms defined in the system.
- EntityPlayList: describes <entity, unit, role> association, i.e., which roles have been adopted by an entity (agent) within each unit.

The OMS offers all of the services needed for a suitable organization performance. These services are classified as: structural services, which modify the structural and normative organization specification; and dynamical services, which allow agents to enter or leave the organization dynamically, and to adopt roles. The complete list of OMS services is detailed in Table 2. These services are briefly described as follows.

3.2.1. Structural services

The OMS provides a group of services for registering or deregistering structural components, specific roles, norms and units. It also offers services for reporting on these components.

As previously explained, a role represents a position within the unit in which it is defined. It is related to some interaction norms, imposed by the unit structure and its specific position within the unit, as well as to some behaviour norms, which specify its functionality (services that it needs and offers), restrict its actions (prohibition, obligations and permissions) and establish the consequences of these norms (sanctions and rewards).

Therefore, a norm indicates the obligations, permissions and prohibitions of roles related to service registry, request and fulfilment; service composition or quality of service results. In this way, a norm defines restrictions that cannot be expressed by means of service preconditions or post-conditions.

Finally, a unit represents groups of agents and establishes the topological structure of the system. It is also a recursive concept, so units can be defined within other units. It enables the representation of organizational structures like hierarchy, matrix, coalition, etc. (Argente et al., 2007). Furthermore, it indicates what the structural positions of the system are (i.e. member, supervisor, subordinate), as well as the relationships among these positions imposed by the structure.

The OMS establishes a hierarchy of roles, so any agent that plays a specific role is enabled to request or offer services related to superior hierarchical roles, provided that organizational norms do not explicitly forbid it.

Through the publication of the structural services, the OMS allows the modification of some aspects related to the organization structure, norm and functionality at execution time. For example, a specific agent of the organization can be allowed to add new norms, roles or units during system execution. These types of services should be restricted to the internal roles of the system, which have a level of permission high enough to perform these kinds of operations (i.e. supervisor role). Moreover, these services might not be published in the SF in some specific applications in which the system structure must not be dynamically modified.

Optionally, more complex services for updating organization components can be offered by means of the composition of the services above. For example, a complex service which offers the inclusion of a new role indicating its name attributes and related norms; or a complex service for unit creation which allows the creation of an empty unit with its associated norms and roles. Moreover, services for modifying component features might also be offered. For example, a service for changing the visibility value of a specific role.

Another type of structural service is information services, which provide specific information on all of the components of the organization. They might be restricted to some internal roles of the system. Furthermore, if the OMS is the only component which uses those services, then they are not directly published in the SF.

3.2.2. Dynamic services

The OMS offers a set of basic services for dynamic role adoption and the entry/exit of unit members, which are not directly accessible to agents, but are combined through compound services.

The OMS also offers a set of compound services that can be used by agents for adopting roles, leaving them, and applying sanctions.

As previously explained, the OMS is responsible for managing the life-cycle of the organizations. Thus, it includes meta-services for defining structural components of organizations, i.e. roles, units and norms. These structural components could be dynamically modified over the lifetime of the organization. Moreover, the OMS includes services for creating new organizations (i.e. creating new units), admitting new members within those organizations (i.e. acquiring roles) and resigning members (i.e. expulsing or leaving roles). The management of the agent life-cycle is done by the PK component, which is explained in the following section.

Table 2
OMS meta-services.

Type	Subtype	Meta-service	Description
Structural	Registration	RegisterRole	Creates a new role within a unit
		RegisterNorm	RegisterNorm includes a new norm within a unit
		RegisterUnit	Creates a new unit within a specific organization
		DeregisterRole	Removes a specific role description from a unit
		DeregisterNorm	Removes a specific norm description
		DeregisterUnit	Removes a unit from an organization
	Information	InformAgentRole	Indicates roles adopted by an agent
		InformMembers	Indicates entities that are members of a specific unit
		QuantityMembers	Provides the number of current members of a specific unit
		InformUnit	Provides unit description
		InformUnitRoles	Indicates which roles are the ones defined within a specific unit
		InformRoleProfiles	Indicates all profiles associated to a specific role
		InformRoleNorms	Provides all norms addressed to a specific role
Dynamic	Basic	RegisterAgentRole	Creates a new <entity, unit, role> relationship
		DeregisterAgentRole	Removes a specific <entity, unit, role> relation
	Compound	AcquireRole	Requests the adoption of a specific role within a unit
		LeaveRole	Requests to leave a role
		Expulse	Forces an agent to leave a specific role

3.3. Platform Kernel

The Platform Kernel (PK) is in charge of providing the usual services required in a multi-agent platform. Therefore, it is responsible for managing the life-cycle of the agents included in the different organizations, and it also makes it possible to have a communication channel (incorporating several message transport mechanisms) to facilitate interaction among entities. The PK also provides safe connectivity and the mechanisms necessary for allowing multi-device interconnectivity.

A previous security mechanism is implemented for some of the services described below, which permits the management of who can invoke each service and over whom. For example, the supervisor of an organization may have the option of creating new agents within its organization. For this, the agent Register Service should be invoked at the platform kernel level.

The services offered must be FIPA legacy, with some modifications. The PK services needed in a THOMAS infrastructure are classified into four types as shown in Table 3: (i) Registration: services for adding, modifying and removing native agents from the platform. (ii) Discovery: services for getting information about the native agents active in the platform. (iii) Management: services for controlling the activation state of native agents in the platform. (iv) Communication: services for communicating with agents in the platform and outside.

The THOMAS proposal does not pursue the development of a new multi-agent platform kernel. The services required are a sub-

Table 3
PK services.

Type	Meta-service	Description
Registration	Register	Registers a new agent in the platform
	Deregister	Eliminates an agent registration
	Update register	Modifies the information appearing in an agent register (except the agent name)
Discovery	Agent Search	Requests information from a registered agent in the platform
	Get description	Obtains the platform description
Management	Suspend	Suspends the execution of a specific agent
	Activation	Activates the execution of an agent that is currently suspended
Communication	Send	Sends a message to any agent in the platform or outside

Table 4
Initial content of UnitList for structural specification of the system.

UnitName	ParentUnit	Goal	Type
Virtual (world)	–	–	Flat
HomeCare	Virtual	HomeCare	Congregation
HCServiceUnit	HomeCare	HomeCareService	Flat
LocationUnit	HomeCare	HomeCareLocation	Flat
AlertUnit	HomeCare	HomeCareAlert	Flat

Table 5
Initial content of RoleList.

RoleName	inUnit	Accesibility	Position	Inheritance
Patient	HomeCare	Public	Member	–
Doctor	HomeCare	Public	Member	–
Provider	HomeCare	Public	Member	–
Family	HomeCare	Private	Member	–
HCServicePatient	HCServideUnit	Public	Member	Patient
HCServiceProvider	HCServideUnit	Public	Member	Provider
HCServiceDoctor	HCServideUnit	Public	Member	Doctor
LocationPatient	LocationUnit	Public	Member	Patient
LocationProvider	LocationUnit	Public	Member	Provider
LocationDoctor	LocationUnit	Public	Member	Doctor
AlertPatient	AlertUnit	Public	Member	Patient
AlertProvider	AlertUnit	Public	Member	Provider
AlertDoctor	AlertUnit	Public	Member	Doctor

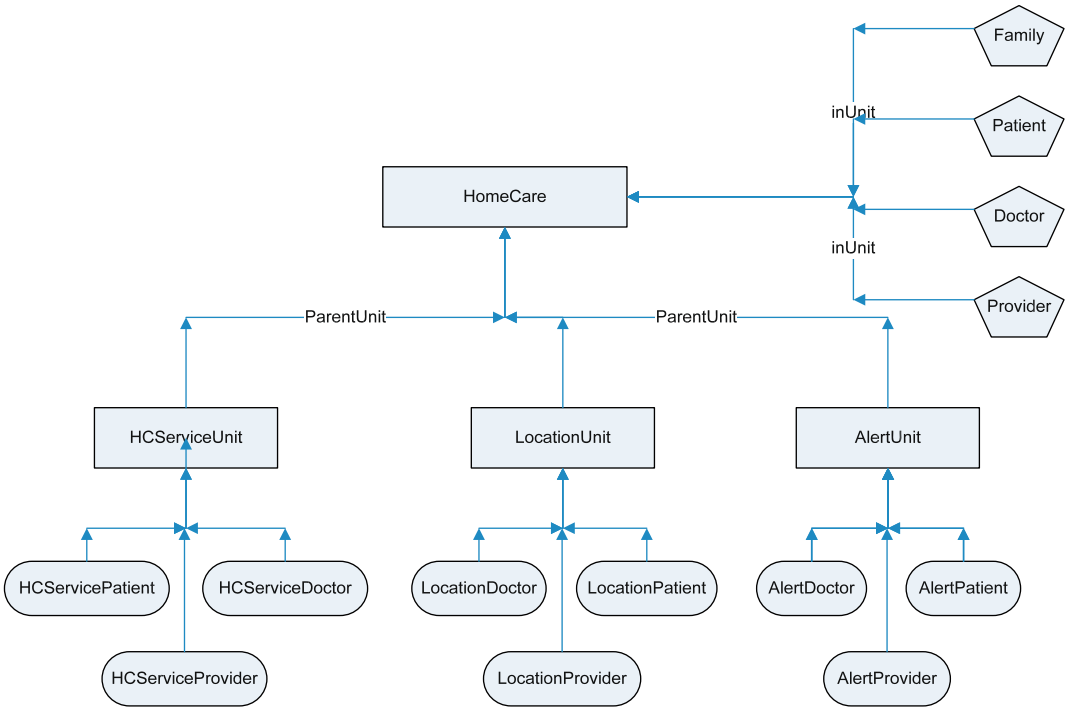

Fig. 2. Home Care structure (units and roles).

set of the services typically offered by diverse, well-known agent platforms. Therefore, PK functionalities can be provided by any agent platform which offers the minimum services mentioned above. Moreover, the chosen agent platform must offer an appropriate communication mechanism which at the very least offers FIPA compatibility.

In summary, this section has described the main components of the THOMAS architecture focusing on their functionalities and

Table 6
Service profiles for the HomeCare system.

Profiles of HCServiceUnit		
Service: OpenCloseDoor **UnitID**: HCServiceUnit **Inputs**: iddoor: string operation: string	**ProfileID**: OpenCloseDoorPF **ClientRole**: HCServicePatient **Outputs**: [door ok] iddoor: string state: string	**Description**: Open or close a door **ProviderRole**: HCServiceProvider **Outputs**: [not ok door] error
Service: OnOffLight **UnitID**: HCServiceUnit **Inputs**: idlight: string operation: string	**ProfileID**: OnOffLightPF **ClientRole**: HCServicePatient **Outputs**: [light ok] idlight: string state: string	**Description**: On or off a light **ProviderRole**: HCServiceProvider **Outputs**: [not ok light] error
Service: LockUnlockAccess **UnitID**: HCServiceUnit **Inputs**: idaccess: string operation: string	**ProfileID**: LockUnlockAccessPF **ClientRole**: HCServicePatient **Outputs**: [access ok] idaccess: string state: string	**Description**: Lock or unlock an access **ProviderRole**: HCServiceProvider **Outputs**: [not ok access] error
Service: OnOffHeating **UnitID**:HCServiceUnit **Inputs**: idheating: string operation: string temperature: string	**ProfileID**: OnOffHeatingPF **ClientRole**: HCServicePatient **Outputs**: [heating ok] idheating: string state: string temperature: string	**Description**: On or off the heating **ProviderRole**:HCServiceProvider **Outputs**: [not ok heating] error
Service: OnOffAirCond **UnitID**: HCServiceUnit **Inputs**: idair: string operation: string temperature: string	**ProfileID**: OnOffAirCondPF **ClientRole**: HCServicePatient **Outputs**: [aircond ok] idair: string state: string temperature: string	**Description**: On or off the air conditioning **ProviderRole**: HCServiceProvider **Outputs**: [not ok aircond] error
Service: UpDownBlind **UnitID**:HCServiceUnit **Inputs**: idblind: string operation: string	**ProfileID**: UpDownBlind PF **ClientRole**: HCServicePatient **Outputs**: [blind ok] idblind: string state: string	**Description**: Up or down a blind **ProviderRole**: HCServiceProvider **Outputs**: [not ok blind] error
Profiles of LocationUnit		
Service: SearchPatient **UnitID**: LocationUnit **Inputs**: idhome: string idpatient: string	**ProfileID**: SearchPatientPF **ClientRole**: LocationProvider, LocationDoctor, Family **Outputs**: [patient ok] name: string location: string	**Description**: Search for a patient in their home **ProviderRole**: LocationProvider **Outputs**: [not in home] error
Service: IdentifyPatient **UnitID**: LocationUnit **Inputs**: idpatient: string	**ProfileID**: SearchPatientPF **ClientRole**: LocationProvider **Outputs**: [patient ok] location: string date: time	**Description**: Identify a patient **ProviderRole**: LocationProvider **Outputs**: [not ok patient] error
Profiles of AlertUnit		
Service: SendSms **UnitID**: AlertUnit **Inputs**: sms: string phone: string	**ProfileID**: SendSmsPF **ClientRole**: AlertProvider, AlertDoctor, Family, AlertPatient **Outputs**: [phone ok] idsms: string state: string	**Description**: Send a SMS **ProviderRole**: AlertProvider **Outputs**: [not ok phone] error
Service: SendMms **UnitID**: AlertUnit **Inputs**: mms: data mms phone: string	**ProfileID**: SendMmsPF **ClientRole**: AlertProvider, AlertDoctor, Family,AlertPatient **Outputs**: [phone ok] idmms: string state: string	**Description**: Send a MMS. **ProviderRole**: AlertProvider **Outputs**: [not ok phone] error
Service: ProcessSms **UnitID**: AlertUnit **Inputs**: sms: string phone: string	**ProfileID**: ProcessSmsPF **ClientRole**: AlertProvider, AlertDoctor, Family, AlertPatient **Outputs**: [sms ok] sms: string phone: string	**Description**: Process a SMS **ProviderRole**: AlertProvider **Outputs**: [not ok sms] error
Service: SendMail **UnitID**: AlertUnit **Inputs**: email: string subject: string body: string adj: data	**ProfileID**: SendMailPF **ClientRole**: AlertProvider, AlertDoctor, Family, AlertPatient **Outputs**: [email ok] idsms: string state: string	**Description**: Send a mail **ProviderRole**: AlertProvider **Outputs**: [not ok email] error

describing the services which form the interface with each one of these components. The following section presents a detailed example employing the services described above.

4. Applying THOMAS to Home Care

Home Care facilitates the interconnection between dependent people and their environment and medical staff (doctors, nurses and personal assistant), delimiting services that each one can request or offer. The system controls which services must be provided by each agent. The internal functionality of these services is the responsibility of provider agents. However, the system imposes some restrictions regarding service profiles, service request orders and service results.

A description of the structure elements of the Home Care organization is detailed below. Then, in Section 4.2, a dynamical usage of the organization is explained, providing different execution scenarios.

4.1. Cases-study organization structure

This case study is modelled as an organization (HomeCare) within which there are three organizational units (HCServiceUnit, LocationUnit and AlertUnit) each of which represents a group of agents. The units are dedicated to Home Care services, location services and alert services, respectively.

Four kinds of roles can interact in the Home Care case study: patient, doctor, family and provider. The Patient role requests system services. The Patient role can request home automation services through the alert service. This role also communicates with the medical service or family, and other services in the home. The Doctor role consists of three specialized sub-roles according to the type of communication with each unit (HCServiceDoctor, LocationDoctor and AlertDoctor). The Provider role is in charge of performing services. A provider agent offers home automation, location or alert search services. The Provider role also consists of three specialized sub-roles: HCServiceProvider, LocationProvider and AlertProvider. Finally, the Family role provides advanced consulting services. It represents the family where relatives can check the patient status. As it is a private role, agents are not able to access this Family role. Fig. 2 shows the Home Care structure, with its units, roles and interrelationships

The OMS component stores the list of defined units (UnitList, Table 4) and the list of roles (RoleList, Table 5) internally.

The HomeCare organization offers three services: Automation, Location and Alert service. These services are specialized for each unit. A brief description of the profiles of each of these services is shown in Table 6.

All of these services have been registered in the SF component of the THOMAS platform. In this example, we have assumed that the Home Care system does not initially have an agent registered as a service provider, any agent acting as a patient, or any agent acting as a doctor. Therefore, this system has initially only been structured as a regulated space in which agents might enter to provide or request any of those specific services registered in the SF component. Consequently, in the initial state of the system, there is no provider attached to the HomeCare services.

In the following section, different scenarios are considered in which patient and/or provider and/or doctor agents enter and participate in the system.

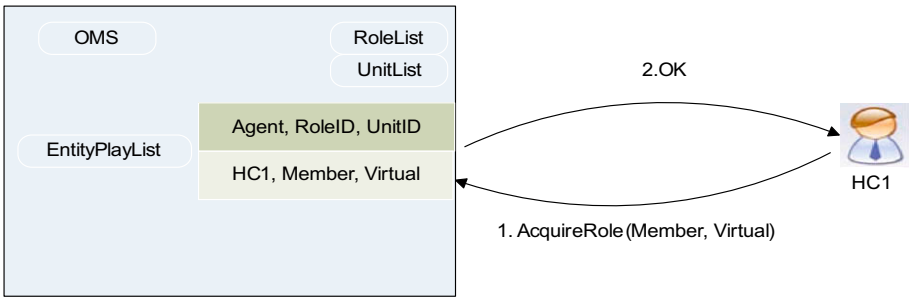

Fig. 3. Example of agent registration.

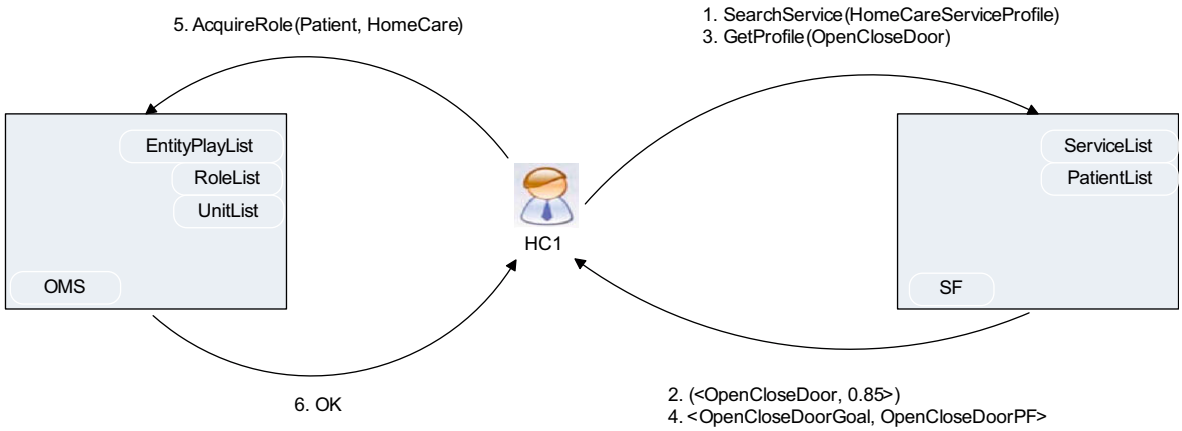

Fig. 4. Example of patient registration.

4.2. System dynamics

In this section, the use of THOMAS meta-services in the HomeCare case study is presented in detail. System dynamics are shown through the specification of different scenarios: (i) an agent joins the THOMAS platform; (ii) a Patient is registered; (iii) the Patient is registered as a PatientLocation; (iv) a new service implementation is defined; (v) new service patients are included; (vi) a doctor is registered; (vii) services are requested; (viii) malicious agents are expulsed; (ix) a Provider is deregistered; and (x) a new unit is created.

4.2.1. Agent registration

This scenario details the sequence of services that an agent should request in order to join the THOMAS platform. In Fig. 3, HC1 is an agent that represents a home automation provider. Its functionality allows it to offer services belonging to its company. Agents join THOMAS platform by placing a request with the OMS for membership of the virtual organization, using *AcquireRole* service (Fig. 3, message 1).

The OMS checks all restrictions (existence of unit and role identifiers, role compatibility, etc.) and registers HC1 agent as a new member of the THOMAS platform. The OMS makes use of *RegisterAgentRole* service to add this agent-role adoption to *EntityPlayList*.

4.2.2. Patient registration

In this scenario, the process for registering a new Patient is detailed (Fig. 4). Once HC1 has been registered as a member of the THOMAS platform, it asks SF which defined services have a profile similar to its own "Home Care service". This request is carried out using the SF *SearchService* (Fig. 4, message 1), in which *HomeCareServiceProfile* corresponds to the profile of the patient search service implemented by HC1.

The SF returns service identifiers that satisfy these search requirements together with a *ranking value* for each service (message 2). *Ranking value* indicates the degree of suitability between a service and a specified service purpose. Then HC1 executes *GetProfile* (message 3) in order to obtain detailed information about the *OpenCloseDoor* service. Service outputs are "service goal" and "profile" (message 4). The *OpenCloseDoor* profile specifies that service providers have to play a *Patient* role within *HCService*. Thus, HC1 requests the *AcquireRole* service from the OMS in order to acquire this patient role (message 5). *AcquireRole* service is carried out

searchPatientProcess	
process:hasInput =	?idhome
	?idpatient
process:hasOutput =	?name
	?location
process:hasDoctor =	locationDoctor
	process:TheDoctor
process:performedBy =	locationDoctor
	ch1.thomas
	process:TheServer
service:describes =	searchPatientService

Fig. 7. Example of SearchPatient process.

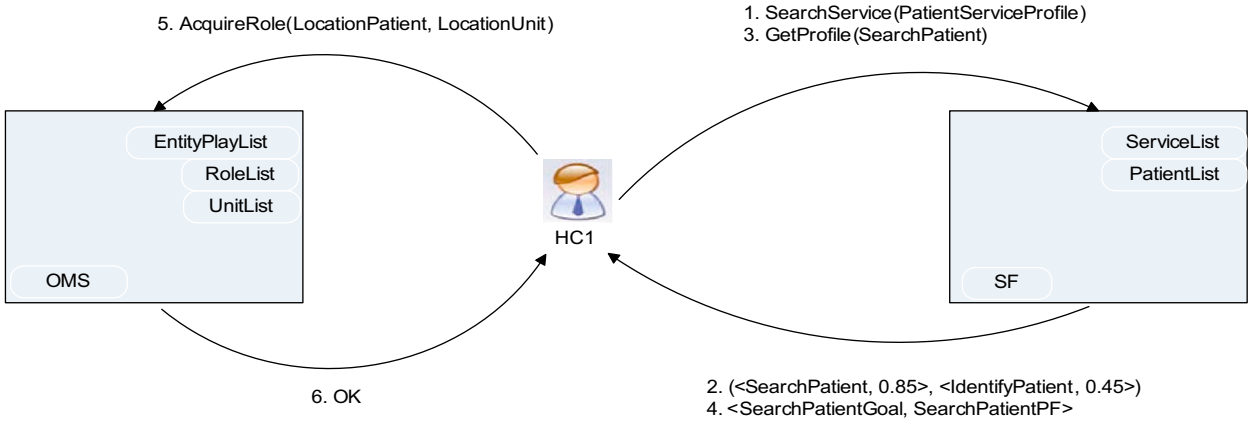

Fig. 5. Example of LocationPatient registration.

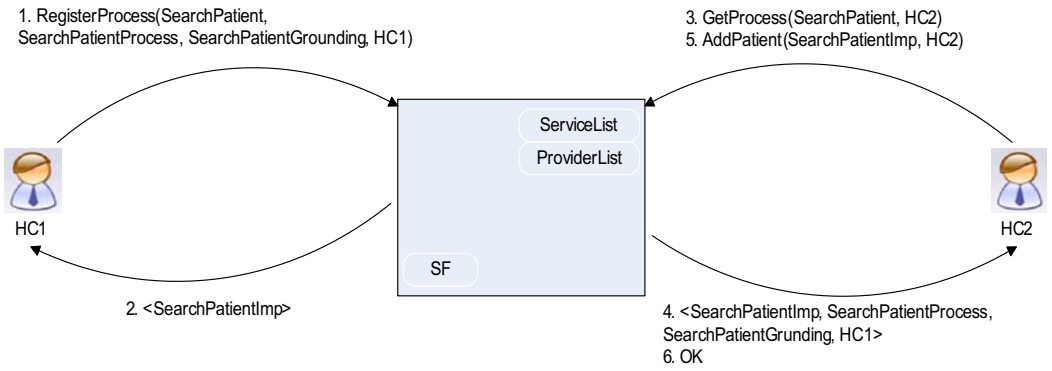

Fig. 6. Example of service implementation and patient registration.

successfully (message 6), because *HCService* is accessible from *Virtual* organization, thus HC1 is registered as a *Patient*.

4.2.3. LocationPatient registration

Once the "patient registering" process has been detailed, the registration of a location patient is illustrated (Fig. 5). HC1 is able to provide a search patient in the Home Care domain. Therefore, it asks SF whether an available service description with a closer profile exists, requesting *SearchService* from SF as before (Fig. 5, message 1).

In this case, SF returns both *SearchPatient* and *IdentifyPatient* since these two services are visible within *HomeCare* unit. As indicated in the service result, *IdentifyPatient* service is more appropriate for HC1 functionality. Therefore, HC1 requests information about this service from SF, using *GetProfile* (message 3). The *IdentifyPatient* profile returned (message 4) specifies that service providers must place *LocationPatients* within *LocationUnit*. Then HC1 places a request with OMS to adopt *LocationPatient* role (message 5). *AcquireRole* service is carried out successfully (message 6), so HC1 agent is registered as a *LocationProvider*.

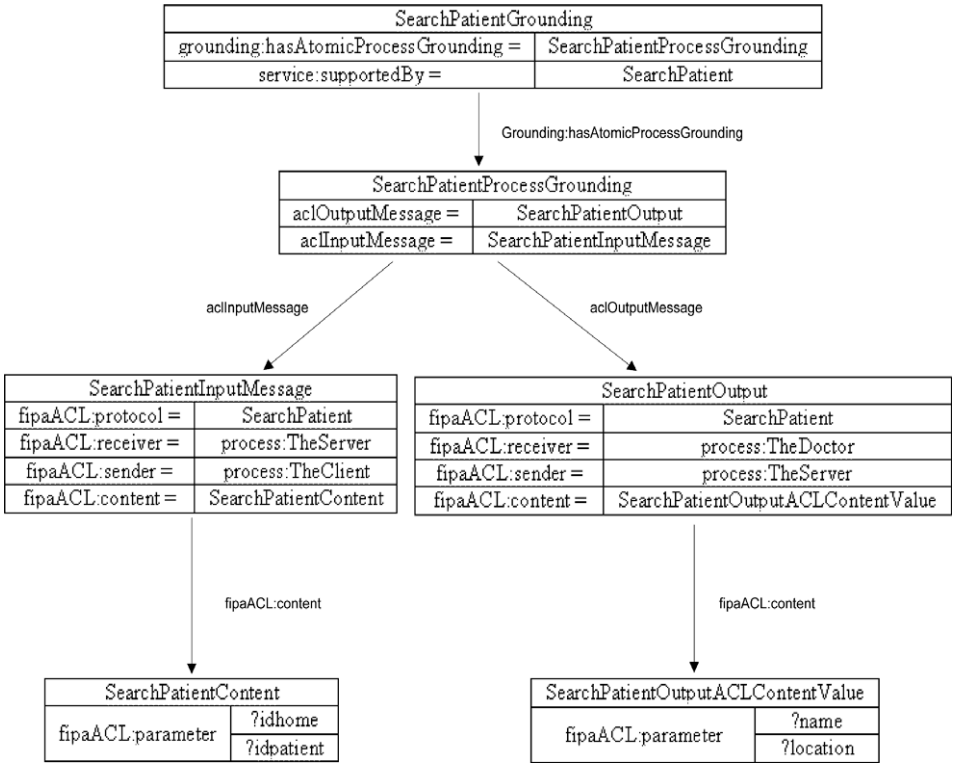

Fig. 8. Example of SearchPatient grounding.

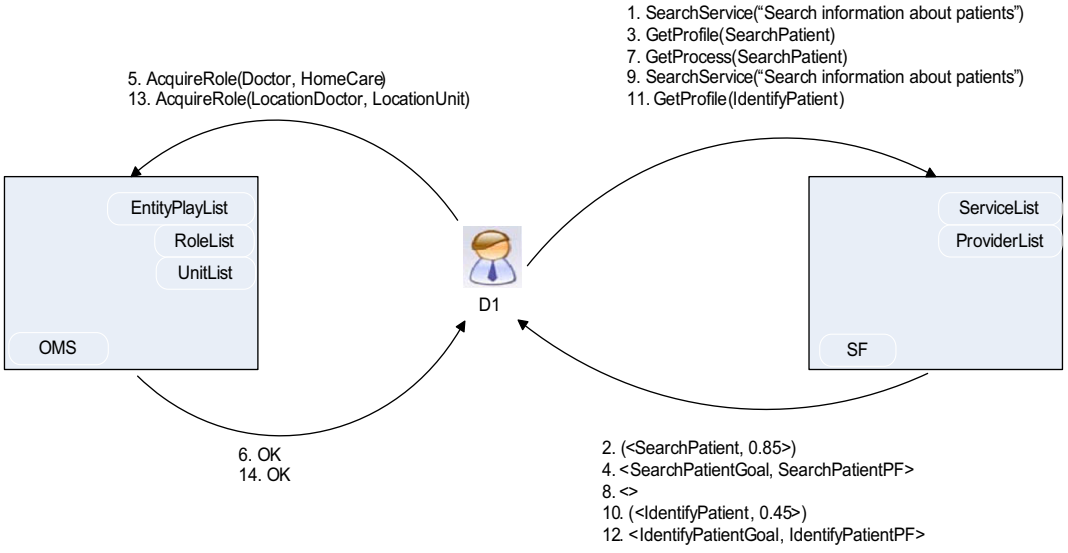

Fig. 9. Example of doctor registration.

4.2.4. Registering a new service implementation

In this example, the sequence of actions that allows an agent to register its own implementation of a service is explained in detail (Fig. 6). In this case, HC1 has already adopted *LocationPatient* role and is interested in providing its own implementation of *SearchPatient* service. Therefore it registers itself as service provider in SF, employing *RegisterProcess* service (Fig. 6, message 1), in which *SearchPatientProcess* and *SearchPatientGrounding* correspond to service process and grounding, respectively. Fig. 7 contains the specification of *SearchPatient* service process. Fig. 8 shows *SearchPatient* grounding. This type of grounding specifies how a service can be requested by means of sending ACL messages.

4.2.5. Adding new service patient

This section demonstrates how another *LocationPatient* (HC2) is added to a service patient list (Fig. 6). In this example, HC2 has already adopted the *LocationPatient* role and HC1 has been registered as a provider of the *SearchPatient* service. HC2 initially asks what the registered implementations of *SearchPatient* service are (Fig. 6, message 3). SF provides a list that contains service implementations details (message 4). HC2 decides to employ the same service process as HC1, so it uses *AddPatient* service in order to request its inclusion as a provider of *SearchPatient* service (Fig. 6, message 5).

4.2.6. Doctor registration

The following scenario shows the set of service calls for registering new agents as service doctors within the *HomeCare* (Fig. 9). A new doctor agent D1, which has already been registered in the THOMAS platform, requests *SearchService* from SF to find services of interest (message 1). As a result, D1 obtains *SearchPatientl* service identifier and ranking value (message 2). Then, D1 employs *GetProfile* (message 3), which specifies that service doctor must play *Doctor* Role within *HomeCare* (message 4). Therefore, D1 must acquire *Doctor* Role to demand this service (messages 5 and 6).

Once D1 plays this Doctor Role, it employs *GetProcess* service in order to find out who the service providers are and how this service can be requested (message 7). However, there are no providers for the general SearchPatient service (message 8).

Within the *HomeCare* unit, D1 requests *SearchService* again (message 9). In this case, SF returns *IdentifyPatient* services because both services are accessible from *HomeCare* organization.

D1 demands the profile of *IdentifyPatient* service (using *GetProfile*, message 11), since this service is more appropriate for its needs. Taking the *IdentifyPatient* profile into account (message 12), D1 requests the adoption of *LocationDoctor* role within *LocationUnit* (message13).

4.2.7. Service request

This scenario shows how doctor agents make demands for services (Fig. 10). Once D1 adopts the Doctor Role for *SearchPatient* service, it is allowed to demand services from providers. Assuming that D1 wants to make an information search about patients, it should use *GetProcess* service to obtain the implementations of available services and also its provider identifiers (message 1).

An implementation of *SearchPatient* has been previously registered by HC1 and HC2. After comparing providers of *SearchPatient* service returned in message 2, D1 chooses to make a service request from HC1 agent (message 3). Both ACL contents for requesting and reporting messages are detailed in Table 7.

4.2.8. Agent expulsion

In this scenario, the expulsion of a malicious agent is carried out (Fig. 11). *Provider* agent detects that different doctor agents (D1

Table 7
Service profiles for the HomeCare system.

```
request
(
    :sender d1.thomas
    :receiver hc1.thomas
    :content
    (
      ...
      <idhome
      rdf:datatype="string">3</idhome>
      <idpatient
      rdf:datatype="string">3</idpatient>
      ...
    )
    :in-reply-to
    :language
    :ontology
    :protocol SearchPatient
Inform
(
    :sender hc1.thomas
    :receiver d1.thomas
    :content
    (
      ...
      <>
      <>
      ...
      )
      :in-reply-to
      :language
      :ontology
      :protocol SearchPatient
)
```

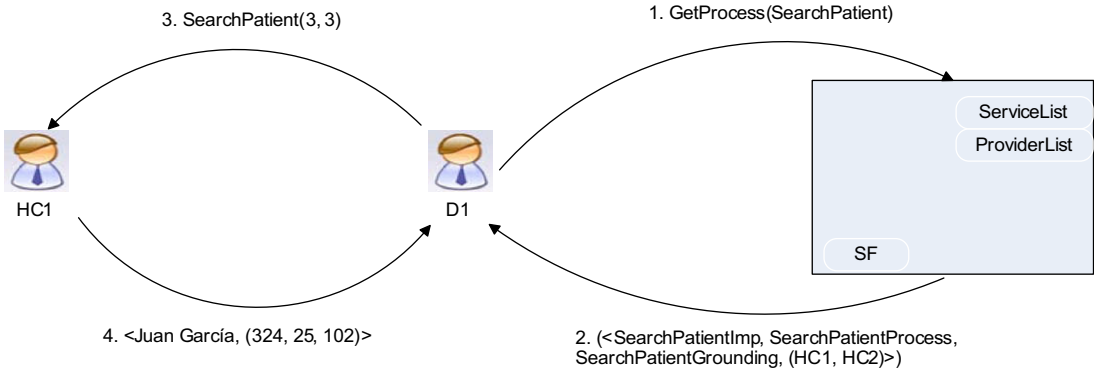

Fig. 10. Example of service request.

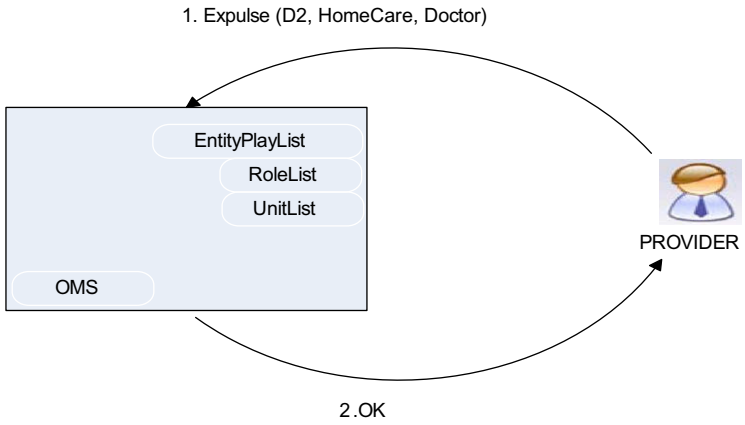

Fig. 11. Example of agent expulsion.

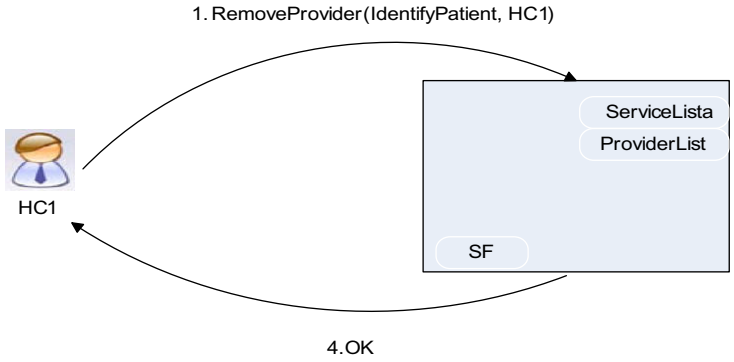

Fig. 12. Example of agent deregistration

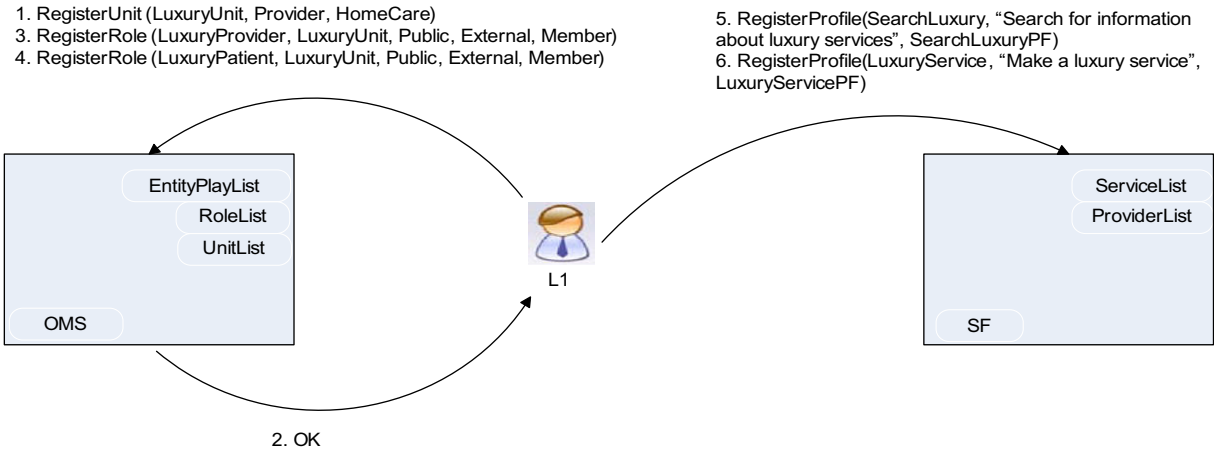

Fig. 13. Example of new unit creation.

and D2) have registered with the same identifier number. It consults its database and determines that D2 has been employing an identifier number that does not belong to it. D2 is punished for its fraudulent behaviour and is expelled from *HomeCare*. *Provider* requests the expulsion of D2 from OMS employing *Expulse* service (message 1).

4.2.9. Provider deregistration

Below, the process of a service provider deregistering is described (Fig. 12). HC1 loses connection with its internal database. As a result, it is temporarily unable to provide services. Therefore HC1 deregisters itself as provider of *IdentifyPatient* using *RemoveProvider* (Fig. 12, messages 1 and 2).

HC1 is deleted from the service provider list. Nevertheless, as it continues playing the *Provider* role, HC1 will be able to register itself as service provider if it recovers its functionality.

4.2.10. Unit creation

This last scenario illustrates the creation of new units within *HomeCare* (Fig. 13). Agent L1 represents a luxury Home Care

Table 8
Final content of SF ServiceList after execution of all scenarios.

ServiceFacilitator		
ServiceID	Profile	Providers
OpenCloseDoor	OpenCloseDoorPF	
SearchPatient	SearchPatientPF	SearchPatient(HC1, HC2)
IdentifyPatient	IdentifyPatientPF	
SearchLuxury	SearchLuxuryPF	

Table 9
Final content of OMS internal lists after execution of all scenarios.

UnitList			
UnitName	ParentUnit	Goal	Type
Virtual(world)	–	–	Flat
HomeCare	Virtual	HomeCare	Congregation
HCServiceUnit	HomeCare	HomeCareService	Flat
LocationUnit	HomeCare	HomeCareLocation	Flat
AlertUnit	HomeCare	HomeCareAlert	Flat
LuxuryUnit	HomeCare	HomeCareLuxury	Flat

RoleList				
RoleName	inUnit	Accessibility	Position	Inheritance
Patient	HomeCare	Public	Member	–
Doctor	HomeCare	Public	Member	–
Provider	HomeCare	Public	Member	–
Family	HomeCare	Private	Member	–
HCServicePatient	HCServideUnit	Public	Member	Patient
HCServiceProvider	HCServideUnit	Public	Member	Provider
HCServiceDoctor	HCServideUnit	Public	Member	Doctor
LocationPatient	LocationUnit	Public	Member	Patient
LocationProvider	LocationUnit	Public	Member	Provider
LocationDoctor	LocationUnit	Public	Member	Doctor
AlertPatient	AlertUnit	Public	Member	Patient
AlertProvider	AlertUnit	Public	Member	Provider
AlertDoctor	AlertUnit	Public	Member	Doctor
LuxuryPatient	LuxuryUnit	Public	Member	Patient
LuxuryProvider	LuxuryUnit	Public	Member	Provider
LuxuryDoctor	LuxuryUnit	Public	Member	Doctor

EntityPlayList		
Entity	Unit	Role
Doctor	HomeCare	Doctor
HC1	LocationUnit	LocationPatient
HC2	LocationUnit	LocationPatient
D1	LocationUnit	LocationDoctor
D2	HomeCare	Doctor
L1	LuxuryUnit	LuxuryPatient

company which specializes in luxury services. It is interested in providing information and very luxurious services. This L1 has already adopted the *Provider* role within *HomeCare* unit. However, since the services offered within *LocationUnit* and *AlertUnit* are specialized in location and alert domains, L1 decides to create a new unit (*LuxuryUnit*) within *HomeCare* (Fig. 13, message 1). This new unit will be focused on luxury Home Care. Once the OMS informs L1 about the successful creation of the new unit, L1 defines luxury-specific roles and services (messages 3 to 6). Finally, luxury agents would be able to adopt the *LuxuryProvider* role and start offering services to patient agents.

After all these scenarios, several agents have joined the THOMAS platform and offer or request services within this system. Table 8 shows the evolution of the *ServiceList* content, in which all of the new elements and relationships included as a result of the execution of these scenarios are emphasized. Similarly, OMS internal tables have also been updated (Table 9).

5. Results and conclusions

An important issue in the development of real open multi-agent systems is to provide developers with methods, tools and appropriate architectures which support all of the requirements of these kinds of systems. This paper has studied this problem by proposing a case study on abstract architecture for the development of virtual organizations. Moreover, the proposal aims to instigate the total integration of two promising technologies, multi-agent systems and service-oriented computing, as the foundation of such virtual organizations. In THOMAS architecture, agents can transparently offer and invoke services from other agents, virtual organizations or entities; additionally, external entities can interact with agents through the use of the services offered.

A case study example was employed as an illustration of the usage of THOMAS components and services. Dynamic applications are also developed with the same architecture. In this way, examples of THOMAS service calls have been shown through several scenarios, along with the evolution of different dynamic virtual organizations. The participative approach presented in this work is also applicable to other knowledge production tasks as software development, especially in analysis and design phases. Nevertheless, further validation is needed to assess the usefulness of the architecture in different scenarios. We are also conducting tests on the impact of the number of agents in the overall effectiveness of the model. However, interaction processes are not completely independent from one another. Therefore, the participation of agents in the system can be dynamic, so that an agent can change its membership to some other system if the knowledge produced by the agent affects interaction processes carried out in that system.

Acknowledgements

This work was supported by the Spanish Ministry of Science and Technology project TIN2006-14630-C03-03 and the JCyL SA071A08 Project.

References

Anastasopoulos, M., Niebuhr, D., Bartelt, C., Koch, J., & Rausch, A. (2005). Towards a reference middleware architecture for ambient intelligence systems. In *ACM conference on object-oriented programming, systems, languages, and applications*.

Angulo, C., & Tellez, R. (2004). Distributed intelligence for smart home appliances. In *Tendencias de la minería de datos en España* (pp. 1–12). Barcelona: Red Española de Minería de Datos.

Argente, E., Palanca, J., Aranda, G., Julian, V., Botti, V., García-Fornes, A., et al. (2007) (pp. 236–245). *Lecture notes in artificial intelligence* (Vol. 4696). Berlin: Springer.

Augusto, J. C., & McCullagh, P. (2007). Ambient intelligence: Concepts and applications. *International Journal on Computer Science and Information Systems, 4*(1), 1–28.

Bahadori, S., Cesta, A., Grisetti, G., Iocchi, L., Leone, R., Nardi, D., et al. (2003). RoboCare: Pervasive intelligence for the domestic care of the elderly. *Artificial Intelligence, 1*(1), 16–21.

Brogi, A., Corfini, S., & Popescu, R. (2005). Composition-oriented service discovery. In *Lecture notes in computer sciences* (pp. 15–30). Berlin: Springer.

Camarinha-Matos, L., & Afsarmanesh, H. (2004). TeleCARE: Collaborative virtual elderly care support communities. *The Journal on Information Technology in Healthcare, 2*(2), 73–86.

Carrascosa, C., Bajo, J., Julian, V., Corchado, J. M., & Botti, V. (2008). Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems With Applications, 34*(1), 2–17.

Cesta, A., Bahadori, S., Cortellesa, G., Grisetti, G., & Giuliani, M. (2003). The RoboCare project, cognitive systems for the care of the elderly. In *Proceedings of international conference on aging, disability and independence (ICADI'03). Washington, DC, USA*.

Costa-Font, J., & Patxot, C. (2005). The design of the long-term care system in Spain: Policy and financial constraints. *Social Policy and Society, 4*(1), 11–20.

Corchado, J. M., & Laza, R. (2003). Constructing deliberative agents with case-based reasoning technology. *International Journal of Intelligent Systems, 18*, 1227–1241.

Corchado, J. M., Bajo, J., de Paz, Y., & Tapia, D. (2008). Intelligent environment for monitoring alzheimer patients, agent technology for health care. *Decision Support Systems, 34*(2), 382–396.

Decker, K., & Li, J. (1998). Coordinated hospital patient scheduling. In *Proceedings of the 3rd international conference on multi-agent systems (ICMAS'98)* (pp. 104–111). IEEE Computer Society.

Klusch, M., Fries, B., & Sycara, K. (2006). Automated semantic web service discovery with owls-mx. In *Proceedings of 5th international conference on autonomous agents and multi-agent systems* (pp. 915–922). Hakodate, Japan.

Lanzola, G., Gatti, L., Falasconi, S., & Stefanelli, M. (1999). A framework for building cooperative software agents in medical applications. *Artificial Intelligence in Medicine, 16*(3), 223–249.

Meunier, J. A. (1999). A virtual machine for a functional mobile agent architecture supporting distributed medical information. In *Proceedings of the 12th IEEE symposium on computer-based medical systems* (pp. 177). Washington, DC: IEEE Computer Society.

Miksch, S., Cheng, K., & Hayes-Roth, B. (1997). An intelligent assistant for patient health care. In *Proceedings of the 1st international conference on autonomous agents, California, USA* (pp. 458–465). New York: ACM.

Nealon, J. L., & Moreno, A. (2003). Applications of software agent technology in the health care domain. In A. Moreno & J. L. Nealon (Eds.). *Whitestein series in Software Agent Technologies* (Vol. 212). Basel, Germany: Birkhäuser Verlag AG.

Pecora, F., & Cesta, A. (2007). Dcop for smart homes: A case study. *Computational Intelligence, 23*(4), 395–419.

Sycara, K., Widoffand, S., Klusch, M., & Lu, J. (1982). Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Journal on Autonomous Agents and Multi-Agent Systems, 5*, 173–203.

Van Woerden, K. (2006). Mainstream developments in ICT: Why are they important for assistive technology? *Technology and Disability, 18*(1), 15–18.

Want, R., Pering, T., Borriello, G., & Farkas, K. (2002). Disappearing hardware. *Pervasive Computing, 1*(1).