DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Informe Técnico / Technical Report

Ref. No: DSIC-II/13/08 **Pages: 31**

Title: An Abstract Architecture for Virtual Organizations: The THOMAS project

Author (s): E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, M. Rebollo

Date: 7-10-2008

Key Words: Multi-agent systems; virtual organizations

VºBº **Author (s):**

Leader of Reasearch Group

An Abstract Architecture for Virtual Organizations: The THOMAS project

E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, M. Rebollo

Departamento de sistemas informáticos y computación
Universidad Politécnica de Valencia
Camino de Vera S/N 46022 Valencia (Spain)
http://gti-ia.dsic.upv.es
vbotti@dsic.upv.es

1 Introduction

In multi-agent system (MAS) field one of the goals is to build systems capable of making decisions in an autonomous and flexible way. Moreover, these systems must cooperate with other systems inside a "society". Due to the technological advances of recent years, the term "society", in which the multi-agent system participates, needs to meet requirements such as: distribution, constant evolution, flexibility to allow members to enter or exit in 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 these requirements define a set of characteristics that can be addressed through the open system paradigm and virtual organisations.

MAS technology allows to cover a broad area of problems. Typically we are talking about systems where there are several entities (Requesters), which may require one or more elements or goals from other different entities (Bidders). As an example, in the area of leisure activities and entertainment, the Requesters would be customers/users/citizens and the Bidders would be companies or company clusters that provide leisure activities, such as cinema, theaters, museums or restaurants. Obviously, the development of this type of systems is complex and, therefore, it is necessary to analyse in detail the intrinsic characteristics of these typical application environments.

The main goal of this work is to obtain a new open multi-agent system architecture consisting of a related set of modules that are suitable for the development of systems applied in environments such as those above raised. This requires, as a first step, the high-level design of a related abstract architecture. In this design will be determined, at a high-level of abstraction, all components needed to cover all the characteristics and needs for systems of this kind. This new architecture has been called THOMAS (Me*TH*ods, Techniques and Tools for *O*pen *M*ulti-*A*gent *S*ystems).

Over recent years have appeared several works trying to solve the problem of integrating the multi-agent system paradigm and the service-oriented computing paradigm. It is obvious, that there are many similarities among them.

Both paradigms try to offer solutions for the development of complex and adaptive systems in distributed open environments. In this line, integrating these technologies is possible to model autonomous and heterogeneous computational entities in dynamic and open environments. Such entities may be reactive, proactive and with the ability to communicate in a flexible way with other entities. One of the existing proposals works in the line to create links, as a gateway, between the two directions. The proposed solution tries to communicate agents and web services in a transparent, but independent, way. This is the line of the Agent and Web Services Interoperability (AWSI) IEEE FIPA Working Group (http://www.fipa.org/subgroups/AWSI-WG.html). Although interesting, our proposal tries to go beyond, raising a total integration of both technologies. So agents can offer and invoke services in a transparent way to other agents or entities, as well as external entities can interact with our agents through the use of the offered services.

This paper is structured as follows: section 2 presents the proposed architecture model as well as a description of the services offered by each one of the modules that constitute the reference model; section 3 shows an implementation example of the new possibilities provided by this type of architecture; finally some conclusions and future lines of work are shown in section 4.

2 Architecture Model

THOMAS architecture consists basically of a set of modular services. THOMAS feeds initially of the FIPA architecture expanding its capabilities. The agents have access to the infrastructure offered by THOMAS through a range of services including on different modules or components. The main components of THOMAS are the following (Figure 2):

- Service Facilitator (SF), this component offers simple and complex services to the active agents and organizations. Basically, its functionality is like a yellow page service and a service descriptor in charge of providing a green page service.
- Organization Manager Service (OMS), it is mainly responsible of the management of the organizations and their entities. Thus, it allows the creation and the management of any organization.
- Platform Kernel (PK), it maintains basic management services for an agent platform.

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

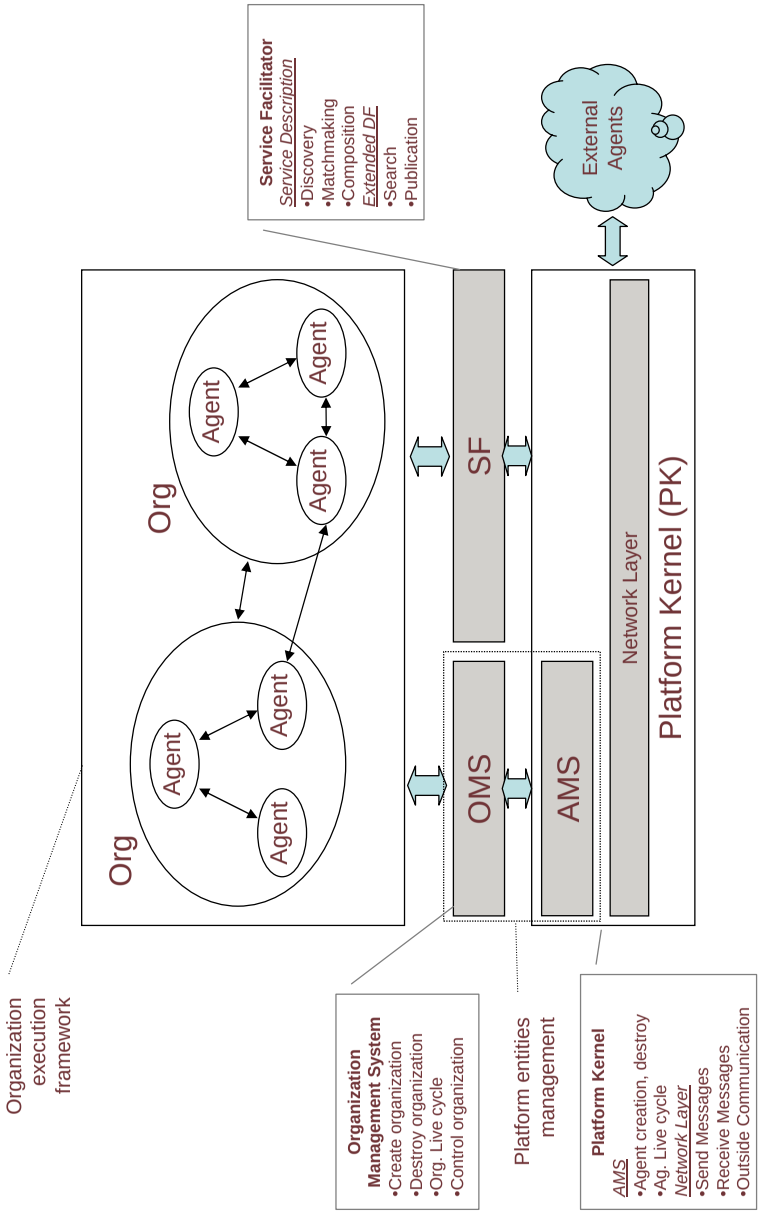

Fig. 1. THOMAS Architecture

2.1 Service Facilitator

The SF is a mechanism and support by which the organization and agents can, at the same time, offer and discover 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 searching for a given service profile or searching by the goals that can be fulfilled executing the service. This is done using the matchmaking and service composition mechanisms that are provided by the SF. The SF acts also as a yellow pages manager and in this way it can also find which entities provide a given service.

A service offers some capacities, each of which to fulfil a given goal. The service may have some pre-conditions that have to be true before the service can be executed. It exchange one or more input and output messages. Before a successful service execution it has some effects on its environment. Moreover, there could be additional parameters, which are independent of the service functionality (non-functional parameters), such as quality of service, deadlines, and security protocols among other. And finally, the service results can be enhanced using automatic service composition mechanisms (for example, partial matchmaking). To do this the SF maintains the description of the internal processes that are executed when the service is running.

A service represents an interaction of two entities, which are modeled as communications among independent processes. In our case, the Multi-agent Technology provides us with FIPA communication protocols which are well established mechanisms in order to standardize the interactions. In this way, every service has an associated protocol. In those cases in which the service requires the execution of a chain of protocols, the service is marked as "complex". Taking into account that we are working with semantic services, another important data is the ontology used in the service. In this way, when the service description is accessed, any entity will has all the needed information in order to interact with the service and how to make an application that can use the service. Such a description can also be used for pre-compiled services, in which the process model of the service will be, instead of the internal processes of the service, the sequence of the elementary services that will be executed.

The SF entries are service descriptions using the following structure:

<ServiceID, Providers, ServiceGoal, ServiceProfile>
Providers = <ServiceImpID, ProviderIDList, ServiceModel, ServiceGrounding> +
ProviderIDList = ProviderID +

- **ServiceID** is a unique service identifier.
- **Providers** is a set of tuples made up by a Providers identifier list (ProviderIDList), the service process model specification, and the particular instantiation of the service that is provided by these providers.
- **ProviderIDList** maintains a list of identifier of the service providers.

- **ServiceGoal** is a general definition of the goal that can be fulfilled executing the service. It provides a first abstraction level for service composition.
- **ServiceProfile**, specifies what the service does, in way readable for those agents that are searching information (or matchmaking agents which act as searching service agents). This type of representation includes: a description of what the service fulfils, the constraints about its applicability and the quality of service, and the requirements that the clients have to satisfy in order to use the service. The ServiceProfile is specified using the OWL-S standard for service Profile definition augmented with the following attributes:

  providerRole specifies the role of the entity which offers the service. It is optional.

  clientRole specifies the role of the entity which requires the service. It is optional.
- **ServiceModel** specifies to the client how it has to use the service. The ServiceModel details the semantic content for using the service, the situations in which the results are obtained, and, whenever it is required, the step by step processes to get these results. In other words, it specify how to call a service, and what happen when the service executes. The ServiceModel is specified using the OWL-S standard.
- **ServiceGrounding** specifies in details how an agent can access the service. A grounding specifies a communication protocol, the message formats, and other specific details of the service such as the used port to contact the service. The ServiceGrounding is specified using the OWL-S standard augmented with the FIPA protocols.

Besides the parcitular information about the service, all services provided by the SF return a service status (success or error) and an error value in cased of failure. The most general error values are

- Not-found: the specified value for the parameter (provider or service) is not found
- Duplicate: the entry already exists
- Invalid: the structure of the parameter is not correct
- Access: the client has not privileges to invoke the service

The SF provides the following standard services:

1. **RegisterProfile**: it is used when an autonomous entity wants to register a service description. To do this the following structure has to be completed in order to provide the service description.

   ***RegisterProfile*(ServiceID ?sID, ServiceGoal ?sGoal, ServiceProfile ?sProfile)**

   The results of this service can be:

Service Specification					
Name:	RegisterProfile				
Description:	It is a meta-service and is used to register a service description in the SF.				
Supplied by:	SF				
Required by:	any role.				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
ServiceGoal	Defines the service global goal	Yes	String		
ServiceProfile	Specifies the service description	Yes	ServiceProfile-Structure		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
ServiceID	Unique service identifier. It is automatically generated by the SF	No	String		
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	Duplicate, Invalid-Struct, Invali-dAccess	
Precondition					
Pre1:	$\neg\exists S \in SF \mid S.ServiceProfile = ServiceProfile$				
Postcondition					
Post1:	$\exists S \in SF \mid S.ServiceID = ServiceID \wedge S.ServiceProfile = ServiceProfile$				

- ServiceID, which is automatically generated by the SF and Service-Status indicating success when the service was successfully executed. This result implies that the service is publicly available.

2. **RegisterProcess**: it is used when an agent wants to register a particular implementation of a given service. The ID of the service provider entity has to be specified.

***RegisterProcess*(ServiceID ?sID, ServiceModel ?sModel, ServiceGrounding ?sGrounding, EntityID ?ProviderID)**

There could be several providers for the same service implementation. The first time the *RegisterProcess* is called the Provider is specified (EntityID). The other providers can be added or modified calling the *AddProvider* and *RemoveProvider* services.
The results of this service can be:

- Service-status indicating success, if the service was successfully executed. This implies that the service implementation is publicly available.

Service Specification	
Name:	RegisterProcess
Description:	It is a meta-service and is used to register a service particular implementation in the SF.
Supplied by:	SF
Required by:	any role.

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
ServiceID	Specifies the service to which this process corresponds to	Yes	String		
ServiceModel	Specifies how an agent may use the service. That is, how to request the service and what happens when the service is executed	Yes	Service-Model-Structure		
ServiceGrounding	Specifies the process by which an agent may access the service. That is, it includes a communication protocol, message formats, communication port ID, etc.	Yes	Service-Grounding-Structure		
ProviderID	Specifies the provider entity identification	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-ImplementationID	Unique identifier for the new implementation of the service. It is generated by the SF	No	String		
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	Duplicate, Invalid-Struct, InvalidAccess, Invalid-ServiceID	

Precondition	
Pre1	$\exists S \in SF \mid S.ServiceID = ServiceID \wedge (\neg\exists I \in S.Providers \mid I.ServiceModel = ServiceModel \wedge I.ServiceGrounding = ServiceGrounding)$
Postcondition	
Post1	$\exists S \in SF \mid S.ServiceID = ServiceID \wedge (\exists I \in S.Providers \mid I.ServiceImpID = ServiceImpID \wedge ProviderID \in I.ProvidersIDList \wedge I.ServiceModel = ServiceModel \wedge I.ServiceGrouning = ServiceGrouning)$

3. **DeregisterProfile**: it is used to delete a service description. The following parameters have to be completed:

***DeregisterProfile*(ServiceID ?sID)**

The results of this service are:
- Service-status indicating success, if the service profile has been successfully removed.

Service Specification					
Name:	DeregisterProfile				
Description:	It is a meta-service and is used to delete from the SF a registered service				
Supplied by:	SF				
Required by:	any role.				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
ServiceID	Specifies the service that will be deleted	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	NotFound, InvalidAccess, InvalidServiceID	
Precondition					
Pre1	$\exists S \in SF \mid S.ServiceID = ServiceID$				
Postcondition					
Post1	$\nexists S \in SF \mid S.ServiceID = ServiceID$				

4. **SearchService**: it searchs a service whose description satisfies the client request. The search process can use matchmaking, composition and other techniques to solve complex queries. The required information for the request is:

SearchService(ServicePurpose ?sPurpose)

where ServicePurpose is a general structure in which the request is stored. It can be expressed as a SeviceGoal, a ServiceProfile description or a combination of both.
The output of this service is:
   - list of tuples <ServiceID, Ranking> and a Service-status indicating success éxito, an appropriate service has been found. Ranking models the matching between the service and the request.

Service Specification					
Name:	SearchService				
Description:	It is a meta-service and is used to search a service which satisfies the client requirements				
Supplied by:	SF				
Required by:	any role.				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
ServicePurpose	Specifies the client requirements. The requirements may be specified in terms of Service-Goal-Structure, an incomplete Service-Profile-Structure, or a combination of both	Yes	Service-Goal-Structure / Service-Profile-Structure		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
ServiceList	A list of <ServiceID, Ranking> tuplas	No	Service-RankList-Structure		
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	Not found, Invalid-Struct, InvaliAccess	
Precondition					
-	-				
Postcondition					
-	-				

5. **SearchProvider**: it is used to find a service provider for an specific service. The following information has to be included in the user request:

***SearchProvider*(ServiceID ?sID)**

The output of this service is:
- ProviderID list and Service-status indicating success if the provider has been found.

Service Specification					
Name:	SearchProvider				
Description:	It is a meta-service and is used to search for the provider of a given service				
Supplied by:	SF				
Required by:	any role.				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
ServiceID	Specifies the service ID	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-ProviderList	A list of ProviderID	No	Provider-IDList-Structure		
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	Not found, Invalid-Struct, Invali-dAccess	
Precondition					
Pre1	$\exists serv \in SF \mid serv.ServiceID = ServiceID$				
Postcondition					
-	-				

6. **ModifyProfile**: it is used to modify the description (profile) of a registered service. The client specifies the part of the service to be modified. The service Id will not change.

***ModifyProfile*(ServiceID ?sID, ServiceGoal ?Sgoal, ServiceProfile ?Sprofile)**

The output of this service is:
- Service-status indicating successful, if hte service has been changed.

Service Specification					
Name:	ModifyProfile				
Description:	It is a meta-service and is used to modify the description of an already registered service				
Supplied by:	SF				
Required by:	any role.				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
ServiceID	Specifies the service ID	Yes	String		
ServiceGoal	Specifies the new service Goal	No	String		
ServiceProfile	Specifies the new service profile	No	Service-Profile-Structure		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	Not found, Invalid-Struct, Invali-dAccess	
Precondition					
Pre1	$\exists serv \in SF \mid serv.ServiceID = ServiceID$				
Postcondition					
Post1	$\exists serv \in SF \mid serv.ServiceID = ServiceID \wedge serv.ServiceGoal = ServiceGoal \wedge serv.ServiceProfile = ServiceProfile$				

7. **ModifyProcess**: it is used to modify the implementation of a registered service. The client specifies the part of the service to be modified. The service Id will not change.

   ***ModifyProcess*(ServiceID ?sID, ServiceModel ?Smodel, ServiceGrounding ?Sgrounding, EntityID ?ProviderID)**

   If more than one provider implements the service, then the implementation will not be modified.
   The output of this service is:
   - Service-status indicating successful, if the service has been changed.
   - Service-status indicating error + Not-empty, there is more than one provider for the required implementation.

Service Specification					
Name:	ModifyProcess				
Description:	It is a meta-service and is used to modify a given implementation of an already registered service				
Supplied by:	SF				
Required by:	any role.				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Service-ImplementationID	Specifies the service implementation ID	Yes	String		
ServiceModel	Specifies the new service model	No	Service-Model-Structure		
ServiceGrounding	Specifies the new service grounding	No	Service-Grounding-Structure		
ProviderID	Specifies the provider entity ID	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	Not found, Invalid-Struct, Invali-dAccess, Not empty	
Precondition					
Pre1	$\exists serv \in SF \mid serv.ServiceID = ServiceID \wedge \exists! prov \in serv.Providers \wedge prov.ProviderID = ProviderID$				
Postcondition					
Post1	$\exists prov \in Providers \mid prov.ServiceImpID = ServiceImpID \wedge \exists! prov \in serv.Providers \wedge prov.ProviderID = ProviderID \wedge prov.ServiceModel = ServiceModel \wedge prov.ServiceGrouning = ServiceGrouning$				

8. **AddProvider**: adds a new provider to an existing service implementation.

***AddProvider*(ServiceID ?sID, EntityID ?ProviderID)**

The output of this service is:
- Service-status indicating successful, if the provider has been added.

Service Specification					
Name:	AddProvider				
Description:	It is a meta-service and is used to add a new provider to a given service implementation				
Supplied by:	SF				
Required by:	any role.				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Service-ImplementationID	Specifies the service implementation ID	Yes	String		
ProviderID	Specifies the new provider entity ID	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	Not found, Invalid-Struct, InvalidAccess	
Precondition					
Pre1	$\exists serv \in Providers \mid serv.ServiceImpID = ServiceImpID \wedge ProviderID \notin serv.ProviderIDList$				
Postcondition					
Post1	$\exists serv \in Providers \mid serv.ServiceImpID = ServiceImpID \wedge ProviderID \in serv.ProviderIDList$				

9. **RemoveProvider**: it deletes a provider from a service implementation.

***RemoveProvider*(ServiceID ?sID, EntityID ?ProviderID)**

If it is the last provider, the the implementation is auomatically erased. Furthermore, if this were the last implementation of the service, then the provider is alerted and it can deregister the service.
The output of this service is:

- Service-status indicating success, if the prodiver has been erased.

Service Specification					
Name:	RemoveProvider				
Description:	It is a meta-service and is used to remove a provider to a given service implementation				
Supplied by:	SF				
Required by:	any role.				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Service-ImplementationID	Specifies the service implementation ID	Yes	String		
ProviderID	Specifies the provider entity ID to be deleted	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Service Result	Yes	Enum	Ok, Error	
Error-Value	Error condition	No	Enum	Not found, Invalid-Struct, InvalidAccess	
Precondition					
Pre1	$\exists serv \in Providers \vert serv.ServiceImpID = ServiceImpID \wedge ProviderID \in serv.ProviderIDList$				
Postcondition					
Post1	$\exists serv \in Providers \vert serv.ServiceImpID = ServiceImpID \wedge ProviderID \in serv.ProviderIDList$				
Post2	$\exists serv \in Providers \vert serv.ProviderIDList = \emptyset \rightarrow [ModifyProcess(serv.ServiceID, \emptyset, \emptyset, \emptyset)]$				
Post3	$\exists serv \in SF \vert serv.Providers = \emptyset \rightarrow [Deregister(serv.ServiceID)]$				

2.2 Organization Manager Service

This component is in charge of organizations life cycle management, including specification and administration of both their structural components (roles, units and norms) and their execution components (participant agents and the roles they play; active units in each moment).

To carry out this management the OMS makes use of the following lists:

1. *UnitList*: it stores the list of existing units, together with their objectives, type and parent unit (*SuperUnit*).
2. *RoleList*: is stores the list of roles defined in each unit and their attributes (accessibility, visibility, position and inheritance).
3. *NormList*: it stores the list of norms defined in the system.
4. *EntityPlayList*: it stores the list of units in which each agent has been registered as a member, together with its adopted roles inside.

OMS offers all services needed for a suitable organization performance. These services are classified as: structural services, that modify the structural and normative organization specification; and dynamical services, that allow agents to entry or leave the organization dynamically, as well as role adoption.

By means of the publication of the *structural services*, OMS allows modifying, in execution time, some aspects related to the organization structure, functionality or normativity. For example, a specific agent of the organization could be allowed to add new norms, roles or units. This type of services should be restricted to internal roles of the system, which have enough permission for doing this kind of operations (i.e. supervisor role). Moreover, in some concrete applications those services might not be published in the SF, so then agents cannot dynamically modify the structural components.

Dynamical services manage creation of new agents in the organization, entry or exit of unit members and role adoption. These services are always published in the SF.

Structural services. The OMS provides a group of services for registering or deregistering structural components, specifically roles, norms and units. Also it offers services for informing about these components.

A *role* represents a position inside the unit in which it is defined. It is related to some interaction norms, imposed by the unit structure and its concrete position inside the unit; and some behaviour norms, that specify its functionality (services that needs and offers), restrict its actions (prohibition, obligations and permissions) and establish the consequences of these norms (sanctions and rewards).

Therefore, a *norm* indicates obligations, permissions and prohibitions of roles related to service registering, requesting and fulfilment; service composition, or quality of service results. Thus, a norm defines those restrictions that cannot be expressed by means of service preconditions or postconditions.

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 inside others units. It enables the representation of organizative structures like hierarchy, matrix, coalition, etc. Furthermore, it indicates which are the structural positions of the system (i.e. member, supervisor, subordinate), as well as the relationships among these positions imposed by the structure.

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. For example, an agent that plays "HotelCustomer" role can request directly services that are assigned to "Customer" role. But, on the contrary, it is necessary that a "Customer" agent requests to OMS to acquire "HotelCustomer" role before making use of the services related to this role.

Following, **register services** of structural components are described:

1. **RegisterRole:** service used for requesting the registration of a new role inside a unit. As input parameters, it requires the role identifier, the unit in which this role must be registered, its visibility (whether it is public or private), its accessibility (internal or external), its position (whether it inherits from "member", "supervisor", "subordinated"), as well as its parent role in the role hierarchy. Only role and unit identifiers are mandatory.

RegisterRole(RoleID ?Role, UnitID ?Unit, Visibility ?Visible, Accessibility ?Accessible, Position ?position, IsA ?SuperRole)

Service Specification	
Name:	RegisterRole
Description:	Request registration of a new role inside a specific unit
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
RoleID	Role identifier	Yes	String		
Accessibility	Role can be acquired	No	Enum.	External, Internal	External
Visibility	Provide information of this role	No	Enum.	Public, Private	Public
Position	Structural position	No	Enum.	Member, Supervisor, Subordinate	Member
isA	Inheritance of roles. Role identifier of its direct parent in role hierarchy	No	String		Member
UnitID	Unit Identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Value Range**
Service-Status	Service result	Yes	Enum.	Ok, Error
Error-Value	Error Condition	No	Enum.	Duplicate, Invalid

Precondition	
Pre1:	$\neg\exists R \in RoleList \mid R.RoleID = RoleID$
Pre2:	$\exists U \in UnitList \mid U.UnitID = UnitID$
Pre3:	$\exists PR \in RoleList \mid PR.RoleID = isA$
Postcondition	
Post:	$\exists R \in RoleList \mid R.RoleID = RoleID \wedge R.Accessibility = Accessibility \wedge R.Visibility = Visibility \wedge R.Position = Position \wedge R.isA = isA \wedge R.UnitID = UnitID$

2. **RegisterNorm:** used for requesting the registration of a new norm inside a unit. A norm definition includes which role it is addressed and which is its content (including deontic value, conditions, actions and associated sanctions or rewards). Optionally, it also indicates which role is in charge of the fulfilment of the norm (*issuer*), who will carry out the sanction (*defender*) and who is in charge of its reward (*promoter*).

RegisterNorm(NormID ?NID, AddressedRole ?Role, Content ?Cont, Issuer ?IssuerRole, [Defender ?DefenderRole], [Promoter ?PromoterRole])

Service Specification	
Name:	RegisterNorm
Description:	Include a new norm inside a unit
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
NormID	Norm identifier	Yes	String		
AddessedRole	Role identifier affected by the norm	Yes	String		
IssuerRole	Role identifier in charge of norm fulfilment	No	String		
Content	Deontic content of the norm	Yes	Deontic Content		
DefenderRole	Role identifier in charge of carrying out the sanction.	No	String		
PromoterRole	Role identifier in charge of carrying out the reward.	No	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Value Range**
Service-Status	Service result	Yes	Enum.	Ok, Error
Service-Value	Error condition	No	Enum.	Duplicate, Invalid ,Contradiction

Precondition	
Pre1:	$\neg\exists N \in NormList \mid N{=}NormID$
Pre2:	$\exists PR \in RoleList \mid PR.RoleID = AddessedRoleID$
Pre3:	$\exists PR \in RoleList \mid PR.RoleID = IssuerRoleID$
Pre4:	$\exists PR \in RoleList \mid PR.RoleID = DefenderRoleID$
Pre5:	$\exists PR \in RoleList \mid PR.RoleID = PromoterRoleID$
Postcondition	
Post:	$\exists N \in NormList \mid N.NormID = NormID \wedge N.AddressedRole = AddressedRole \wedge N.IssuerRole = IssuerRole \wedge N.DefenderRole = DefenderRole \wedge N.PromoterRole = PromoterRole \wedge N.Content = Content$

The *Content* of a norm is formed by:

<DeonticConcept, Entity, Action, ServiceName, TemporalCondition, StateCondition, Sanction, Reward>

Deontic Content				
Name	**Description**	**Mand.**	**Type**	**Range**
Deontic Concept	Deontic permission of the norm	Yes	Enum.	Obliged, Forbidden, Permitted
Action	Action related to registering, requesting or providing services	Yes	Enum.	Request, Serve, Register
Service	Identifier of affected service	Yes	String	
StateCondition	State condition for the norm activation	Yes	Normative Condition	
TemporalCondition	Temporal condition for finishing the norm. If satisfied and the norm has not been performed yet, then the sanction must be carried out. Otherwise, the reward is applied.	No	Integer	
SanctionNormID	Norm identifier. Addressed to the defender role	No	String	
RewardNormID	Norm identifier. Addressed to the promoter role, for performing the reward	No	String	

3. **RegisterUnit:** used for requesting the registration of a new empty unit in the organization, with a specific structure, goal and parent unit.

RegisterUnit(UnitID ?AID, UnitType ?Type, UnitGoal ?Goal, [UnitParent ?UnitParent])

Service Specification	
Name:	RegisterUnit
Description:	Request registering a new empty unit in OMS
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
UnitID	Unit identifier	Yes	String		
Type	Type of organizative structure	No	Enum.	Flat, Team, Hierarchy	Flat
Plays	Role played by the new unit inside the superior unit	No	String		Member
ParentUnitID	Superior unit identifier, to which the new one belongs	No	String		Virtual

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	Enum.	Ok, Error
Error-Value	Error condition	No	Enum.	Duplicate, Invalid

Precondition	
Pre1:	$\neg\exists U \in UnitList \vert U.UnitID = UnitID$
Pre2:	$\exists R \in RoleList \vert R.RoleID = Plays$
Pre3:	$\exists PU \in UnitList \vert PU.UnitID = ParentUnitID$
Postcondition	
Post:	$\exists U \in UnitList \vert U.UnitID = UnitID \wedge U.Type = Type \wedge U.Plays = Plays \wedge R.ParentUnitID = ParentUnitID$

All these structural services are implemented (*grounding*) by means of the FIPA-Request protocol. Thus, a client of the service sends a "Request" message, which contains all needed information for requesting the service. Then the server replies with an "Agree" message, if it agrees to provide the service, and later with an "Inform-done", with the corresponding value of *service-status*.

Optionally, more complex services for updating organization components can be offered by means of composition of the above services. For example, a complex service that offers the inclusion of a new role indicating its name, attributes and related norms. Or a complex service for unit creation that 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.

On the other hand, OMS offers services for deregistration of structural components. These **deregister services** are:

1. **DeregisterRole:** used for requesting the deregistration of a role. There must not be any agent playing this role nor any norm addressed to it.

DeregisterRole(RoleName ?Role, UnitID ?Unit)

Service Specification	
Name:	DeregisterRole
Description:	Delete a role from a specific unit.
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
RoleID	Role identifier	Yes	String		
UnitID	Unit identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	Enum.	Ok, Error
Error-Value	Error condition	No	Enum.	Not-found, Invalid

Precondition	
Pre1:	$\exists R \in RoleList \mid R.RoleID = RoleID \wedge R.UnitID = UnitID$
Pre2:	$\neg\exists N \in NormList \mid N.AddressedRoleID = RoleID$
Pre3:	$\neg\exists E \in EntityPlayList \mid E.RoleID = RoleID$
Postcondition	
Post:	$\neg\exists R \in RoleList \mid R.RoleID = RoleID$

2. **DeregisterNorm:** used for deleting a norm. The role that requests this service should be the issuer of the norm, that is, the controller of the norm.

DeregisterNorm(NormID ?NID)

Service Specification	
Name:	DeregisterNorm
Description:	Eliminate a norm
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
NormID	Norm identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	Enum.	Ok, Error
Error-Value	Error condition	No	Enum.	Not-found, Invalid

Precondition	
Pre1:	$\exists N \in NormList \wedge \exists E \in EntityPlayList \mid N.NormID = NormID \wedge E.AgentID = ClientID \wedge E.RoleID = N.issuerRole$
Postcondition	
Post:	$\neg\exists N \in NormList \mid R.NormID = NormID$

3. **DeregisterUnit:** service used for deleting a unit. This unit must be completely empty, without agents, nor roles or units inside. If the *UnitParent* input parameter is not given, it is assumed that the unit belongs to a "virtual" unit created by the agent platform.

DeregisterUnit(UnitID ?UID, [UnitParent ?UnitParent])

Service Specification					
Name:	DeregisterUnit				
Description:	Eliminate a unit				
Supplied by:	OMS				
Required by:	ClientRole				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
UnitID	Unit identifier	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	
Service-Status	Service result	Yes	Enum.	Ok, Error	
Error-Value	Error condition	No	Enum.	Not-found, Invalid	
Precondition					
Pre1:	$\exists U \in UnitList \vert U.UnitID = UnitID$				
Pre2:	$\neg\exists R \in RoleList \vert R.UnitID = UnitID$				
Postcondition					
Post:	$\neg\exists U \in UnitList \vert U.UnitID = UnitID$				

All these deregister structural services are also implemented (*grounding*(by means of the FIPA-REQUEST protocol.

Information services offered by OMS provide specific information of all components of the organization and they might be restricted to some internal roles of the system. Furthermore, if OMS is the only one which uses those services, then they are not directly published in the SF. Following, the set of informative services is detailed:

1. **InformAgentRole:** service used for requesting the list of roles and units in which an agent is in a specific moment. This service accesses to *EntityPlayList*.

InformAgentRole(AgentId ?AID)

Service Specification					
Name:	InformAgentRole				
Description:	Request the list of roles and units in which an agent participates in a specific moment				
Supplied by:	OMS				
Required by:	ClientRole				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
AgentID	Agent identifier	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	
Service-Status	Service result	Yes	Enum.	Ok, Error	
Service-Value	Error condition	No	Enum.	Not-Found, Invalid	
RoleUnitList	List of units and roles played by the agent	No	List(RoleID, UnitID)		
Precondition					
Pre1:	$\exists E \in EntityPlayList \vert E.AgentID = AgentID$				
Postcondition					
–	—				

2. **InformMembers:** used for requesting the list of entities that are members of a specific unit. Optionally, it is possible to specify a role of this unit, so then only members playing this role are detailed. This service accesses to *EntityPlayList*.

InformMembers(UnitID ?Unit [,RoleID ?Role])

Service Specification					
Name:	InformMembers				
Description:	Request the list of entities that are members of a specific unit				
Supplied by:	OMS				
Required by:	ClientRole				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
UnitID	Unit identifier	Yes	String		
RoleID	Role identifier	No	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	
Service-Status	Service result	Yes	Enum.	Ok, Error	
Service-Value	Error condition	No	Enum.	Not-Found, Invalid	
AgentRoleList	List that contains for each item agent and role identifiers	No	List(AgentID, RoleID)		
Precondition					
Pre1:	$\exists U \in UnitList \vert U.UnitID = UnitID$				
Pre2:	$\exists R \in RoleList \vert R.RoleID = RoleID$				
Postcondition					
–	—				

3. **QuantityMembers:** used for requesting the number of current members of a specific unit. Optionally, if a role is indicated then only the quantity of members playing this role is detailed. This service accesses to *EntityPlayList*.

QuantityMembers(UnitID ?Unit [,RoleID ?Role])

Service Specification					
Name:	QuantityMembers				
Description:	Request the number of current members of a specific unit				
Supplied by:	OMS				
Required by:	ClientRole				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
UnitID	Unit identifier	Yes	String		
RoleID	Role identifier	No	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	
Service-Status	Service result	Yes	Enum.	Ok, Error	
Service-Value	Error condition	No	Enum.	Not-Found, Invalid	
Quantity	Number of agents that are playing the specified role	No	Integer		
Precondition					
Pre1:	$\exists U \in UnitList \vert U.UnitID = UnitID$				
Pre2:	$\exists R \in RoleList \vert R.RoleID = RoleID$				
Postcondition					
–	—				

4. **InformUnit:** used for requesting information about a specific unit that has been registered in *UnitList*.

InformUnit(UnitID ?Unit)

Service Specification					
Name:	InformUnit				
Description:	Requests the information about a specific unit				
Supplied by:	OMS				
Required by:	ClientRole				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
UnitID	Unit identifier	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	
Service-Status	Service result	Yes	Enum.	Ok, Error	
Service-Value	Error condition	No	Enum.	Not-Found, Invalid	
UnitType	Unit type	No	Enum.	Flat, Team, Hierarchy	
ParentID	Identifier of the parent unit	No	String		
UnitGoal	Unit goals	No	List(ServiceID)		

Precondition	
Pre1:	$\exists U \in UnitList \vert U.UnitID = UnitID$
Postcondition	
–	—

5. **InformRole:** used for requesting the list of roles that have been registered inside a unit. This service accesses to *RoleList*.

InformRole(UnitID ?Unit)

Service Specification	
Name:	InformRole
Description:	Request the list of roles that have been registered inside a unit
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
UnitID	Unit identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	Enum.	Ok, Error
Service-Value	Error condition	No	Enum.	Not-Found, Invalid
RoleList	Role list	No	List(RoleID)	

Precondition	
Pre1:	$\exists U \in UnitList \vert U.UnitID = UnitID$
Postcondition	
–	—

6. **InformProfile:** used for requesting the list of profiles associated to a specific role, according to the norms assigned to this role. Those norms specify its functionality.

InformProfile(RoleID ?Role)

Service Specification	
Name:	InformProfile
Description:	Requests the list of profiles associated to a specific role
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
RoleID	Role identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	Enum.	Ok, Error
Service-Value	Error condition	No	Enum.	Not-Found, Invalid
ProfileList	List of profiles assigned to the role	No	List(ProfileID)	

Precondition	
Pre1:	$\exists R \in RoleList \vert R.RoleID = RoleID$
Postcondition	
–	—

7. **InformNorm:** used for requesting the list of norms addressed to a specific role. This service accesses to the *NormList*.

InformNorm(RoleID ?Role)

Service Specification	
Name:	InformNorm
Description:	Request the list of norms addressed to a specific role
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
RoleID	Role identifier	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	
Service-Status	Service result	Yes	Enum.	Ok, Error	
Service-Value	Error condition	No	Enum.	Not-Found, Invalid	
NormList	Norm list	No	List(NormID)		
Precondition					
Pre1:	$\exists R \in RoleList \mid R.RoleID = RoleID$				
Postcondition					
–	—				

All information services are implemented by means of the FIPA-Query protocol, so a client sends a "Query-ref" message requesting information about a specific concept; and the server answers with an "Inform" message containing the corresponding data.

Dynamic Services. OMS offers a set of basic composed services for dynamical role adoption and entry/exit of unit members. Most of these basic services are not directly accessible for agents, but are combined through compound services.

Basic services for role adoption are:

1. **RegisterAgentRole:** used for registering a new item in *EntityPlayList*, indicating that an agent plays a specific role inside a unit. This service is not directly published in the SF.

RegisterAgentRole(AgentID ?AID, RoleId ?Role, UnitID ?UID)

Service Specification					
Name:	RegisterAgentRole				
Description:	Register that an agent plays a role inside a unit.				
Supplied by:	OMS				
Required by:	—				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
RoleID	Role identifier	Yes	String		
UnitID	Unit identifier	Yes	String		
AgentID	Agent identifier	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	
Service-Status	Service result	Yes	String	Ok, Error	
Service-Value	Error condition	No	Enum.	Duplicate, Invalid	
Precondition					
Pre1:	$\exists R \in RoleList \mid R.RoleID = RoleID$				
Pre2:	$\exists U \in UnitList \mid U.UnitID = UnitID$				
Pre3:	$\neg\exists E \in EntityPlayList \mid E.AgentID = AgentID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$				
Postcondition					
Post1:	$\exists E \in EntityPlayList \mid E.AgentID = AgentID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$				

2. **DesregisterAgentRole:** used for deleting an item in *EntityPlayList*, so then a specific agent does not play the role in the unit anymore. This service is not directly published in the SF.

DesregisterAgentRole(AgentID ?AID, RoleId ?Role, UnitID ?UID)

Service Specification	
Name:	DesregisterAgentRole
Description:	Deregister a *Agent-Role-Unit* entry, so an agent does not play a specific role inside a unit
Supplied by:	OMS
Required by:	—

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
RoleID	Role identifier	Yes	String		
UnitID	Unit identifier	Yes	String		
AgentID	Agent identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	String	Ok, Error
Service-Value	Error condition	No	Enum.	Not-Found, Invalid

Precondition	
Pre1:	$\exists R \in RoleList \vert R.RoleID = RoleID$
Pre2:	$\exists U \in UnitList \vert U.UnitID = UnitID$
Pre3:	$\exists E \in EntityPlayList \vert E.AgentID = AgentID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$
Postcondition	
Post1:	$\neg\exists E \in EntityPlayList \vert E.AgentID = AgentID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$

OMS also offers a set of compound services that can be used by agents for adopting roles, leaving them and apply sanctions. Following, these compound services are related:

1. **AcquireRole:** serviced used for acquiring a role in a specific unit.

AcquireRole(UnitID ?Unit, RoleID ?Role)

The execution of this service implies:
- Check that there is not any active norm of the client agent that forbids the execution of this *AcquireRole* service.
- Check that the requested role exits inside the unit and it is accessible.
- Check that the agent is already inside the unit (plays another role there) or it is inside its parent unit.
- Check compatibility restrictions, i.e. the requested role is not incompatible with the other roles played by the agent.
- Agent is informed of the functionality restrictions of the requested role (norms and profiles). Possible options:
  - a) Inform of norms that the agent must follow and protocols attached to its service profiles. The agent is in charge of managing this information and act according to it.
  - b) Establish a contract with the agent regarding its future behavior. In this contract the agent might commit to more restrictive actions that those indicated in the requested role.
- Register *Agent - Role - Unit* entry in *EntityPlayList* (using *RegisterAgentRole* service)
- Activate agent norms related with this requested role.

Service Specification	
Name:	AcquireRole
Description:	Request the role acquisition inside of a specific unit.
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
RoleID	Role identifier	Yes	String		
UnitID	Unit identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	String	Ok, Error
Service-Value	Error condition	No	Enum.	Duplicate, Invalid ,Incompatible, Not-Available

Precondition	
Pre1:	$\exists R \in RoleList \mid R.RoleID = RoleID$
Pre2:	$\exists U \in UnitList \mid U.UnitID = UnitID$
Pre3:	$\neg\exists E \in EntityPlayList \mid E.AgentID = ClientID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$
Postcondition	
Post1:	$\exists E \in EntityPlayList \mid E.AgentID = ClientID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$

2. **LeaveRole:** service used for leaving a role inside a specific unit.

LeaveRole((UnitID ?Unit, RoleID ?Role)

The execution of this service implies:

- Check that there is not any active norm of the client agent that forbids the execution of this *LeaveRole* service.
- Check that the agent plays this role inside the unit.
- Check that the agent has not active norms due to this role.
- Deregister *Agent - Role - Unit* entry in *EntityPlayList* (using *DeregisterAgentRole* service)

Service Specification	
Name:	LeaveRole
Description:	Request leaving a role in a specific unit
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
RoleID	Role identifier	Yes	String		
UnitID	Unit identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	String	Ok, Error
Service-Value	Error condition	No	Enum.	Not-Permitted, Invalid, Not-Available

Precondition	
Pre1:	$\exists R \in RoleList \mid R.RoleID = RoleID$
Pre2:	$\exists U \in UnitList \mid U.UnitID = UnitID$
Pre3:	$\exists E \in EntityPlayList \mid E.AgentID = ClientID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$
Postcondition	
Post1:	$\neg\exists E \in EntityPlayList \mid E.AgentID = ClientID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$

3. **Expulse:** service for forcing an agent to leave a specific role.

Expulse(Agent AID, UnitU, Role R)

The execution of this service implies:

- Check that there is not any active norm of the client agent that forbids the execution of this Expulse service. The client agent must be explicitly enabled for using this service. By default, agents are not allowed to expulse other agents.

- Check that the specified agent plays the indicated role inside the unit.
- Deregister *Agent - Role - Unit* entry in *EntityPlayList* (using *DeregisterAgentRole* service)
- Inform agent that it has been forced to leave this role.

Service Specification	
Name:	Expulse
Description:	Request the expulsion of an agent. This agent is obliged to leave the specified position
Supplied by:	OMS
Required by:	ClientRole

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Range**	**Default**
AgentID	Agent identifier	Yes	String		
RoleID	Role identifier	Yes	String		
UnitID	Unit identifier	Yes	String		

Output Parameters				
Name	**Description**	**Mand.**	**Type**	**Range**
Service-Status	Service result	Yes	String	Ok, Error
Service-Value	Error condition	No	Enum.	Not-Permitted, Invalid, Not-Available

Precondition	
Pre1:	$\exists R \in RoleList \mid R.RoleID = RoleID$
Pre2:	$\exists U \in UnitList \mid U.UnitID = UnitID$
Pre3:	$\exists E \in EntityPlayList \mid E.AgentID = AgentID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$
Postcondition	
Post1:	$\neg\exists E \in EntityPlayList \mid E.AgentID = AgentID \wedge E.UnitID = UnitID \wedge E.RoleID = RoleID$

2.3 Platform Kernel

Component in charge of providing the usual services required in a multi-agent system. Therefore, it is responsible for managing The life cycle of the agents included in the different organizations, and also allows to have a communication channel (incorporating different message transport mechanisms) to facilitate the interaction among different entities. On the other hand, the PK offers a safe connectivity and the necessary mechanisms that allow multi-device interconnectivity.

A previous security mechanism is supposed for some of the services below describe, which permits to manage who and over who can invoke each service. For example, the responsible for an organization may have the option of creating new agents inside its organization. For this, at kernel level of the platform at some point it should be invoked the agent register Service.

The services offered are in most cases FIPA legacy with some modifications. In the case of services directly related with the agent management we can find the following:

– **Register**: Service invoked by an entity of the platform in order to request an agent registration in the platform (which is equivalent to the creation of the agent). This implies that the life-cycle management of the agent will be managed in this platform.

register(Name ?n Address ?ad State ?s Attributes $a)

This service is invoked by the OMS because the registration of an agent is the result of the creation of an agent in a specific organization. This creation will be managed by the OMS, who will be responsible to inform the PK through the invocation of the agent registration service.

Service Specification					
Name:	Register				
Description:	To invoke the register of a new agent				
Supplied by:	PK				
Required by:	OMS				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Name	Name of the agent	Yes	String		
Address	Physical address of the agent	Yes	URL		
State		Yes	String	A, S W [1]	
Attributes		No	Set of String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Result of the service	Yes	String	Ok, Error	
Service-Value	Error condition	No	String	Duplicate, Invalid, Access	
Precondition					
Pre1:	$\neg\exists Ag \in AMS.AgentList \mid Ag.Name = Name \wedge Ag.Address = Address$				
Pre2:	$\exists Ad \in IP\_Address \mid Ad = Address$				
Pre3:	$State \in A, S, W$				
Postcondition					
Post:	$\exists Ag \in AMS.AgentList \mid Ag.Name = Name \wedge Ag.Address = Address \wedge Ag.State = State$				

– **Deregister**: an entity of the platform, for whatever reason, request to the platform for the elimination of an agent registration. The life cycle ceases to be controlled in this platform, which means that the agent is dead.

deregister(Name ?n)

This service is invoked by the OMS, as in the previous, case the removal of an agent on the platform is the responsibility of the OMS which probably transmits possible orders from the managers of an organization.

Service Specification	
Name:	Deregister
Description:	To invoke the deregister of an agent
Supplied by:	PK
Required by:	OMS

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Name	Name of the agent	Yes	String		

Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Result of the service	Yes	String	Ok, Error	
Service-Value	Error condition	No	String	Invalid, Not-found, Access	

Precondition	
Pre:	$\exists Ag \in AMS.AgentList \vert Ag.Name = Name$
Postcondition	
Post:	$\neg\exists Ag \in AMS.AgentList \vert Ag.Name = Name$

– **Update register**: Service that enables the modification of the information which appears in an agent register with the exception of the agent name.

modify(Name ?n Address ?ad State ?s Attributes $a)

This service is invoked by the same agent or the OMS.

Service Specification	
Name:	Modify
Description:	To modify the register of an specific agent previously registered
Supplied by:	PK
Required by:	OMS

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Name	Name of the agent	Yes	String		
Address	Physical address of the agent	No	URL		
State		No	String	A, S W	
Attributes		No	Set of String		

Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Result of the service	Yes	String	Ok, Error	
Service-Value	Error condition	No	String	Invalid, Not-found, Access	

Precondition	
Pre1:	$\exists Ag \in AMS.AgentList \mid Ag.Name = Name$
Pre2:	$\exists Ad \in IP\_Address \mid Ad = Address$
Pre3:	$State \in A, S, W$
Postcondition	
Post:	$\exists Ag \in AMS.AgentList \mid Ag.Name = Name \wedge Ag.Address = Address \wedge Ag.State = State \wedge Ag.State = Attributes$

– **Agent search**: Service that can be invoked by an entity to request information from a registered agent on the platform.

search(Name ?n Address ?ad State ?s Attributes $a)

This service is public, the search in the white pages are public unless the parameters of the registration indicate that this registration is private. In this case the search van be only invoked by the OMS.

Service Specification					
Name:	Search				
Description:	To search an agent in the platform				
Supplied by:	PK				
Required by:	agents				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Name	Name of the agent	No	String		
Address	Physical address of the agent	No	URL		
State		No	String	A, S W	
Attributes		No	Set of String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Result of the service	Si	String	Ok, Error	
Service-Value	Error condition	No	String	Invalid, Not-found, Access	
Precondition					
–	–				
Postcondition					
–	–				

– **Suspend an agent**: This service is invoked by an entity of the platform in order to suspend the execution of an specific agent.

suspend(Name ?n)

This service can be invoked by the same agent or the OMS.

Service Specification					
Name:	Suspend				
Description:	To suspend the execution of an agent				
Supplied by:	PK				
Required by:	the own agent and OMS				
Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Name	Name of the agent	Yes	String		
Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Result of the service	Yes	String	Ok, Error	
Service-Value	Error condition	No	String	Invalid, Not-found, Access	

Precondition	
Pre:	$\exists Ag \in AMS.AgentList \mid Ag.Name = Name$
Postcondition	
Post:	$\exists Ag \in AMS.AgentList \mid Ag.Name = Name \wedge Ag.State = S$

– **Agent activation**: This service is invoked by an entity of the platform to activate the execution of an agent who currently is suspended.

resume(Name ?n)

This service can be invoked by the OMS.

Service Specification	
Name:	Resume
Description:	To activate the execution of a suspended agent
Supplied by:	PK
Required by:	OMS

Input Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	**Default**
Name	Name of the agent	Yes	String		

Output Parameters					
Name	**Description**	**Mand.**	**Type**	**Value Range**	
Service-Status	Result of the service	Yes	String	Ok, Error	
Service-Value	Error condition	No	String	Invalid, Not-found, Access	

Precondition	
Pre:	$\exists Ag \in AMS.AgentList \mid Ag.Name = Name \wedge Ag.State = S$
Postcondition	
Post:	$\exists Ag \in AMS.AgentList \mid Ag.Name = Name \wedge Ag.State = A$

There exists a service in FIPA, which allows to obtain the platform description. This service has been retained in THOMAS for reasons of compatibility but it is not employed.

get-description(APName ?n)

With respect to services for the management of messages, the only service visible by the platform entities is the *send message*, which obviously allows sending a message through the communication layer. As concerning with message reception, the platform distributes messages that are coming to the relevant entity, which has a module for managing mailbox in a individualized form. The remaining actions offered by FIPA at message management level, such as asking for the type of codification are hidden in THOMAS for the entities at highest level.

The high-level description of the service for sending messages only involves an indication of who sends and who receives the message and the own message. The message will be encrypted according to the followed standard (it will include the communication act and its contents). This service can be invoked by any agent in the platform.

send(Sender ?s Receiver ?r Message ?m)

3 Conclusions

An important aspect for the development of true open multi-agent systems is to provide developers with methods, tools and appropiated architectures which support all the requirements for this kind of systems. This document has deepened into this problem trying to propose an abstract architecture for the development of virtual organizations. Moreover, the proposal tries to raise a total integration of two promising technologies, that is, multi-agent systems and service-oriented computing. In THOMAS architecture agents can offer and invoke services in a transparent way to other agents or entities, as well as external entities can interact with agents through the use of the offered services.

This architecture is the first step in order to obtain true deployed virtual organizations. Currently, a software platform based on this proposal has being developed and it is being applied in the development of different scenarios as tourism, leisure activity management on a mall and health emergencies.