



## **Magentix 2 Agents Development Guide for using Thomas organizations**

# Index

1	Thomas Architecture.....	3
2	API for Thomas.....	5
2.1	OMS.....	5
2.2	SF.....	5
2.3	API to acces OMS and SF agents.....	6
2.3.1	OMSProxy.....	6
2.3.2	SFProxy.....	7
2.4	Service Description.....	8
2.4.1	ProcessDescription.....	8
2.4.2	ProfileDescription.....	9
2.5	Utils.....	11
2.5.1	Oracle.....	11
2.5.2	CleanBD.....	11
3	How to program an agent that offer services.....	12
3.1	Register and Acquire rol.....	13
3.2	Service registration.....	14
3.3	Execution of a service (mindswap).....	15
4	How to program an agent client.....	16
4.1	Register and Acquire rol.....	16
4.2	Search Service.....	17
4.3	Request of services.....	18
5	Other examples: Special Agents and Agents initialization.....	19
5.1	Agent Payee.....	19
5.2	Agent Announcement.....	19
5.3	Agents initialization .....	20
6	Initialization tasks.....	23

# 1 Thomas Architecture

THOMAS is an open architecture that employs a service-based approach as the basic building blocks for creating a suitable platform for intelligent agents grouped in VOs.

Both information and mechanisms provided by the traditional FIPA Directory Facilitator (DF) are not good enough for dealing with open systems and system dynamics. In this way, it is necessary to develop intelligent methods to deal with the service management in open, decentralized MAS, i.e. intelligent service location taking into account semantic information or generation and adaptation of composed services. In a typical FIPA architecture, the traditional DF has several limitations: (i) its service descriptions are very basic (name, type, protocol, ontology, language, ownership, properties); (ii) it does not consider organizations; (iii) it is only oriented to the agent paradigm; (iv) it only allows a basic functionality (register, deregister, modify and search), (v) its service discovery algorithm is very simple, since its default search mechanism is assumed to be a depthfirst search across DFs and semantic information is not considered; and (vi) the service discovery is limited to discovering single services, as the default algorithm does not address the issue of considering service compositions.

Therefore, the THOMAS architecture feeds on the FIPA1 architecture, but extends its main components —the Agent Management System (AMS) and the Directory Facilitator (DF) — into an Organization Management System and a Service Facilitator, respectively.

The main components of THOMAS architecture are:

- Platform Kernel (PK), that deals with basic agent management services and it can be provided by any FIPA- compliant platform. Its functionality is related with the agent life-cycle and the network communication layer.
- Service Facilitator (SF), which is a service manager that registers services provided by external entities and facilitates service discovering for potential clients. The SF can be considered as a yellow pages server.
- Organization Management System (OMS), which is responsible of the management of virtual organizations, taking control of their underlying structure, the roles played by the agents inside the organization and the norms that rule the system behaviour.

The SF component is a redefinition of the traditional FIPA DF, being now capable of dealing with services in a more elaborated way, following Service Oriented Architectures guidelines. Thus, the SF copes with the limitations of the traditional DF, considering semantic information, service composition as well as service roles, goals and durations.

The SF supplies a set of standard services for managing the functionality of organizations and individual agents. SF services are classified in three types:

- **Registration:** they allow adding, modifying and removing services from the SF directory (*RegisterProfile, RegisterProcess, ModifyProfile, ModifyProcess*).
- **Affordability:** for managing the association between providers and their services (*AddProvider, RemoveProvider*).
- **Discovery:** for searching and composing services as an answer to user requirements (*SearchService, GetProfile, GetProcess*).

The OMS component is in charge of controlling how the agent groups, named Organizational Units, are created; which are the entities participating inside; how these entities are related to each other and which roles they play through time. Thus, the OMS provides agents with a set of services for organization life-cycle management, classified in:

- **Structural services**, which modify the structural and normative organization specification, i.e. roles, norms and organizational units (*RegisterRole, DeregisterRole, RegisterNorm, De-registerNorm, RegisterUnit, DeregisterUnit*).
- **Informative services**, that give information of the current state of the organization (*InformUnitRoles, InformAgentRoles, InformMembers, InformQuantity, InformUnit, InformRoleProfiles, InformRoleNorms*).
- **Dynamic services**, that allow managing role enactment and dynamic entry/exit of agents (*AcquireRole, LeaveRole, Expulse*).[1]

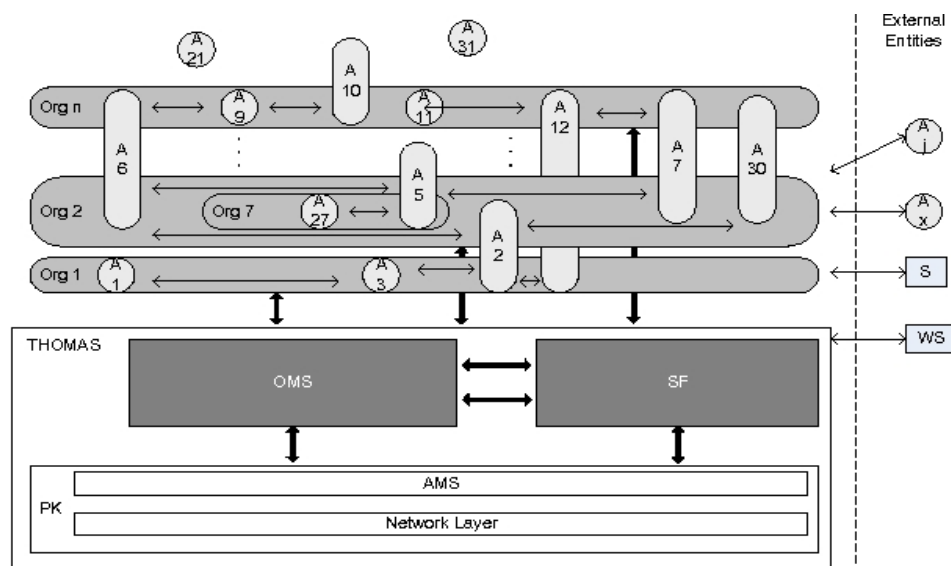


Fig.1. Thomas Architecture

## 2 API for Thomas

There exist two types of intermediary agents in the platform Magentix2 (SF, OMS) that operate of translators between the agents FIPA of the platform and the Services Web. This type of agents receive requests for services through the FIPA-request protocol and are responsible to access the corresponding Web Services.

### 2.1 OMS

The OMS we can find inside the package

**es.upv.dsic.gti\_ia.organization.OMS**

To launch the OMS agent in our platform we have designed and implemented a class with all the logic of OMS agent, therefore only have to create an instance of OMS agent and launch it into our main method.

```
import es.upv.dsic.gti_ia.organization.OMS;  
  
OMS agenteOMS = OMS.getOMS();  
agenteOMS.start();
```

### 2.2 SF

The SF we can find inside the package

**es.upv.dsic.gti\_ia.organization.SF**

As the class of the OMS, we have implemented one for the agent SF, for instantiate and launch it we will do the same thing that in the class OMS.

```
import es.upv.dsic.gti_ia.organization.SF;  
  
SF agenteSF = SF.getSF();  
agenteSF.start();
```

## 2.3 API to acces OMS and SF agents.

To facilitate the interaction of our agents with the agents OMS and SF we have developed two classes (SFProxy and OMSPProxy) that will do us of proxy with the agents, encapsulating and hiding all process of interaction (protocol Request). But also we can create and complete the message ACLMessage, for it we will have to use the protocol fipa request.

### 2.3.1 OMSPProxy

The OMSPProxy we can find inside the package:

```
import es.upv.dsic.gti_ia.organization.OMSPProxy;
```

To use this functionality must create a new instance of the class OMSPProxy, then they can access the methods contained in the OMS.

In the constructor we enter the url where services are deployed OMS (.war)

```
private OMSPProxy omsProxy = new OMSPProxy("url");
```

or empty where the route will take us from the settings.xml file.[6]

```
private OMSPProxy omsProxy = new OMSPProxy();
```

The methods will return a status indicating whether the operation was successful or not, we must also handle exceptions that can give us.

```
try {  
    omsProxy.acquireRole(this, "member", "virtual");  
} catch (Exception e) {  
    logger.error(e.getMessage());  
}
```

To consult all methods that we can offer OMSPProxy consult the Magentix2 API or the accompanying documents on the web (clasificación servicios thomas.pdf).

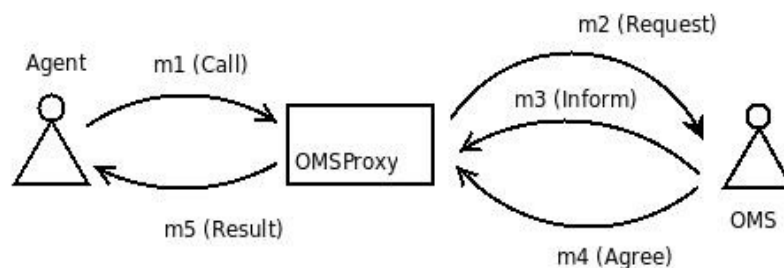


Fig. 2. Interaction with OMSPProxy and OMS agent

### 2.3.2 SFProxy

The SFProxy we can find inside the package

```
import es.upv.dsic.gti_ia.organization.SFProxy;
```

To use this functionality must create a new instance of the class SFProxy, then they can access the methods contained in the SF.

In the constructor we enter the url where services are deployed SF (.war)

```
private SFProxy sfProxy = new SFProxy("url");
```

or empty where the route will take us from the settings.xml file.[6]

```
private SFProxy sfProxy = new SFProxy();
```

The methods will return a status indicating whether the operation was successful or not, we must also handle exceptions that can give us..

```
try {  
    results = sfProxy.searchService(this,  
    "SearchCheapHotel");  
}  
} catch (Exception e) {  
    logger.error(e.getMessage());  
}
```

To consult all methods that we can offer SFProxy consult the Magentix2 API or the accompanying documents on the web (clasificación servicios thomas.pdf).

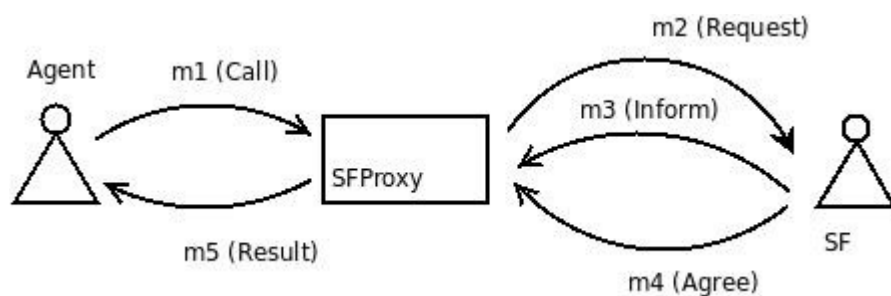


Fig.3. Interaction with SFProxy and SF agent

To learn more about creating services and owl<sup>1</sup> files consult thomas documentation<sup>2</sup>.

<sup>1</sup> <http://www.w3.org/TR/owl-features/>

<sup>2</sup> <http://users.dsic.upv.es/grupos/ia/sma/tools/Thomas/downloads.htm>

## 2.4 Service Description

To facilitate the maintenance of the process and the profiles that we will register in the organization we have created some templates which can adequately fill in all required fields in a profile (ProfileDescription) and process (ProcessDescription).

### 2.4.1 ProcessDescription

The ProcessDescription we can find inside the package **es.upv.dsic.gti\_ia.organization.ProcessDescription**

These templates can find all methods and variables needed for proper configuration of a process. The most important are:

- **ServiceID**

The service profile id is a number generated automatically by the sf database.

- **Implementation ID**

Is the id of the service implementation. It is composed of: `serviceid@serviceprocessid-agentid`

- **URLProcess**

The URL where the process owl document is located

- **Servicemodel**

It is a url which makes reference to the service process description document.

Also exist two important methods:

- **getProcess(ACLMessage inmsg)**

Returns a **org.mindswap.owls.process.Process**<sup>3</sup> to use him to the execution of a service.

- **getServiceRequestValues(ACLMessage inmsg)**

Returns a ValueMap with the name of the field and the value that there gives us the client who calls to the service.

This class is useful when the agent acts as an agent provider and want to register a new service we want to offer to other agents, for example a travelagency .

---

<sup>3</sup> <http://www.mindswap.org/2004/owl-s/api/>



The constructor parameters are :

- **URLprocess**  
The URL where the process owl document is located
- **Process name**  
Name to the service process description document.

In order to explain further the previous concepts, the following code illustrates this approach:

```
import es.upv.dsic.gti_ia.organization.SFProcessDescription;

ProcessDescription process = new ProcessDescription(
"http://localhost:8080/sfservices/THservices/owl/owls/SearchCheapH  
otelProcess.owl", "SearchCheapHotel");

try {
    results = sfProxy.searchService(this, "SearchCheapHotel");
    process.setProfileID(results.get(0));
    sfProxy.registerProcess(this, process);
} catch (Exception e) {
    logger.error(e.getMessage());
}
```

#### **2.4.2 ProfileDescription**

The ProfileDescription we can find inside the package  
**es.upv.dsic.gti\_ia.organization.ProfileDescription**

These templates can find all methods and variables needed for proper configuration of a profile. The most important are::

- **ServiceID**  
Is generated automatically by the database. It's a numeric value.
- **URLProfile**  
The URL where the profile owl document is located
- **Serviceprofile**  
It is a url which makes reference to the service profile description document

This class is useful when an agent wants to register a profile in the organization and there is none that fits your needs, for example if we do a search for cheap hotels.

The constructor parameters are:

- **URLprofile**  
The URL where the profile owl document is located
- **Profile name**  
Name to the service profile description document.

In order to explain further the previous concepts, the following code illustrates this approach:

```
import es.upv.dsic.gti_ia.organization.SFProfileDescription;

ProfileDescription profile = new ProfileDescription(
"http://localhost:8080/sfservices/THservices/owl/owls/SearchCheapH
otelProfile.owl", "SearchCheapHotel");

try {

SFservices.registerProfile(this, profile);

} catch (Exception e) {
    logger.error(e.getMessage());
}
```

## 2.5 Utils

### 2.5.1 Oracle

This class allows us to parse a profile in order to extract relevant information, such as service inputs, outputs, list of roles for both providers, such as customers.

Once we have the profile and call the oracle for example we can return the roles they have to acquire in order to register a service or to apply for the service as a customer, you will see that you will accept service inputs and what values we will return.

The constructor parameters are:

- URL del profile

This URL can be obtained with the method `getProfile ()` of `SFPROxy` class and then convert it into a URL

```
import java.net.URL;
import es.upv.dsic.gti_ia.organization.Oracle;
private Oracle oracle;
String URLProfile = sfProxy.getProfile(this, results.get(0));
URL profile;
try {
    profile = new URL(URLProfile);
    oracle = new Oracle(profile);
} catch (MalformedURLException e) {
    logger.error("ERROR: Profile URL Malformed!");
    e.printStackTrace();
}
oracle.getProviderList();
```

### 2.5.2 CleanBD

This class will use the principle of running to clean the database, both of Thomas as the Jena<sup>4</sup>

We must be careful to have the `settings.xml` configuration file properly configured (default is already set) and this located in the folder `configuration`[6].

```
import es.upv.dsic.gti_ia.organization.CleanBD;

CleanBD clean = new CleanBD();
clean.clean_database();
```

---

<sup>4</sup><http://jena.sourceforge.net/>

### 3 How to program an agent that offer services.

These examples are extracted from a simulation of a unit (travelagency) that appears in Thomas Demo.<sup>5</sup>

#### Abstract

Travel Agency application is an application that facilitates the interconnection between customers, interested in information services and tourist reservations, and providers of these services, as for example hotel companies, airlines, etc. This case study is modeled as a unit (travelagency) within which are provided search services tourist information and reservation of hotels and flights.

**Considerations:** the agents that we create extend of the template QueueAgent and we will have to put the whole logic of the agent overloading the method execute (explained more up of the document).

```
import es.upv.dsic.gti_ia.architecture.QueueAgent;
import es.upv.dsic.gti_ia.core.AgentID;

public class AgentClient extends QueueAgent {

    public AgentClient(AgentID aid) throws Exception {
        super(aid);
    }

    public void execute() {

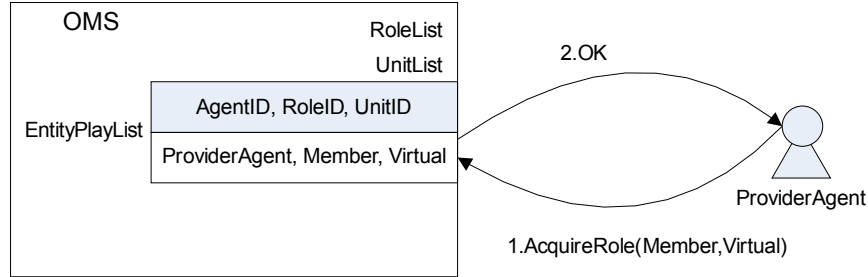
        logger.info("Executing, I'm " + this.getName());
        .
        .
        .
    }
}
```

---

<sup>5</sup> <http://users.dsic.upv.es/grupos/ia/sma/tools/Thomas/downloads.htm>

### 3.1 Register and Acquire rol.

- Registration of an agent on the platform Thomas. The agents request to register as members of the platform THOMAS across the service AcquireRole:

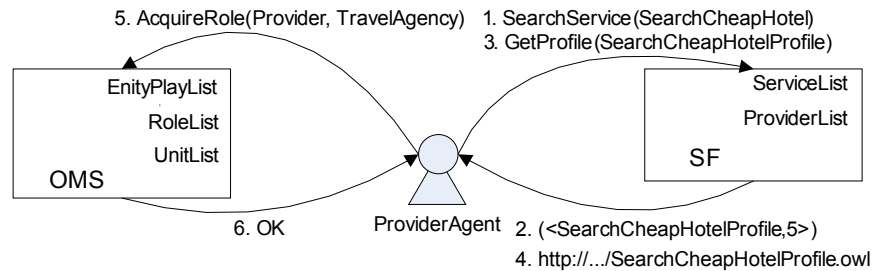


```

try {
    omsProxy.acquireRole(this, "member", "virtual");
} catch (Exception e) {
    logger.error(e.getMessage());
}

```

- Registering a new provider. After registering as a member of Thomas another agent platform can make use of the services offered by the OMS and SF. So that the provider agent requests to the SF the list of services that have been registered in the platform by a profile similar to the service of search of cheap hotels (SearchCheapHotel) that the same one offers:



```

results = sfProxy.searchService(this, "SearchCheapHotel");

```

Then the agent ProviderAgent requests the service GetProfile to the SF to obtain an information more detailed to near the service SearchCheapHotel:

```

String URLProfile = sfProxy.getProfile(this, results.get(0));

```

The SF responds with the URI where the service profile description are located. The ProviderAgent analyzes the profile of service , it indicates that the providers of the service must adopt the Provider role inside the unit TravelAgency.:

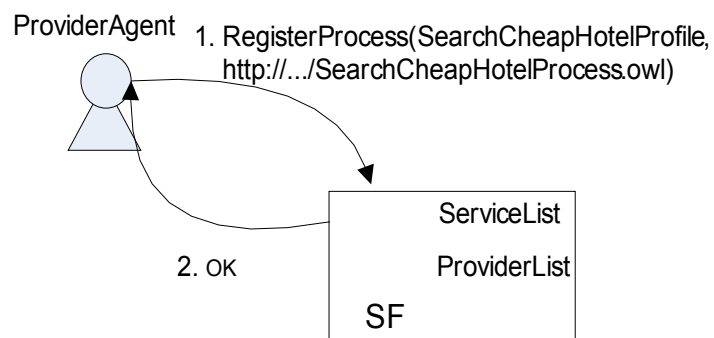
```
URL profile;
try {
    profile = new URL(URLProfile);
    oracle = new Oracle(profile);

} catch (MalformedURLException e) {
    logger.error("ERROR: Profile URL Malformed!");
    e.printStackTrace();
}

omsProxy.acquireRole(this,
    oracle.getProviderList().get(0), "travelagency");
```

### 3.2 Service registration

- Registering a new implementation of a service . In this case the service provider agent want to register your own service implementation SearchCheapHotel (RegisterProcess).



```
try {

    processDescription.setProfileID(results.get(0));

    sfProxy.registerProcess(this, processDescription);

} catch (Exception e) {
    logger.error(e.getMessage());
}
```

### 3.3 Execution of a service (mindswap)<sup>6</sup>

For the execution of the service we will have to use the api of mindswap, in particular we will use the call `execute`, this call takes two input parameters, a `Process` and input values are filled in by the client and that he was sending in the content of the message `ACLMessage`. To take these values we can make use of the two methods of `processDescription` 2.4.1

```
import org.mindswap.owls.OWLSFactory;
import org.mindswap.owls.process.Process;
import org.mindswap.owls.process.execution.ProcessExecutionEngine;
import org.mindswap.query.ValueMap;

// create an execution engine
ProcessExecutionEngine exec = OWLSFactory.createExecutionEngine();

try {
    Process aProcess = processDescription.getProcess(inmsg);

    // initialize the input values to be empty
    ValueMap values = new ValueMap();

    values = processDescription.getServiceRequestValues(inmsg);

    values = exec.execute(aProcess, values);
} catch (Exception e) {
    e.printStackTrace();
}
```

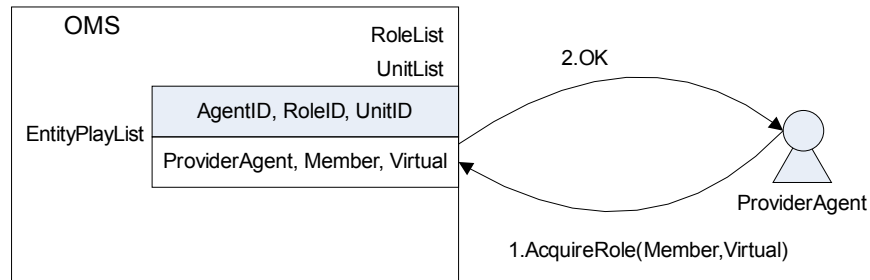
---

<sup>6</sup> <http://www.mindswap.org/>

## 4 How to program an agent client.

### 4.1 Register and Acquire rol.

- Registration of an agent on the platform Thomas. The agents request to register as members of the platform THOMAS across the service AcquireRole::

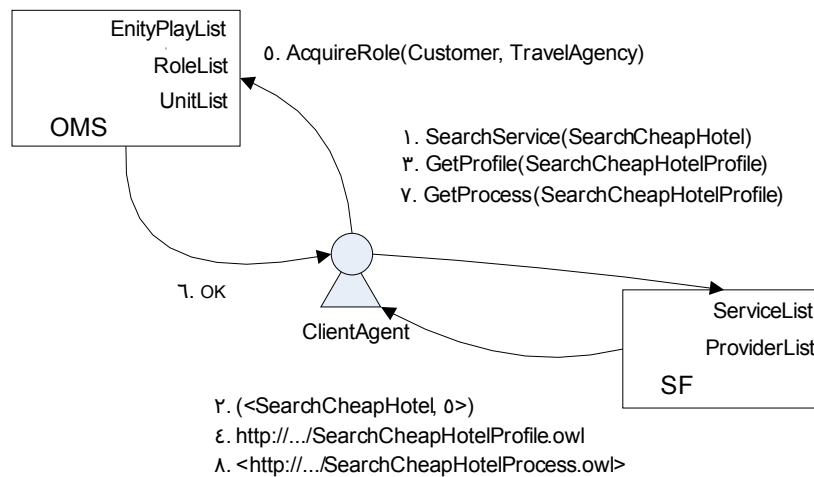


```
try {  
    omsProxy.acquireRole(this, "member", "virtual");  
} catch (Exception e) {  
    logger.error(e.getMessage());  
}
```



## 4.2 Search Service.

- Search of a service . The first step, which an agent who wants to use some functionality in THOMAS, must follow is to register as member of THOMAS. Later it can use the service SearchService to find services of his interest. After it the agent client obtains a more detailed information of the service in that he is interested across the analysis of the profile of the service (getProfile). To obtain information about the service implementation that allows you to use it, must acquire the client role, in this case must acquire rol Customer inside the TravelAgency. Once have registered with the client role of the service can request the service SF GetProcessGetProcess and obtain information on implementation.



```

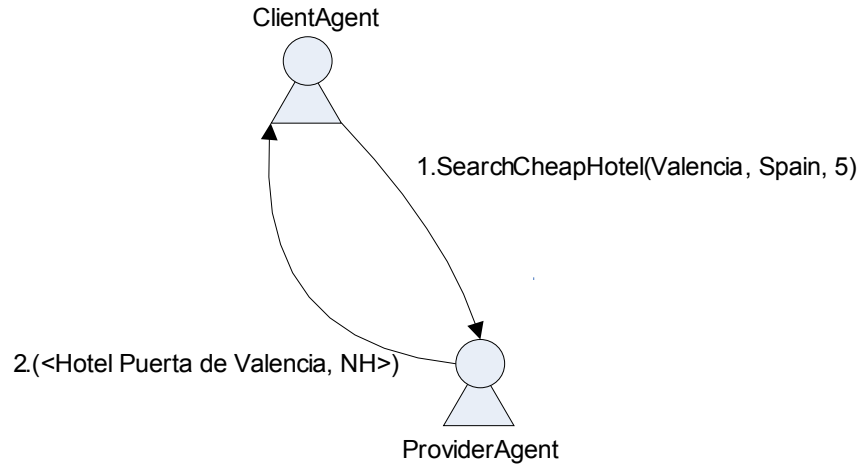
try {
    result = OMSservices.acquireRole(this, "member", "virtual");
do {
    results = SFservices.searchService(this, "SearchCheapHotel");
} while (results.size() == 0);

URLProfile = SFservices.getProfile(this, results.get(0));
URL profile;
try {
    profile = new URL(URLProfile);
    oracle = new Oracle(profile);
} catch (MalformedURLException e) {
    logger.error("ERROR: Profile URL Malformed!");
    e.printStackTrace();
}
OMSservices.acquireRole(this, oracle.getClientList().get(0),
"travelagency");

agents = SFservices.getProcess(this, results.get(0));
} catch (Exception e) {
    logger.error(e.getMessage());
}
  
```

### 4.3 Request of services.

- Request a service . After the agent analyzing the details of the implementation, that is to say, the process of the service client can contact with the agent supplier and realize the consultation corresponding to the search of a hotel:



```
try {
    ArrayList<String> arg = new ArrayList<String>();

    int i = 0;
    for (String input : oracle.getInputs()) {
        switch (i) {
            case 0:
                System.out.println("Input: " + input);
                arg.add("5");
                break;
            case 1:
                System.out.println("Input: " + input);
                arg.add("Spain");
                break;
            case 2:
                System.out.println("Input: " + input);
                arg.add("Valencia");
                break;
        }
        i++;
    }

    Enumeration<AgentID> agents1 = agents.keys();

    AgentID agentToSend = agents1.nextElement();

    URLProcess = agents.get(agentToSend);

    // call the service SearchCheapHotel
```

```

        lista = SFservices.genericService(this, agentToSend,
        URLProfile, URLProcess, arg);

        Enumeration<String> e = lista.keys();

        while (e.hasMoreElements()) {
            String key = e.nextElement();
            System.out.println(" " + key + " = " + lista.get(key));
        }
    } catch (Exception e) {
        logger.error(e.getMessage());
    }
}

```

## 5 Other examples: Special Agents and Agents initialization.

### 5.1 Agent Payee

- Registering an agent of type payee. The new agent requests service again role acquisition to register as a payee within the unit TravelAgency:

```

try {
    omsProxy.acquireRole(this, "payee", "travelagency");
} catch (Exception e) {
    logger.error(e.getMessage());
}

```

To prevent other external agents assume the role payee's agent registers a new rule of incompatibility in the system. This rule makes it impossible for other agents to assume the role payee:

```

omsProxy.registerNorm(this, "normal",
"FORBIDDEN_Member_REQUEST_acquireRole_MESSAGE(CONTENT(ROLE_'Payee'
))");

```

### 5.2 Agent Announcement

This agent register a profile in the registry of SF.

```

ProfileDescription profile = new ProfileDescription(

"http://localhost:8080/SearchCheapHotel/owl/owls/SearchCheapHotelP
rofile.owl",
    "SearchCheapHotel");

public void execute() {

    logger.info("Executing, I'm " + getName());
    String result;

```

```

try {
    omsProxy.acquireRole(this, "member", "virtual");
    omsProxy.acquireRole(this, "provider", "travelagency");

    sfProxy.registerProfile(this, profile);
    logger.info("[BroadCastAgent]The operation register
Profile return: " + profile.getServiceID() + "\n");
} catch (Exception e) {
    logger.error(e.getMessage());
}

```

### 5.3 Agents initialization

Briefly explain some details of the initialization and execution of agents. First we see the configuration properties , then we connect to the Qpid broker and agents launched the OMS and Sf. In addition we create agents (explained in previous sections), as a detail important to see that we must first throw a few agents and then we wait some time to launch the rest.

```

DOMConfigurator.configure("configuration/login.xml");
Logger logger = Logger.getLogger(Run.class);

/**
 * Clean database
 */
CleanBD clean = new CleanBD();
clean.clean_database();

/**
 * Connecting to Qpid Broker, default localhost.
 */
AgentsConnection.connect();
try {
    /**
     * Instantiating a OMS and FS agent's
     */
    OMS agenteOMS = OMS.getOMS();
    agenteOMS.start();
    SF agenteSF = SF.getSF();
    agenteSF.start();
    /**
     * Execute the agents
     */
    AgentPayee payeeAgent = new AgentPayee(new
AgentID("agentPayee"));
    AgentProvider providerAgent = new AgentProvider(new
AgentID("providerAgent"));
    AgentAnnouncement registerAgent = new

```

```

AgentAnnouncement(new AgentID("registerAgent"));
    AgentClient clientAgent = new AgentClient(new
AgentID("clientAgent"));

    registerAgent.start();
    payeeAgent.start();

    es.upv.dsic.gti-ia.architecture.Monitor m = new
es.upv.dsic.gti-ia.architecture.Monitor();
    m.waiting(10 * 1000);
    providerAgent.start();
    m.waiting(10 * 1000);
    clientAgent.start();

} catch (Exception e) {
    logger.error(e.getMessage());
}

```






**Please note!!!**Whenever we execute the example they will have to stop the services sf and oms, we it can do in an individual service or stopping and returning to start the whole container of servlets.



## 6 Initialization tasks

Finally, we will explain how to specify parameters for thomas and jena and thomas database via a configuration file called Settings.xml. (Remember that there are only two configuration files, login.xml and Setting.xml).

Properties for thomas owl's, this indicates where the services are deployed , the file is set to localhost. We can use the method OMSPProxy() and the method SFProxy(). Note that if you do not specify all parameters in Setting.xml file, you must not use the OMSPProxy() and SFProxy() method.

	Properties thomas
▼  entry	
@ key	OMSServiceDescriptionLocation
	http://localhost:8080/omsservices/OMServices/owl/owls/
▼  entry	
@ key	SFServiceDescriptionLocation
	http://localhost:8080/sfservices/SFservices/owl/owls/

Here we can find the configuration of the system management of database Mysql, the database thomas (thomas.sql) that comes attaches with the distributable one has these parameters formed.

!--	Properties mysql
▼ [e] entry	
@a key	serverName
	localhost
▼ [e] entry	
@a key	databaseName
	thomas
▼ [e] entry	
@a key	userName
	thomas
▼ [e] entry	
@a key	password
	thomas

!--	Properties jena
▼ [e] entry	
@a key	dbURL
	jdbc:mysql://localhost/thomas
▼ [e] entry	
@a key	dbType
	MySQL
▼ [e] entry	
@a key	dbDriver
	com.mysql.jdbc.Driver

It also should be checked that the data contained in the file settings.xml conform to the actual settings for both the existing database to the location of Web services.

## References

[1] E. del Val , N. Criado , M. Rebollo , E. Argente and V. Julian.: Service-Oriented Framework for Virtual Organizations