Develop agents for Magentix2.

1. Agents for thomas organization.
2. Agents Magentix2. Implementing protocols.

Thomas_Example. Develop agent for thomas organization.

In order to create agents that make use of the functionality of thomas we create an agent that extends from QueueAgent.

```
public class ClientAgent extends QueueAgent {

	public ClientAgent(AgentID aid) throws Exception {

		super(aid);

	}
```

We will have to create two variables a proxy for the OMS and the SF, these variables we encapsulate calls to SF and OMS, and offers us all the APIs thomas.

```
	//We create the class that will make us the agent proxy oms, facilitates access to the methods of the OMS
	OMSProxy OMSservices = new OMSProxy();

	//We create the class that will make us the agent proxy sf,  facilitates access to the methods of the SF
	SFProxy SFservices = new SFProxy();
```

If we have a service will create a new variable for each type of service that we SFServiceDescription, the builder will receive as a parameter where the owl are deployed,the profile and the process.

```
	//We create a SFServiceDescription, one for service that we have
		SFServiceDescription serviceOne = new SFServiceDescription("http://localhost:8080/broadcastservices/owl/owls/","http://localhost:8080/broadcastservices/owl/owls/");
		SFServiceDescription serviceTwo = new SFServiceDescription("http://localhost:8080/sfservices/THservices/owl/owls/","http://localhost:8080/sfservices/THservices/owl/owls/");
```

SFProxy

Service	Input		Output
DeregisterProfile	QueueAgent	SFAgentDescription	Exception
GetProcess	QueueAgent	ServiceID :String	ArrayList<AgentID>
GetProfile	QueueAgent	ServiceID:String	String
ModifyProcess	QueueAgent	SFAgentDescription	Exception
ModifyProfile	QueueAgent	SFAgentDescription	Exception

Register Process	QueueAgent	SFAgentDescription	Exception
Register Profile	QueueAgent	SFAgentDescription	Exception
Remove Provider	QueueAgent	SFAgentDescription	Exception
SearchService	QueueAgent	serviceGoal :String	ArrayList<String>
Generic Service	QueueAgent, AgentProvider URLProfile URLProcess ArrayArguments		ArrayList

OMSProxy

Service	Input							Output	
RegisterNorm	QueueAgent		NormID			NormContent		String	Exception
RegisterRole	QueueAgent	RegisterRoleInputRoleID	UnitID	Accessibility	Position	Visibility	Inheritance	String	Exception
RegisterUnit	QueueAgent		UnitID	Type		Goal	ParentUnitID	String	Exception
DeregisterNorm	QueueAgent				NormID			String	Exception
DeregisterRole	QueueAgent		RoleID			UnitID		String	Exception
DeregisterUnit	QueueAgent				UnitID			String	Exception
AcquireRole	QueueAgent		RoleID			UnitID		String	Exception
LeaveRole	QueueAgent		AgentID		RoleID	UnitID		String	Exception
Expulse	QueueAgent		AgentID		RoleID	UnitID		String	Exception
InformAgentRole	QueueAgent				AgentID			ArrayList<String>	Exception
InformMemebers	QueueAgent		RoleID			UnitID		ArrayList<String>	Exception

InformRoleNorms	QueueAgent	RoleID		ArrayList<String>	Exception
InformRoleProfiles	QueueAgent	UnitID		ArrayList<String>	Exception
InformUnit	QueueAgent	UnitID		ArrayList<String>	Exception
InformUnitRoles	QueueAgent	UnitID		ArrayList<String>	Exception
QuantityMembers	QueueAgent	RoleID	UnitID	Int	Exception

Launching the new agents and the SF and OMS agents.

First create the logger configuration variables.

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

Then you have to clean the database for the new execution.

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

We create the connection to the Qpid broker, the default values are in configuration file settings.xml.

```
AgentsConnection.connect();
```

Launching agents SF, OMS and new agents.

```
try
{

  /**
* Instantiating a OMS and FS agent's
*/
  OMS agenteOMS = OMS.getOMS();
  agenteOMS.start();


  SF agenteSF = SF.getSF();
  agenteSF.start();


  /**
```

```
        * Instantiating a BroadCast agent
        */
  BroadCastAgent broadCastagent = new BroadCastAgent(new AgentID("BroadCastAgent"));

  /**
        * Instantiating a ClientAgent agent
        */
  ClientAgent clientAgent = new ClientAgent(new AgentID("ClientAgent"));

  /**
        * Execute the agents
        */

  broadCastagent.start();
  clientAgent.start();


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

}
```

Client and Provider Agents

Abstract

Development of a new agent client and a provider agent, is modeled as a unit (travelagency) within which are provided search services tourist information and booking of hotels and flights.

Two types of roles within the unit interact travelagency: the role client (customer) and the role of service provider (provider).

Client Agent:

The client agent searches the provider agents which offer the service SearchCheapHotel into unit TravelAgency

```
//acquired the member role at the organization

      result = OMSservices.acquireRole(this, "member", "virtual");

      result = OMSservices.acquireRole(this,"customer", "travelagency");

//waiting that the agentBroadcast registered service SearchCheapHotel

      do{
            results = SFservices.searchService(this, "SearchCheapHotel");
      }while(results.size()==0);
```

```
	agents = SFservices.getProcess(this, results.get(0));



	for (AgentID agent : agents)
			System.out
					.println("[ClientAgent] agents who have the service SearchCheapHotel: "
							+ agent.name+"\n");


	String profile = SFservices.getProfile(this,results.get(0));



	ArrayList<String> arg = new ArrayList<String>();
	arg.add("FirstParam");
	arg.add("SecondParam");
	arg.add("ThirdParam");

	//call the service SearchCheapHotel

	SFservices.genericService(this,agents.get(0),profile,"http://localhost:8080/sfservices/THservices/owl/owls/Sea
rchCheapHotelProcess.owl", arg);
```

Provider Agent.

The provider agent register the service SearchCheapHotel and add the new task with responder rol to attend the request.

```
//We create the class that will make us the agent proxy oms,facilitates access to the methods of the OMS
      OMSProxy OMSservices = new OMSProxy();

      //We create the class that will make us the agent proxy sf,facilitates access to the methods of the SF
      SFProxy SFservices = new SFProxy();


      SFServiceDescription serviceOne = new SFServiceDescription("http://localhost:8080/sfservices/THservices/
owl/owls/","http://localhost:8080/sfservices/THservices/owl/owls/");

      String result;

      ArrayList<AgentID> agents;



      try
      {

            result = OMSservices.acquireRole(this, "member","virtual");

            System.out.println("[ProviderAgent]Acquire Role result: "+result+"\n");

            OMSservices.acquireRole(this,"provider", "travelagency");
```

```
        //Initializing services


                    serviceOne.setServiceGoal("SearchCheapHotel");


                    SFservices.registerProfile(this,serviceOne);
                    System.out.println("[ProviderAgent]The operation register Profile return: "+  serviceOne.getID()
+"\n");

                    SFservices.registerProcess(this, serviceOne);
                    System.out.println("[ProviderAgent]The operation register Process return: "+
serviceOne.getImplementationID()+"\n");



                     ArrayList<String> serviceProfile = new ArrayList<String>();

                    System.out.println("[ProviderAgent]Register Porcess return: "+ serviceTwo.getImplementationID()+"\
n");


                  //Rol responder
                    Responder responder = new Responder(this);

                    this.addTask(responder);

                    //when we do not have to create more roles we await the expiration of the other roles
                   Monitor m = new Monitor();
                   m.waiting();

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

!!!!WAIT We must stop the tomcat server every time we stop the execution.

Develop agent for Magentix2. Implementing protocols

Agent Responder.

```
public class myClassResponder extends QueueAgent {


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

      protected void execute() {

            MessageTemplate plantilla = new MessageTemplate(
                        InteractionProtocol.FIPA_REQUEST);

            ManejadorResponder responder = new ResponderManagement(this, plantilla);

            this.addTask(responder);

            while(true){}
      }
class ResponderManagement extends FIPARequestResponder {
            public ResponderManagement(QueueAgent a, MessageTemplate mt) {
                  super(a, mt);
            }
      .
      .
      .

      }

}
```

Agent Initiator.

```
public class myClassInitiator extends QueueAgent {


	private Monitor monitor = new Monitor();


	public myClassInitiator(AgentID aid) throws Exception {

		super(aid);
	}
	protected void execute() {

		ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
		// for (int i = 0; i < args.length; ++i)
		msg.setReceiver(new AgentID("HospitalAgent"));
		msg.setProtocol(InteractionProtocol.FIPA_REQUEST);
		msg.setContent("message");
		msg.setSender(this.getAid());

		this.addTask(new InitiatorManagement(this, msg));

		monitor.waiting(100);
	}

class InitiatorManagement extends FIPARequestInitiator {
		public ManejadorInitiator(QueueAgent a, ACLMessage msg) {
			super(a, msg);
		}
	.
	.
	.
	}
```

Launching agents.

```
public static void main(String[] args) {

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

	/**
	 * Connecting to Qpid Broker, default localhost.
	 */
	AgentsConnection.connect();

	/**
	 * Instantiating a Hospital agent
	 */
	myClassResponder myClass1 = new myClassResponder(new AgentID("HospitalAgent"));


	/**
	 * Instantiating a witness agent
	 */
	myClassInititator myClass2 = new myClassInititor(new AgentID("witnesAgent"));


	/**
	 * Execute the agents
	 */
	myClass1.start();
	myClass2.start();
	}catch(Exception e){

		logger.error(e.getMessage());
	}
```