User's Manual

EMFGormas v.2.0

Tool and Integrated Development Environment

Mario Rodrigo Solaz - mrodrigo@dsic.upv.es

GTI IA
Grupo de Tecnología Informática
Inteligencia Artificial

April 2012

History Version

Date	Version	Update
28.06.11	1	Creation.
28.12.11	2	Update Agents from *QueueAgent* to *CAgent*. Added code to provide internal and external services. Created a conversation to shutdown agents. Added smart shutdown (no more cleanDB methods are needed). Created a java file (*Utils.java*) which provides methods to store data locally (no need to query OMS nor SF agents). Created a *Role.xpt* file to mantain all Role related translation rules.
17.04.12	3	Moved to *Role.xpt* file all acquire role related translation rules. Changed the way to show results from *RegisterService* and *DeregisterService*. Used the user's selected value for *UnitType* parameter from model. Previously, this parameter always had taken the value *FLAT*. Changed position for Shutdown conversation to give it the most importance. Created a java file (*Constants.java*) which provides constants to simplify the election of *UnitType*, *Accessibility*, *Position* and *Visibility*.

Contents

History Version I

1 Introduction **1**

2 Installation **3**
2.1 Requirements 3
2.1.1 Java Development Kit 3
2.1.2 EMFGormas v.2 3

3 Using the software **7**
3.1 Option 1: Main menu entry 7
3.2 Option 2: Context menu entry 8

4 Applying M2T in EMFGormasv.2 **11**
4.1 Organizational Unit 13
4.2 Agent 13
4.3 Role 14
4.4 Services 15
4.5 Example: Calculator 16

A Run.java file **19**

B GodAgent.java file **21**

C AdditionAgent.java file **35**

D ProductAgent.java file **43**

E JamesAgent.java file **51**

F Utils.java file **65**

G Constants.java file **73**

List of Figures

2.1 Eclipse downloads webpage. 4

3.1 New entry in main menu. 8
3.2 Dialog where user selects the gormas model to translate. 9
3.3 Dialog where user selects the path where generated code will be saved. 9
3.4 Context menu with the new option for code generation. 9

4.1 EMFGormas v.2 metamodel Organization External View. 13
4.2 EMFGormas v.2 metamodel Structural View. 13
4.3 EMFGormas v.2 metamodel Agent View. 14
4.4 EMFGormas v.2 metamodel Structural View. 14
4.5 EMFGormas v.2 metamodel Structural View. 14
4.6 EMFGormas v.2 metamodel Agent View. 15
4.7 EMFGormas v.2 metamodel Service Port View. 15
4.8 Calculator Structural Diagram. 16
4.9 Calculator Organization External Diagram. 16
4.10 Calculator Agent Diagram. 17
4.11 Calculator Service Port Diagram. 17

CHAPTER

1 Introduction

EMFGormas v.2[1] is an approach for modeling Service-oriented Open Multiagent Systems using the MDA Eclipse Technology. It offers a CASE tool based on a unified meta-model for engineering large-scale open systems in which the constituent entitites interact among them by means of services. Due to the fact that EMFGormas v.2 is developed under the MDA Eclipse[2] Technology, all functionality inside this software, is created and packed in plugins format. In this way, functionalities (plugins) are independent, and new ones can be added without modifying or interfering with old ones.

This document describes a new functionality to make the Multi-Agent Systems' coding process easier. With this new plugin, the EMFGormas v.2 diagrams can be transformed into java code in an automatic way. So, the user can use the same application to define diagrams with *Organizational Units*, *Agents*, *Services*, etc., and to obtain, from them, java skeletons ready to fill and execute.

[1] `http://gti-ia.upv.es/sma/tools/EMFGormas/index.php`
[2] `http://www.eclipse.org/`

CHAPTER

2

Installation

2.1 Requirements 3

This functionality is developed under the MDA Eclipse Technology and it is integrated with the early functionalities of EMFGormas v.1. So it is mandatory to have a Java Development Kit installed and a EMFGormas v.2 copy ready to run, where the code generation function could be included.

2.1 Requirements

2.1.1 Java Development Kit

This software needs, at least, the Java SE Development kit 1.6 to run properly. You can get it from Sun's webpage (Windows/Unix[1] machines). Furthermore, you can get it from your favourite package management tool (Unix machines), like *APT*, *RPM Package Manager*, *Pacman*, etc.

2.1.2 EMFGormas v.2

There are two options for running EMFGormas v.2 tool. The first option is to download[2] the package which contains the full EMFGormas v.2 tool (Eclipse environment + plugins); and the

[1] `http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u25-download-346242.html`

[2] `http://gti-ia.upv.es/sma/tools/EMFGormas/downloads.php`

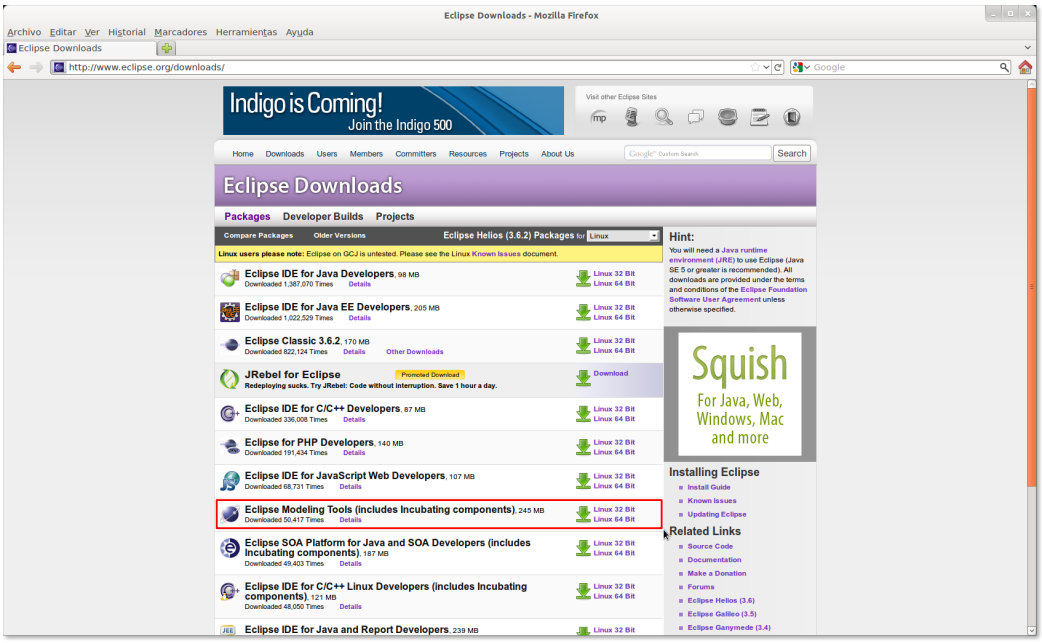

Figure 2.1: Eclipse downloads webpage.

second one is to download[3] the package with only the necessary plugins to run the CASE tool. Whichever option you choose, if you have problems to unzip the file, please try to unzip it in a shorter path[4] (for example, in */home/yourUserName/EMFGormas* or *C:\EMFGormas*).

2.1.2.1 Full EMFGormas v.2 tool

This package[3] includes Eclipse Helios 3.6 and all the necessary plugins to run the CASE tool. Choose your operating system and extract the file. You don't need to install Eclipse or the EMFGormas v.2 plugins. To run EMFGormas v.2 tool, you only need to double-click the *eclipse* executable file in the extracted folder.

2.1.2.2 EMFGormas v.2 plugins

This package[3] only includes the EMFGormas v.2 plugins. So, you must download a copy of Eclipse according to your operating system. Go to Eclipse downloads webpage[5], download Eclipse Helios 3.6 with the Modeling Tools (figure 2.1) and extract the file. Once eclipse is

[3] `http://gti-ia.upv.es/sma/tools/EMFGormas/downloads.php`
[4] For further information see `http://wiki.eclipse.org/SDK_Known_Issues`
[5] `http://www.eclipse.org/downloads/packages`

unzipped, double-click on the *eclipse* executable file and go to *Help / Install Modeling Components*. Make sure the checkbox *Incubation* is selected and type *Xpand* on the search box. Select this Model-to-Text component and push *Finish* to install it.

After that, download the EMFGormas v.2 plugins and unzip it into the eclipse folder. You don't need to install Eclipse or the EMFGormas v.2 plugins. To run EMFGormas v.2 tool you only need to double-click the *eclipse* executable file in the extracted folder.

CHAPTER

3

Using the software

3.1 Option 1: Main menu entry 7
3.2 Option 2: Context menu entry 8

The EMFGormas v.2 works like the previous one (EMFGormas v.1). So, you can draw, as usual on EMFGormas v.1, your systems with your organizational units, your agents and services, etc.[1] When you finish drawing, you can generate code from them. You have two ways to do this. The first one is a new entry called *Gormas Code Generation Tool* on main menu. And inside it, another menu entry with the title *Generate Code from model...* The other option is a context menu that is activated automatically every time you click with your secondary mouse button over a *.gormas* file.

3.1 Option 1: Main menu entry

This option, called *Gormas Code Generation Tool*, is located on main menu. It deploys a submenu with the text *Generate Code from model...* (figure 3.1), and when you click over it, launches a process that finishes with the code generation. This process needs two inputs in order to execute the code generation. These inputs are:

- The model you want to translate.
- The path where you want to save the generated code.

[1] EMFGormas metamodel manual can be found at `http://gti-ia.upv.es/sma/tools/EMFGormas/archivos/downloads/metamodels_manual.pdf`

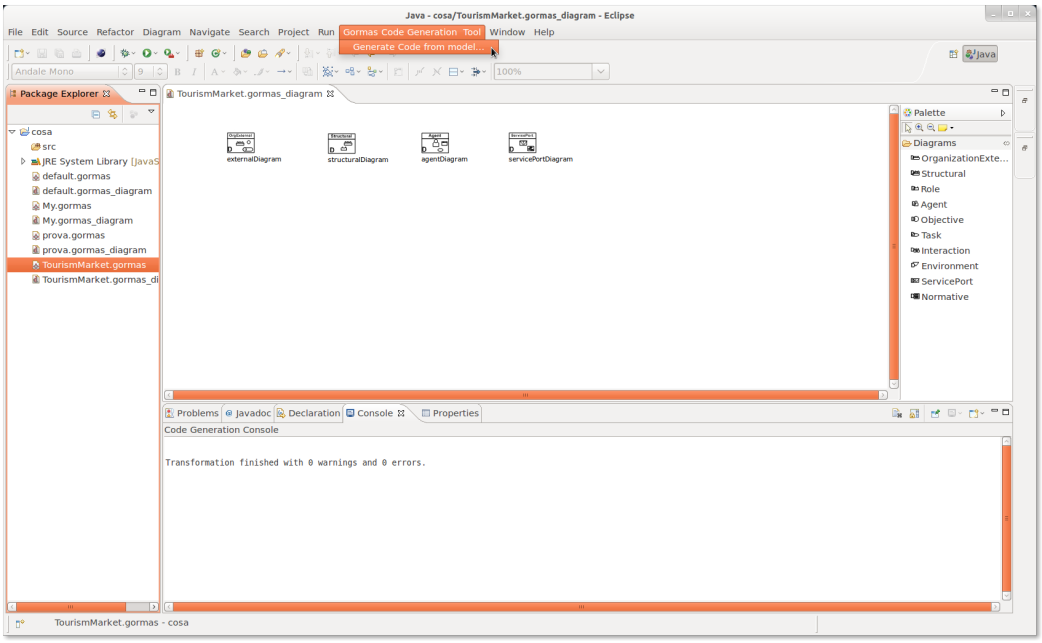

Figure 3.1: New entry in main menu.

This required information can be easily introduced through file dialogs (figures 3.2 and 3.3).

So you can draw your system with your organizational units, your agents and services; and when you finish drawing, just click on *Gormas Code Generation Tool / Generate Code from model...*, and select your model to translate, the path where you want to save the outputs and finally, your model will be translated to Java code and saved into the provided path.

3.2 Option 2: Context menu entry

This option, called *Generate Code from this model*, is located on context menu, and only appears when you click over *.gormas* files with your secondary mouse button (figure 3.4). This menu entry launches the same process as the above option, but in this case, the system only asks you for the path where it must save the output of the code generation (figure 3.3), thus the model to be translated is the one you have chosen (doing right-click on it).

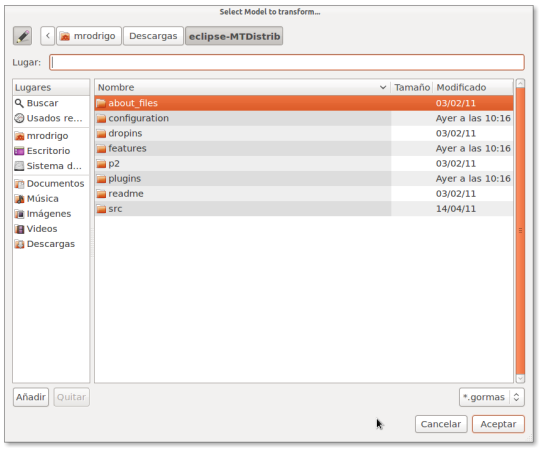

Figure 3.2: Dialog where user selects the gormas model to translate.

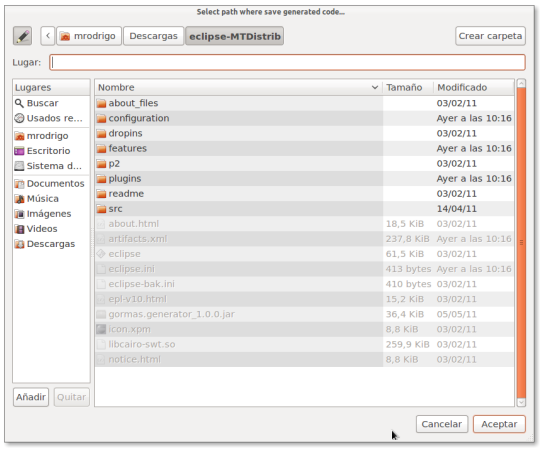

Figure 3.3: Dialog where user selects the path where generated code will be saved.

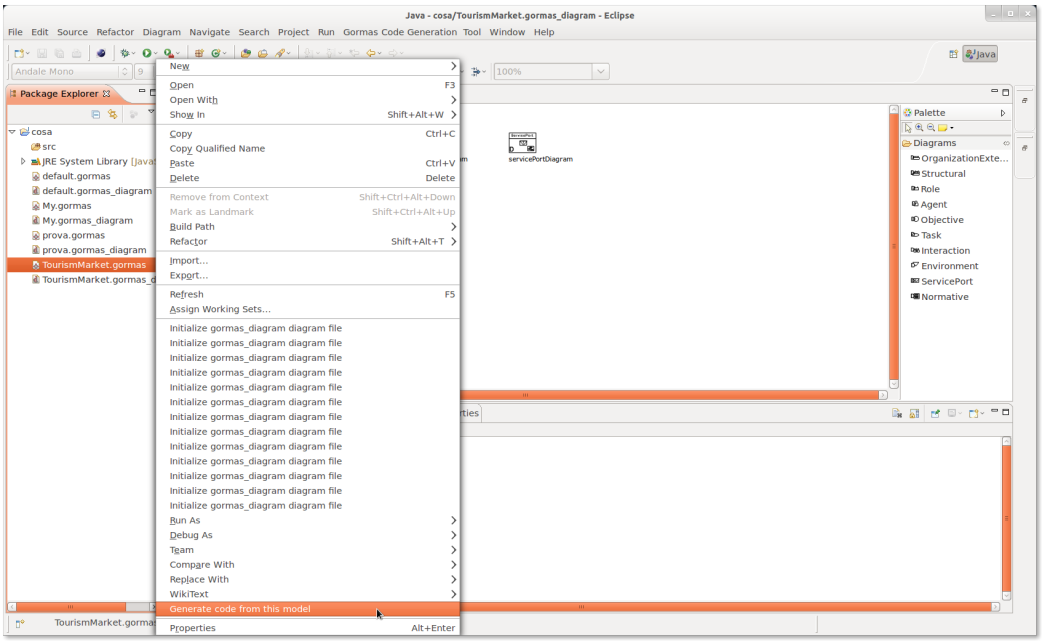

Figure 3.4: Context menu with the new option for code generation.

CHAPTER

4

Applying M2T in EMFGormasv.2

4.1 Organizational Unit 13
4.2 Agent 13
4.3 Role 14
4.4 Services 15
4.5 Example: Calculator 16

This chapter describes, what is and what is not translated by EMFGormas v.2 at this moment. The code generation process builds a set of Java skeletons. This generated code is executable on *THOMAS*[1] framework after the user fulfill some gaps. But this only is a starting point, so the user must complete the resultant files to fit his/her needs.

EMFGormas v.2 has ten types of diagrams (*OrganizationExternal*, *Structural*, *Role*, *Agent*, *Objective*, *Task*, *Interaction*, *Environment*, *ServicePort* and *Normative*), that user can employ to model his/her system. In this first version, only *OrganizationExternal*, *Structural*, *Agent*, and *ServicePort* diagrams are involved in transformation process. Each one of these selected diagrams contains several entities and, in this first version of EMFGormas v.2, not all of them are translated to code. The following sections shows what EMFGormas v.2's components are translated.

The result of the translation feature is a set of *Java* files. These files are not full filled. The user must fill the gaps in the files before running the generated code. The generated code provides a set of skeletons with appropriate instructions to create organizational units, roles, agents and services according to the proposed initial stage in the model drawn. This set of *Java* files is

[1] `http://gti-ia.upv.es/sma/tools/Thomas/index.php`

composed by:

Run.java : This file prepares the environment to run *THOMAS* framework and starts GodAgent.

GodAgent.java : This file contains the code responsible of creating the *OrganizationalUnit*s and the *Role*s associated to the *OrganizationalUnit*s drawn at user's model. This agent also contains the instructions that starts agents contained in user's model. One of the task associated to *GodAgent* is launching the shutdown conversation. In this conversation, the *participants* are the agents which GodAgent started. Morevover, the GodAgent is the *initiator* of the conversation. The aim of this conversation is that participants leave their *Role*s and deregister their services in order to obtain a clean database. Once the *participants* have accomplished their task, the *initiator* starts its cleaning procedure. *GodAgent* only cleans (deregister units and roles), the initial state contained in user's model. Anything created later, must be managed by the user.

Agent.java : There is one *java* file per agent contained in user's model. Each file contains the agent's actions:

- Methods to register services.
- Methods to deregister services.
- Methods to manage requests to him.
- Sequence of actions associated to itself. In this case: enter in *THOMAS*' organizations, register the services itself provides (if any) and wait for other agents' requests.
- Methods to revert all the creation actions he has made (*OrganizationalUnit*s, *Role*s and services).

Utils.java : This file contains methods for manage everything related to *OrganizationalUnit*s, *Role*s, etc in *THOMAS* framework.

Constants.java : This file contains constants to make easier to user choose type of parameters *Unit*, *Accessibility*, *Position* and *Visibility*.

After explaining this, the manual focuses on the decisions taken to translate each entity of EMFGormasv.2.

4.1 Organizational Unit

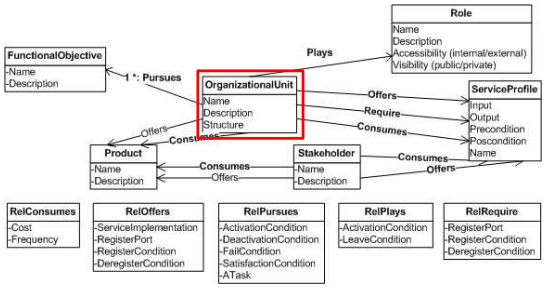

Figure 4.1: EMFGormas v.2 metamodel Organization External View.

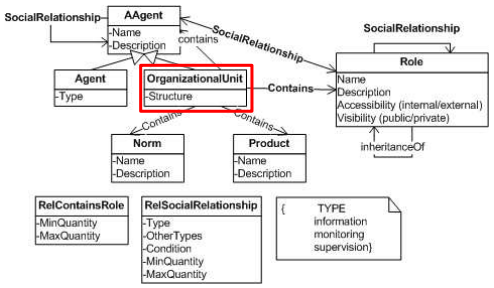

Figure 4.2: EMFGormas v.2 metamodel Structural View.

An *OrganizationalUnit* entity can be drawn in several EMFGormas v.2's diagrams. However, for this version of code generation tool, only two of them have been selected: *OrganizationExternal* and *Structural* views (figures 4.1 and 4.2).

The user can draw entities of this type to model his/her system, and when the user launches the translation functionality, the tool retrieves all the data available (*Name*, *Structure*, *Role*s contained, etc.), of each *OrganizationalUnit* and generates code accordingly. For this entity, the rule writes a set of instructions that will register, at runtime, a new unit with *OrganizationalUnit*'s name.Moreover, it navigates through *RelContainsRole* relations contained inside each *OrganizationalUnit* and it writes into *GodAgent.java* file the needed instructions to register, at runtime, the *Role*s inside the *OrganizationalUnit*s accordingly.

4.2 Agent

The *Agent* entity, can be drawn in several EMFGormas v.2's diagrams. However, for this version of code generation tool, only two of them have been selected: *Agent* and *Structural* views (figures 4.3 and 4.4).

When the translation functionality is launched, the tool retrieves all the data available (Name, Roles played, Services used and offered, etc.), of each *Agent* and generates code accordingly. For this entity, the rule creates a new file for each *Agent*. With each register process, it is included the set of instructions to save the new *Role* or *OrganizationalUnit* at the provided structures.

Similarly, the tool writes in file the actions to manage the *Shutdown* conversation, where the agent must deregister his services and undo the register of *Role*s and *OrganizationalUnit*s in

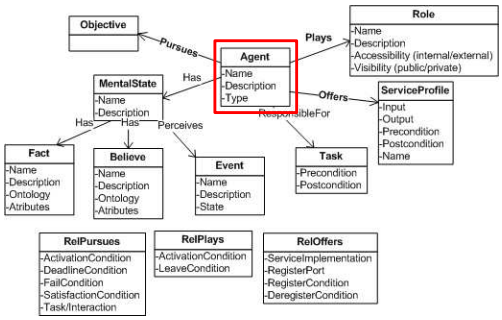

Figure 4.3: EMFGormas v.2 metamodel Agent View.

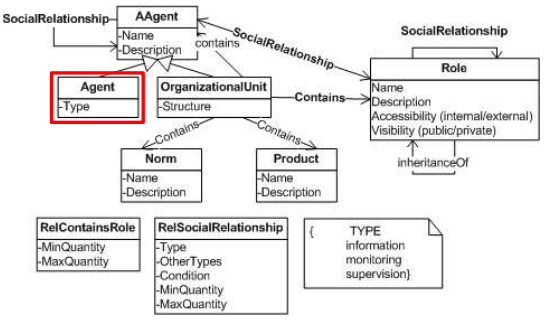

Figure 4.4: EMFGormas v.2 metamodel Structural View.

order to prepare itself to shutdown.

4.3 Role

The *Role* entity, can be drawn in several EMFGormas v.2's diagrams. However, for this version of code generation tool, only *Structural* view (figure 4.5) has been selected.

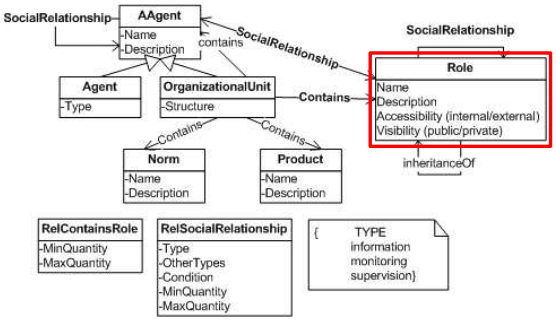

Figure 4.5: EMFGormas v.2 metamodel Structural View.

When the translation functionality is launched, the tool retrieves all the data available (Name, Description, etc.), of each *Role* and generates code inside *GodAgent.java* or *Agent.java* files accordingly. This code is the set of instructions that registers the *Agent* inside the *Organizational Unit* which contains it with specific role.

4.4 Services

The services can be included trough two entities: *ServiceProfile* and *ServiceImplementation*. Besides, if this service is not private and you want to be published in a Service Directory, you need to specify a *ServicePort* in *Service Port* view. This entities can be drawn in *Agent* and *Service Port* views (figures 4.6 and 4.7).

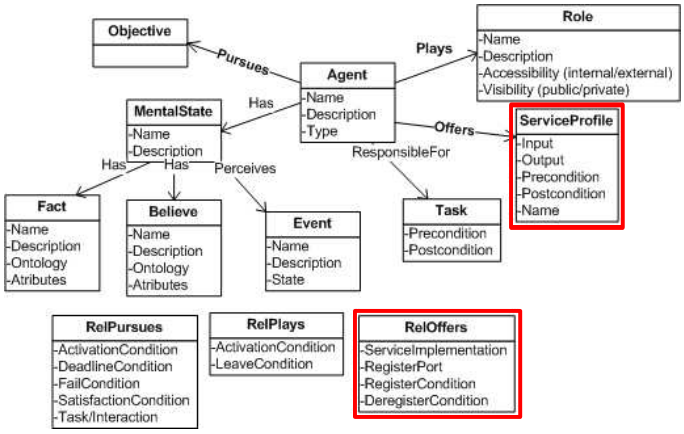

Figure 4.6: EMFGormas v.2 metamodel Agent View.

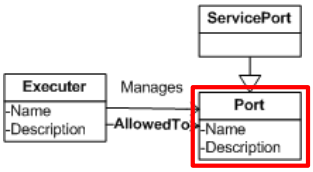

Figure 4.7: EMFGormas v.2 metamodel Service Port View.

When the translation functionality is launched, the tool retrieves all the data available (Name, Register Port where is published, etc.) of each *Service* and generates code inside *Agent.java* files accordingly of them. No matter if the service is private or is not. The rule generates the code correctly according with the user definition of it.

For this entity, if the service is not private, the rule registers it into *Service Facilitator (SF)* of *THOMAS*.

For all of them, private and not private services, the rule manages the related actions to habilitate the agent to receive and serve requests at this service. This set of actions is available at the appendix chapters C, D and E.

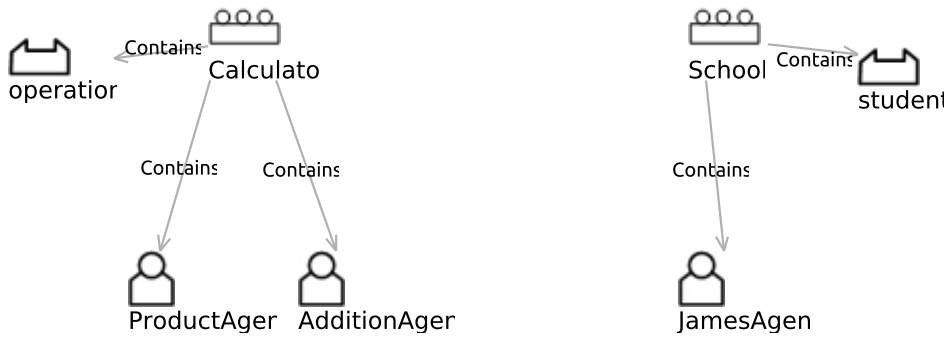

Figure 4.8: Calculator Structural Diagram.

Figure 4.9: Calculator Organization External Diagram.

4.5 Example: Calculator

This section shows you an example of MAS developed using all the above concepts. The figures 4.8, 4.9, 4.10 and 4.11 show the diagrams that serves as input for the code generation process. After defining each of those four diagrams, you can launch the code generation functionality and you will get a set of files with the same content as shown at Appendixes A, B, C, D and E.

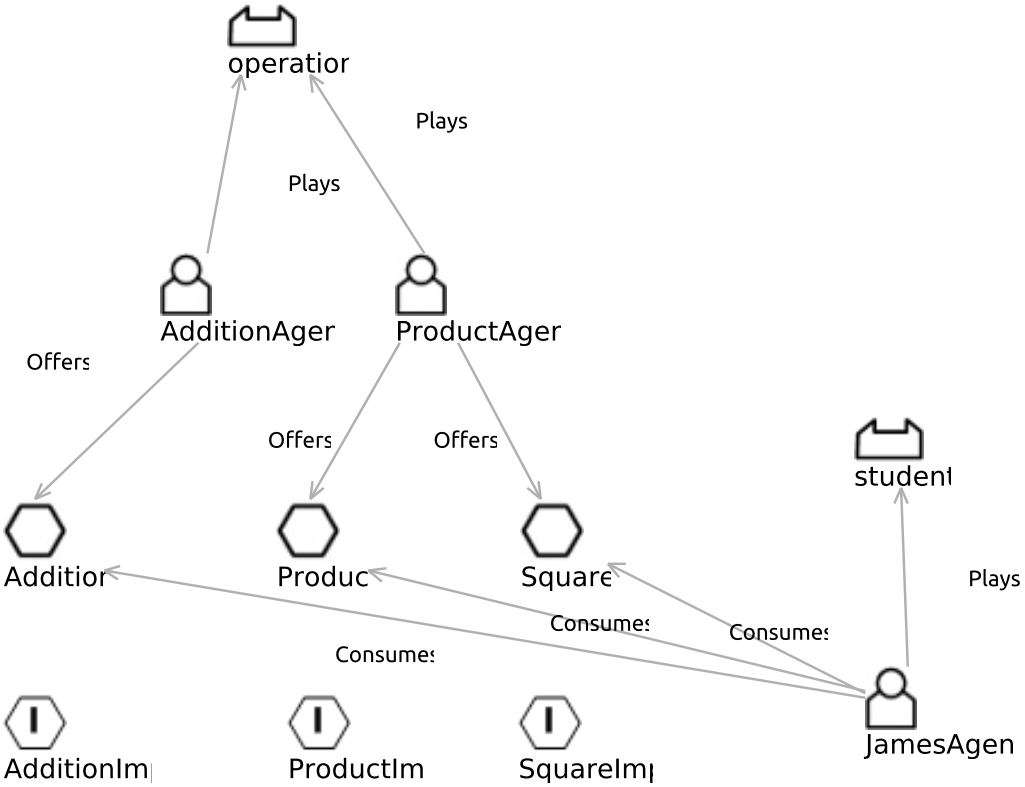

Figure 4.10: Calculator Agent Diagram.

Figure 4.11: Calculator Service Port Diagram.

APPENDIX

A

Run.java file

```java
/**
 * This class has been generated using Gormas2Magentix tool.
 *
 * @author Mario Rodrigo - mrodrigo@dsic.upv.es
 *
 */
package EMFGormas_Example;

import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;

import es.upv.dsic.gti_ia.core.AgentID;
import es.upv.dsic.gti_ia.core.AgentsConnection;

public class Run {

	public static void main(String[] args) {

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

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

		try {
			/**
			 * Execute the agent
			 */
			GodAgent godAgent = new GodAgent(new AgentID("godAgent"));
			godAgent.start();

		} catch (Exception e) {
			logger.error(e.getMessage());
		}
	} // End main
} // End of class
```

APPENDIX

B

GodAgent.java file

```java
/**
 * This class has been generated using Gormas2Magentix tool.
 *
 * @author Mario Rodrigo - mrodrigo@dsic.upv.es
 *
 */
package EMFGormas_Example;

import java.util.ArrayList;

import EMFGormas_Example.Constants.AccessibilityType;
import EMFGormas_Example.Constants.PositionType;
import EMFGormas_Example.Constants.UnitType;
import EMFGormas_Example.Constants.VisibilityType;
import EMFGormas_Example.Utils.LocalData;
import EMFGormas_Example.Utils.UnitRolePair;
import es.upv.dsic.gti_ia.architecture.Monitor;
import es.upv.dsic.gti_ia.cAgents.BeginState;
import es.upv.dsic.gti_ia.cAgents.BeginStateMethod;
import es.upv.dsic.gti_ia.cAgents.CAgent;
import es.upv.dsic.gti_ia.cAgents.CFactory;
import es.upv.dsic.gti_ia.cAgents.CProcessor;
import es.upv.dsic.gti_ia.cAgents.FinalState;
import es.upv.dsic.gti_ia.cAgents.FinalStateMethod;
import es.upv.dsic.gti_ia.cAgents.ReceiveState;
import es.upv.dsic.gti_ia.cAgents.ReceiveStateMethod;
import es.upv.dsic.gti_ia.cAgents.SendState;
import es.upv.dsic.gti_ia.cAgents.SendStateMethod;
import es.upv.dsic.gti_ia.cAgents.WaitState;
import es.upv.dsic.gti_ia.core.ACLMessage;
import es.upv.dsic.gti_ia.core.AgentID;
import es.upv.dsic.gti_ia.core.MessageFilter;
import es.upv.dsic.gti_ia.organization.OMSProxy;
import es.upv.dsic.gti_ia.organization.SFProxy;
import es.upv.dsic.gti_ia.organization.THOMASException;

public class GodAgent extends CAgent {

        // ----------------------------------------
        // FIELDS of the class
        // ----------------------------------------
        private OMSProxy omsProxy;
        private SFProxy sfProxy;
        private ArrayList<String> results;
        Monitor mon = null;
```

```
46          private LocalData myLocalData;
47
48          // ----------------------------------------
49          // CONSTRUCTOR of the class
50          // ----------------------------------------
51          /**
52           * Constructor of the class
53           * @param aid
54           * @throws Exception
55           */
56          public GodAgent(AgentID aid) throws Exception {
57                  super(aid);
58
59                  if (results == null) {
60                          results = new ArrayList<String>();
61                  }
62
63                  if (omsProxy == null) {
64                          omsProxy = new OMSProxy(this);
65                  }
66
67                  if (sfProxy == null) {
68                          sfProxy = new SFProxy(this);
69                  }
70
71                  if (myLocalData == null) {
72                          myLocalData = new LocalData();
73                  }
74
75          } // End of constructor
76
77          // ----------------------------------------
78          // METHODS of the class
79          // ----------------------------------------
80          // This is the main method of the agent
81          @Override
82      protected void execution(CProcessor firstProcessor, ACLMessage welcomeMessage) {
83          String result;
84          ArrayList<String> resultArrayList = null;
85          ArrayList<ArrayList<String>> serviceResultArrayList = null;
86
87          // Requesting for the list of roles and units in which the agent is in a
88          // specific moment.
89          CAgent cAgent = firstProcessor.getMyAgent();
90
91          try {
92              logger.info("[" + getName() + "] Initializing scenario.");
93
94              // Get the list of roles that agent is playing
95              ArrayList<UnitRolePair> agentPlayingRoles = Utils.queryOMSForRolesPlayedByAnAgent(cAgent, omsProxy,
                        logger);
96              for (UnitRolePair unitRolePair : agentPlayingRoles) {
97                  myLocalData.addPlayingRole(unitRolePair);
98              }
99
100             UnitRolePair desiredRole = new UnitRolePair("participant", "virtual");
101
102             if (!myLocalData.getPlayingRoles().contains(desiredRole)) {
103                 // Acquire Role in main Organization
104                 result = omsProxy.acquireRole(desiredRole.getRoleID(), desiredRole.getUnitID());
105
106                 if (result.contains("acquired")) {
107                     // Add the pair Unit - Role to playingRole ArrayList
108                     myLocalData.addPlayingRole(desiredRole);
109                 }
110
111                 logger.debug("[" + getName() + "] Entering in THOMAS: " + result + "\n");
112
113             } else {
```

```
                logger.debug("[" + getName() + "] is already inside THOMAS.");
            }

        } catch (THOMASException e) {
            // If an error occurs trying to acquire role "participant" in
            // "virtual" organization, the agent will not be able to do any
            // action in future. So the only option is exit from the app with
            // the abnormally exit code (1).
            logger.error("[" + getName() + "] Application ended abnormally. Please review your code, your
                 configuration and your database.\n" + e.getContent());

            // Ends the application abnormally
            System.exit(1);
        }









    try{
        // Not exists, so RegisterUnit "Calculator"
        result = omsProxy.registerUnit("Calculator",
 UnitType.TEAM.toString(),
        null, "creator");
        if (result.equalsIgnoreCase("Calculator created")) {
            // Add the Virtual Organization to createdOrganizations
            // ArrayList
            myLocalData.addCreatedUnit("Calculator");

            // Add the pair Unit - Role to playingRole ArrayList
            myLocalData.addPlayingRole(new UnitRolePair("creator", "Calculator"));
        }

        logger.debug("[" + getName() + "] Register Unit \"Calculator\" result: " + result + "\n");

    } catch (THOMASException e) {
        logger.error("[" + getName() + "] Application ended abnormally. Please review your code, your
             configuration and your database.\n" + e.getContent());
        // Ends the application abnormally
        System.exit(1);
    }

    try {
        // Query for roles registered inside "Calculator"
        serviceResultArrayList = omsProxy.informUnitRoles("Calculator");
        ArrayList<String> containedRoles = new ArrayList<String>();

        for (ArrayList<String> tuple : serviceResultArrayList) {
            // Role's name
            containedRoles.add(tuple.get(0));
            logger.debug("[" + getName() + "] Added the role \"" + tuple.get(0) + "\" to the contained roles array
                   for the Unit \"Calculator\".\n");
        }




// Register role "operation"
if (!containedRoles.contains("operation")) {
    result = omsProxy.registerRole("operation", "Calculator", AccessibilityType.EXTERNAL.toString(),
          VisibilityType.PUBLIC.toString(), PositionType.MEMBER.toString());

    if (result.equalsIgnoreCase("operation created")) {
```

```
179             // Add the pair Unit - Role to createdRole ArrayList
180             myLocalData.addCreatedRole(new UnitRolePair("operation", "Calculator"));
181             logger.debug("[" + getName() + "] Register Role \"operation\" result: " + result + "\n");
182         }
183     }
184
185
186
187         } catch (THOMASException e) {
188             logger.error("[" + getName() + "] Application ended abnormally. Please review your code, your
                    configuration and your database.\n" + e.getContent());
189             // Ends the application abnormally
190             System.exit(1);
191         }
192
193
194
195
196
197
198
199
200
201
202
203
204
205         try{
206             // Not exists, so RegisterUnit "School"
207             result = omsProxy.registerUnit("School",
208      UnitType.FLAT.toString(),
209             null, "creator");
210             if (result.equalsIgnoreCase("School created")) {
211                 // Add the Virtual Organization to createdOrganizations
212                 // ArrayList
213                 myLocalData.addCreatedUnit("School");
214
215                 // Add the pair Unit - Role to playingRole ArrayList
216                 myLocalData.addPlayingRole(new UnitRolePair("creator", "School"));
217             }
218
219             logger.debug("[" + getName() + "] Register Unit \"School\" result: " + result + "\n");
220
221         } catch (THOMASException e) {
222             logger.error("[" + getName() + "] Application ended abnormally. Please review your code, your
                    configuration and your database.\n" + e.getContent());
223             // Ends the application abnormally
224             System.exit(1);
225         }
226
227         try {
228             // Query for roles registered inside "School"
229             serviceResultArrayList = omsProxy.informUnitRoles("School");
230             ArrayList<String> containedRoles = new ArrayList<String>();
231
232             for (ArrayList<String> tuple : serviceResultArrayList) {
233                 // Role's name
234                 containedRoles.add(tuple.get(0));
235                 logger.debug("[" + getName() + "] Added the role \"" + tuple.get(0) + "\" to the contained roles array
                        for the Unit \"School\".\n");
236             }
237
238
239
240
241
242     // Register role "student"
243     if (!containedRoles.contains("student")) {
```

```
244         result = omsProxy.registerRole("student", "School", AccessibilityType.EXTERNAL.toString(), VisibilityType.
                PUBLIC.toString(), PositionType.MEMBER.toString());
245
246         if (result.equalsIgnoreCase("student created")) {
247             // Add the pair Unit - Role to createdRole ArrayList
248             myLocalData.addCreatedRole(new UnitRolePair("student", "School"));
249             logger.debug("[" + getName() + "] Register Role \"student\" result: " + result + "\n");
250         }
251     }
252
253
254
255         } catch (THOMASException e) {
256             logger.error("[" + getName() + "] Application ended abnormally. Please review your code, your
                    configuration and your database.\n" + e.getContent());
257             // Ends the application abnormally
258             System.exit(1);
259         }
260
261
262
263
264
265
266
267             logger.info("[" + getName() + "] Scenario initialized.");
268
269
270
271         //Time to instantiate the agents
272
273
274         try {
275             // Instantiating ProductAgent agent
276             ProductAgent ProductAgentAgent = new ProductAgent(new AgentID("ProductAgent"));
277
278             //Execute the agent
279             ProductAgentAgent.start();
280             myLocalData.addStartedAgent(ProductAgentAgent);
281             logger.debug("[" + getName() + "] ProductAgentAgent started successfully.");
282
283         } catch (Exception e) {
284             logger.error("[" + getName() + "] Application ended abnormally during the creation of \"ProductAgentAgent
                    \".\n" + e.getMessage());
285             // Ends the application abnormally
286             System.exit(1);
287         }
288
289
290
291
292
293
294
295
296         try {
297             // Instantiating AdditionAgent agent
298             AdditionAgent AdditionAgentAgent = new AdditionAgent(new AgentID("AdditionAgent"));
299
300             //Execute the agent
301             AdditionAgentAgent.start();
302             myLocalData.addStartedAgent(AdditionAgentAgent);
303             logger.debug("[" + getName() + "] AdditionAgentAgent started successfully.");
304
305         } catch (Exception e) {
306             logger.error("[" + getName() + "] Application ended abnormally during the creation of \"AdditionAgentAgent
                    \".\n" + e.getMessage());
307             // Ends the application abnormally
308             System.exit(1);
```

```
309         }
310 
311 
312 
313 
314 
315 
316 
317 
318         try {
319             // Instantiating JamesAgent agent
320             JamesAgent JamesAgentAgent = new JamesAgent(new AgentID("JamesAgent"));
321 
322             //Execute the agent
323             JamesAgentAgent.start();
324             myLocalData.addStartedAgent(JamesAgentAgent);
325             logger.debug("[" + getName() + "] JamesAgentAgent started successfully.");
326 
327         } catch (Exception e) {
328             logger.error("[" + getName() + "] Application ended abnormally during the creation of \"JamesAgentAgent
                    \".\n" + e.getMessage());
329             // Ends the application abnormally
330             System.exit(1);
331         }
332 
333 
334 
335 
336 
337 
338 
339 
340 
341             // We create a factory in order to send a REQUEST and wait for the
342             // answer
343             class ShutdownAgentsFIPA_REQUEST {
344 
345                 int timesFirstWait = 1;
346                 int timesSecondWait = 1;
347 
348                 /**
349                  * Method to execute at the beginning of the conversation
350                  *
351                  * @param myProcessor
352                  *            the CProcessor managing the conversation
353                  * @param msg
354                  *            first message to send
355                  */
356                 protected void doBegin(CProcessor myProcessor, ACLMessage msg) {
357                     myProcessor.getInternalData().put("InitialMessage", msg);
358                 }
359 
360                 class BEGIN_Method implements BeginStateMethod {
361                     public String run(CProcessor myProcessor, ACLMessage msg) {
362                         doBegin(myProcessor, msg);
363                         if (((GodAgent) myProcessor.getMyAgent()).myLocalData.getStartedAgents().size() > 0) {
364                             return "REQUEST_REQUEST_INITIATOR";
365                         } else {
366                             return "FINAL_REQUEST_INITIATOR";
367                         }
368                     };
369                 }
370 
371                 /**
372                  * Sets the request message
373                  *
374                  * @param myProcessor
375                  *            the CProcessor managing the conversation
376                  * @param messageToSend
```

```
377                 *            request message
378                 */
379               protected void doRequest(CProcessor myProcessor, ACLMessage messageToSend) {
380                   for (CAgent agentToStop : ((GodAgent) myProcessor.getMyAgent()).myLocalData.getStartedAgents()) {
381                       messageToSend.addReceiver(agentToStop.getAid());
382                   }
383                   messageToSend.setSender(myProcessor.getMyAgent().getAid());
384                   messageToSend.setContent("Time to shutdown.");
385                   messageToSend.setHeader("shutdown", "ShutdownAgent");
386                   messageToSend.setPerformative(ACLMessage.REQUEST);
387                   for (int index = 0; index < messageToSend.getTotalReceivers(); index++) {
388                       logger.debug("[" + myProcessor.getMyAgent().getName() + "] I tell " + messageToSend.
                              getReceiver(index).name + " " + messageToSend.getPerformative() + " " + messageToSend.
                              getContent());
389                   }
390               }
391
392               class REQUEST_Method implements SendStateMethod {
393                   public String run(CProcessor myProcessor, ACLMessage messageToSend) {
394                       doRequest(myProcessor, messageToSend);
395                       return "FIRST_WAIT_REQUEST_INITIATOR";
396                   }
397               }
398
399               /**
400                * Method to execute when the initiator receives a not-understood
401                * message
402                *
403                * @param myProcessor
404                *            the CProcessor managing the conversation
405                * @param msg
406                *            not-understood message
407                */
408               protected void doNotUnderstood(CProcessor myProcessor, ACLMessage msg) {
409                   logger.debug("[" + myProcessor.getMyAgent().getName() + "] " + msg.getSender().name + " tell me: "
                          + msg.getPerformative() + " " + msg.getContent());
410               }
411
412               class NOT_UNDERSTOOD_Method implements ReceiveStateMethod {
413                   public String run(CProcessor myProcessor, ACLMessage messageReceived) {
414                       doNotUnderstood(myProcessor, messageReceived);
415                       if (timesFirstWait < ((GodAgent) myProcessor.getMyAgent()).myLocalData.getStartedAgents().size
                              ()) {
416                           timesFirstWait++;
417                           return "FIRST_WAIT_REQUEST_INITIATOR";
418                       } else {
419                           return "FINAL_REQUEST_INITIATOR";
420                       }
421                   }
422               }
423
424               /**
425                * Method to execute when the initiator receives a failure message
426                *
427                * @param myProcessor
428                *            the CProcessor managing the conversation
429                * @param msg
430                *            failure message
431                */
432               protected void doRefuse(CProcessor myProcessor, ACLMessage msg) {
433                   logger.debug("[" + myProcessor.getMyAgent().getName() + "] " + msg.getSender().name + " tell me: "
                          + msg.getPerformative() + " " + msg.getContent());
434               }
435
436               class REFUSE_Method implements ReceiveStateMethod {
437                   public String run(CProcessor myProcessor, ACLMessage messageReceived) {
438                       doRefuse(myProcessor, messageReceived);
439                       if (timesFirstWait < ((GodAgent) myProcessor.getMyAgent()).myLocalData.getStartedAgents().size
                              ()) {
```

```
440                         timesFirstWait++;
441                         return "FIRST_WAIT_REQUEST_INITIATOR";
442                     } else {
443                         return "FINAL_REQUEST_INITIATOR";
444                     }
445                 }
446             }
447
448             /**
449              * Method to execute when the initiator receives a agree message
450              *
451              * @param myProcessor
452              *            the CProcessor managing the conversation
453              * @param msg
454              *            agree message
455              */
456             protected void doAgree(CProcessor myProcessor, ACLMessage msg) {
457                 logger.debug("[" + myProcessor.getMyAgent().getName() + "] " + msg.getSender().name + " tell me: "
458                         + msg.getPerformative() + " " + msg.getContent());
459             }
460             class AGREE_Method implements ReceiveStateMethod {
461                 public String run(CProcessor myProcessor, ACLMessage messageReceived) {
462                     doAgree(myProcessor, messageReceived);
463                     // There are so many answers as Agents started
464                     if (timesFirstWait < ((GodAgent) myProcessor.getMyAgent()).myLocalData.getStartedAgents().size
465                             ()) {
466                         timesFirstWait++;
467                         return "FIRST_WAIT_REQUEST_INITIATOR";
468                     } else {
469                         return "SECOND_WAIT_REQUEST_INITIATOR";
470                     }
471                 }
472             }
473
474             /**
475              * Method to execute when the initiator receives a failure message
476              *
477              * @param myProcessor
478              *            the CProcessor managing the conversation
479              * @param msg
480              *            failure message
481              */
482             protected void doFailure(CProcessor myProcessor, ACLMessage msg) {
483                 logger.debug("[" + myProcessor.getMyAgent().getName() + "] " + msg.getSender().name + " tell me: "
484                         + msg.getPerformative() + " " + msg.getContent());
485             }
486
487             class FAILURE_Method implements ReceiveStateMethod {
488                 public String run(CProcessor myProcessor, ACLMessage messageReceived) {
489                     doFailure(myProcessor, messageReceived);
490                     if (timesSecondWait < ((GodAgent) myProcessor.getMyAgent()).myLocalData.getStartedAgents().
491                             size()) {
492                         timesSecondWait++;
493                         return "SECOND_WAIT_REQUEST_INITIATOR";
494                     } else {
495                         return "FINAL_REQUEST_INITIATOR";
496                     }
497                 }
498             }
499
500             /**
501              * Method to execute when the initiator receives a inform message
502              *
503              * @param myProcessor
504              *            the CProcessor managing the conversation
505              * @param msg
506              *            inform message
507              */
508             protected void doInform(CProcessor myProcessor, ACLMessage msg) {
```

```
505                 logger.debug("[" + myProcessor.getMyAgent().getName() + "] " + msg.getSender().name + " tell me: "
                           + msg.getPerformative() + " " + msg.getContent());
506             }
507             class INFORM_Method implements ReceiveStateMethod {
508                 public String run(CProcessor myProcessor, ACLMessage messageReceived) {
509                     doInform(myProcessor, messageReceived);
510                     // There are so many answers as Agents started
511                     if (timesSecondWait < ((GodAgent) myProcessor.getMyAgent()).myLocalData.getStartedAgents().
                               size()) {
512                         timesSecondWait++;
513                         return "SECOND_WAIT_REQUEST_INITIATOR";
514                     } else {
515                         return "FINAL_REQUEST_INITIATOR";
516                     }
517                 }
518             }
519
520             /**
521              * Method to execute when the initiator ends the conversation
522              *
523              * @param myProcessor
524              *            the CProcessor managing the conversation
525              * @param messageToSend
526              *            final message
527              */
528             protected void doFinal(CProcessor myProcessor, ACLMessage messageToSend) {
529                 messageToSend = myProcessor.getLastSentMessage();
530
531                 CAgent cAgent = myProcessor.getMyAgent();
532                 // Time to deregister the roles .
533                 deregisterRoles(cAgent);
534
535                 // After deregistering them, it is time to
536                 // deregister units.
537                 deregisterUnits(cAgent);
538
539                 // It remains only to leave THOMAS
540                 leaveTHOMAS(cAgent);
541
542                 // The last step is call to his ShutdownAgent method.
543                 // NO USER CODE BEYOND THIS POINT!!
544                 myProcessor.ShutdownAgent();
545             }
546
547             class FINAL_Method implements FinalStateMethod {
548                 public void run(CProcessor myProcessor, ACLMessage messageToSend) {
549                     doFinal(myProcessor, messageToSend);
550                 }
551             }
552
553             /**
554              * Creates a new initiator fipa request cfactory
555              *
556              * @param name
557              *            factory's name
558              * @param filter
559              *            message filter
560              * @param requestMessage
561              *            first message to send
562              * @param availableConversations
563              *            maximum number of conversation this CFactory can
564              *            manage simultaneously
565              * @param myAgent
566              *            agent owner of this CFactory
567              * @param timeout
568              *            for waiting after sending the request message
569              * @return a new fipa request initiator factory
570              */
```

```
571             public CFactory newFactory(String name, MessageFilter filter, ACLMessage requestMessage, int
                     availableConversations, CAgent myAgent, long timeout) {
572
573                 // Create factory
574                 if (filter == null) {
575                     filter = new MessageFilter("performative = REQUEST");
576                 }
577                 CFactory theFactory = new CFactory(name, filter, availableConversations, myAgent);
578
579                 // Processor template setup
580                 CProcessor processor = theFactory.cProcessorTemplate();
581
582                 // BEGIN State
583
584                 BeginState BEGIN = (BeginState) processor.getState("BEGIN");
585                 BEGIN.setMethod(new BEGIN_Method());
586
587                 // REQUEST State
588
589                 SendState REQUEST = new SendState("REQUEST_REQUEST_INITIATOR");
590
591                 REQUEST.setMethod(new REQUEST_Method());
592                 REQUEST.setMessageTemplate(requestMessage);
593                 processor.registerState(REQUEST);
594                 processor.addTransition("BEGIN", "REQUEST_REQUEST_INITIATOR");
595                 processor.addTransition("BEGIN", "FINAL_REQUEST_INITIATOR");
596
597                 // FIRST_WAIT State
598
599                 processor.registerState(new WaitState("FIRST_WAIT_REQUEST_INITIATOR", timeout));
600                 processor.addTransition("REQUEST_REQUEST_INITIATOR", "FIRST_WAIT_REQUEST_INITIATOR");
601
602                 // NOT_UNDERSTOOD State
603
604                 ReceiveState NOT_UNDERSTOOD = new ReceiveState("NOT_UNDERSTOOD_REQUEST_INITIATOR");
605                 NOT_UNDERSTOOD.setMethod(new NOT_UNDERSTOOD_Method());
606                 filter = new MessageFilter("performative = " + ACLMessage.getPerformative(ACLMessage.
                         NOT_UNDERSTOOD));
607                 NOT_UNDERSTOOD.setAcceptFilter(filter);
608                 processor.registerState(NOT_UNDERSTOOD);
609                 processor.addTransition("FIRST_WAIT_REQUEST_INITIATOR", "NOT_UNDERSTOOD_REQUEST_INITIATOR");
610
611                 // REFUSE State
612
613                 ReceiveState REFUSE = new ReceiveState("REFUSE_REQUEST_INITIATOR");
614                 REFUSE.setMethod(new REFUSE_Method());
615                 filter = new MessageFilter("performative = REFUSE");
616                 REFUSE.setAcceptFilter(filter);
617                 processor.registerState(REFUSE);
618                 processor.addTransition("FIRST_WAIT_REQUEST_INITIATOR", "REFUSE_REQUEST_INITIATOR");
619
620                 // AGREE State
621
622                 ReceiveState AGREE = new ReceiveState("AGREE_REQUEST_INITIATOR");
623                 AGREE.setMethod(new AGREE_Method());
624                 filter = new MessageFilter("performative = AGREE");
625                 AGREE.setAcceptFilter(filter);
626                 processor.registerState(AGREE);
627                 processor.addTransition("FIRST_WAIT_REQUEST_INITIATOR", "AGREE_REQUEST_INITIATOR");
628                 processor.addTransition("AGREE_REQUEST_INITIATOR", "FIRST_WAIT_REQUEST_INITIATOR");
629
630                 // SECOND_WAIT State
631
632                 processor.registerState(new WaitState("SECOND_WAIT_REQUEST_INITIATOR", timeout));
633                 processor.addTransition("AGREE_REQUEST_INITIATOR", "SECOND_WAIT_REQUEST_INITIATOR");
634
635                 // FAILURE State
636
637                 ReceiveState FAILURE = new ReceiveState("FAILURE_REQUEST_INITIATOR");
```

```
638                 FAILURE.setMethod(new FAILURE_Method());
639                 filter = new MessageFilter("performative = FAILURE");
640                 FAILURE.setAcceptFilter(filter);
641                 processor.registerState(FAILURE);
642                 processor.addTransition("SECOND_WAIT_REQUEST_INITIATOR", "FAILURE_REQUEST_INITIATOR");
643                 processor.addTransition("FAILURE_REQUEST_INITIATOR", "SECOND_WAIT_REQUEST_INITIATOR");
644
645                 // INFORM State
646
647                 ReceiveState INFORM = new ReceiveState("INFORM_REQUEST_INITIATOR");
648                 INFORM.setMethod(new INFORM_Method());
649                 filter = new MessageFilter("performative = INFORM");
650                 INFORM.setAcceptFilter(filter);
651                 processor.registerState(INFORM);
652                 processor.addTransition("SECOND_WAIT_REQUEST_INITIATOR", "INFORM_REQUEST_INITIATOR");
653                 processor.addTransition("INFORM_REQUEST_INITIATOR", "SECOND_WAIT_REQUEST_INITIATOR");
654
655                 // FINAL State
656
657                 FinalState FINAL = new FinalState("FINAL_REQUEST_INITIATOR");
658
659                 FINAL.setMethod(new FINAL_Method());
660
661                 processor.registerState(FINAL);
662                 processor.addTransition("INFORM_REQUEST_INITIATOR", "FINAL_REQUEST_INITIATOR");
663                 processor.addTransition("FAILURE_REQUEST_INITIATOR", "FINAL_REQUEST_INITIATOR");
664                 processor.addTransition("NOT_UNDERSTOOD_REQUEST_INITIATOR", "FINAL_REQUEST_INITIATOR");
665                 processor.addTransition("REFUSE_REQUEST_INITIATOR", "FINAL_REQUEST_INITIATOR");
666                 return theFactory;
667             }
668
669         } // End of class ShutdownAgentsFIPA_REQUEST
670
671         MessageFilter shutdownFilter = new MessageFilter("performative = " + ACLMessage.getPerformative(ACLMessage
                .REQUEST) + " AND shutdown = ShutdownAgent");
672
673         // Call Shutdown actions for agents contained at startedAgents
674         // ArrayList
675         ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
676         for (CAgent agentToStop : myLocalData.getStartedAgents()) {
677             msg.addReceiver(agentToStop.getAid());
678         }
679         msg.setContent("Time to shutdown.");
680         msg.setHeader("shutdown", "ShutdownAgent");
681         CFactory shutdownTalk = new ShutdownAgentsFIPA_REQUEST().newFactory("ShutdownTalk", shutdownFilter, msg,
                1, firstProcessor.getMyAgent(), 0);
682
683         // ////////////////////////////////////////////////////////////////////////////
684         // The template processor is ready. We activate the factory.
685
686         this.addFactoryAsInitiator(shutdownTalk);
687
688         try {
689             Thread.sleep(//TODO Time to sleep before start this conversation E.G. 120 * 1000);
690         } catch (InterruptedException iE) {
691             logger.error("[" + cAgent.getName() + "] Application ended abnormally. Please review your code, your
                    configuration and your database.\n" + iE.getMessage());
692             // Ends the application abnormally
693             System.exit(1);
694
695         }
696         // Finally starts the conversation.
697         this.startSyncConversation(shutdownTalk.getName());
698
699
700     } // End execution()
701
702         // This method is executed just before the agent ends its execution
703         @Override
```

```
704         protected void finalize(CProcessor firstProcessor,
705                         ACLMessage finalizeMessage) {
706                 CAgent cAgent = firstProcessor.getMyAgent();
707                 logger.info("[" + cAgent.getName() + "] finalize method executed.");
708         } // End finalize()
709
710         /**
711          * This method makes the Agent deregister all the created Roles.
712          *
713          * @param cAgent
714          */
715         private void deregisterRoles(CAgent cAgent) {
716                 // Deregister the roles I have created
717                 ArrayList<UnitRolePair> agentCreatedRoles = myLocalData
718                                 .getCreatedRolesInReverseOrder();
719                 for (UnitRolePair createdRole : agentCreatedRoles) {
720                         try {
721                                 logger.debug("["
722                                                 + cAgent.getName()
723                                                 + "] deregistering role \""
724                                                 + createdRole.getRoleID()
725                                                 + "\" at \""
726                                                 + createdRole.getUnitID()
727                                                 + "\": "
728                                                 + omsProxy.deregisterRole(createdRole.getRoleID(),
729                                                                 createdRole.getUnitID()));
730                         } catch (Exception e) {
731                                 logger.error("[" + cAgent.getName()
732                                                 + "] Exception in DeregisterRole method: "
733                                                 + e.getMessage());
734                         }
735                 }
736         }
737
738         /**
739          * This method makes the Agent deregister all the created Units.
740          *
741          * @param cAgent
742          */
743         private void deregisterUnits(CAgent cAgent) {
744                 // Requesting for the list of created unit by the agent in order
745                 // to deregister them.
746                 for (String unit : myLocalData.getCreatedUnitsInReverseOrder()) {
747                         try {
748                                 logger.debug("[" + cAgent.getName() + "] deregistering unit \""
749                                                 + unit + "\": " + omsProxy.deregisterUnit(unit));
750                         } catch (Exception e) {
751                                 logger.error("[" + cAgent.getName()
752                                                 + "] Exception in DeregisterUnit method: "
753                                                 + e.getMessage());
754                         }
755                 }
756         }
757
758         /**
759          * This method makes the Agent leaves THOMAS.
760          *
761          * @param cAgent
762          */
763         private void leaveTHOMAS(CAgent cAgent) {
764                 try {
765                         logger.debug("[" + cAgent.getName() + "] Exiting from THOMAS: "
766                                         + omsProxy.leaveRole("participant", "virtual"));
767                 } catch (Exception ex) {
768                         logger.error("[" + cAgent.getName()
769                                         + "]  Exception in LeaveRole method: " + ex.getMessage());
770                 }
771         } // End of method leaveTHOMAS ()
772
```

```
773 } //End class GodAgent
```

APPENDIX

C

AdditionAgent.java file

```java
/**
 * This class has been generated using Gormas2Magentix tool.
 *
 * @author Mario Rodrigo - mrodrigo@dsic.upv.es
 *
 */
package EMFGormas_Example;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Hashtable;
import java.util.Iterator;

import EMFGormas_Example.Utils.LocalData;
import EMFGormas_Example.Utils.UnitRolePair;

import es.upv.dsic.gti_ia.cAgents.BeginState;
import es.upv.dsic.gti_ia.cAgents.BeginStateMethod;
import es.upv.dsic.gti_ia.cAgents.CAgent;
import es.upv.dsic.gti_ia.cAgents.CFactory;
import es.upv.dsic.gti_ia.cAgents.CProcessor;
import es.upv.dsic.gti_ia.cAgents.FinalState;
import es.upv.dsic.gti_ia.cAgents.FinalStateMethod;
import es.upv.dsic.gti_ia.cAgents.SendState;
import es.upv.dsic.gti_ia.cAgents.SendStateMethod;
import es.upv.dsic.gti_ia.cAgents.protocols.FIPA_REQUEST_Initiator;
import es.upv.dsic.gti_ia.cAgents.protocols.FIPA_REQUEST_Participant;
import es.upv.dsic.gti_ia.core.ACLMessage;
import es.upv.dsic.gti_ia.core.AgentID;
import es.upv.dsic.gti_ia.core.MessageFilter;
import es.upv.dsic.gti_ia.organization.OMSProxy;
import es.upv.dsic.gti_ia.organization.Oracle;
import es.upv.dsic.gti_ia.organization.Provider;
import es.upv.dsic.gti_ia.organization.SFProxy;
import es.upv.dsic.gti_ia.organization.THOMASException;

public class AdditionAgent extends CAgent {

        // ----------------------------------------
        // FIELDS of the class
        // ----------------------------------------
        private OMSProxy omsProxy;
        private SFProxy sfProxy;
```

```
46          private ArrayList<String> results;
47          private Oracle oracle;
48          private String serviceName;
49          private LocalData myLocalData;
50
51          // ----------------------------------------
52          // CONSTRUCTOR of the class
53          // ----------------------------------------
54          public AdditionAgent(AgentID aid) throws Exception {
55                  super(aid);
56
57                  if (results == null) {
58                          results = new ArrayList<String>();
59                  }
60                  if (omsProxy == null) {
61                          omsProxy = new OMSProxy(this);
62                  }
63
64                  if (sfProxy == null) {
65                          sfProxy = new SFProxy(this);
66                  }
67
68                  if (myLocalData == null) {
69                          myLocalData = new LocalData();
70                  }
71
72          } // End of constructor
73
74          // ----------------------------------------
75          // METHODS of the class
76          // ----------------------------------------
77
78          // This is the main method of the agent
79          @Override
80          protected void execution(CProcessor firstProcessor,
81                          ACLMessage welcomeMessage) {
82                  omsProxy = new OMSProxy(this);
83                  sfProxy = new SFProxy(this);
84                  String result;
85                  ArrayList<String> resultArrayList;
86                  UnitRolePair desiredRole;
87                  ACLMessage msg;
88                  ArrayList<String> searchInputs = new ArrayList<String>();
89                  ArrayList<String> searchOutputs = new ArrayList<String>();
90                  ArrayList<String> searchKeywords = new ArrayList<String>();
91                  ArrayList<ArrayList<String>> foundServices = new ArrayList<ArrayList<String>>();
92
93                  // Requesting for the list of roles and units in which the agent is in a
94                  // specific moment.
95                  CAgent cAgent = firstProcessor.getMyAgent();
96
97                  try {
98                          //Get the list of roles that agent is playing
99                          ArrayList<UnitRolePair> agentPlayingRoles = Utils
100                                         .queryOMSForRolesPlayedByAnAgent(cAgent, omsProxy, logger);
101                         for (UnitRolePair unitRolePair : agentPlayingRoles) {
102                                 myLocalData.addPlayingRole(unitRolePair);
103                         }
104
105                         desiredRole = new UnitRolePair("operation", "Calculator");
106
107                         if (!myLocalData.getPlayingRoles().contains(desiredRole)) {
108                                 // Acquire Role in main Organization
109                                 result = omsProxy.acquireRole("operation", "Calculator");
110
111                                 if (result.contains("acquired")) {
112                                         // Add the pair Unit - Role to playingRole ArrayList
113                                         myLocalData.addPlayingRole(desiredRole);
114                                 }
```

```
115                                     logger.debug("[" + getName() + "] Entering in 'Calculator': "
116                                                     + result + "\n");
117
118                             } else {
119                                     logger.debug("[" + getName() + "] is inside in 'Calculator'.");
120                             }
121
122                     } catch (Exception e) {
123                             logger.error("[" + getName() + "]" + e.getMessage());
124                     }
125
126                     // The agent creates the CFactory that manages every message which its
127                     // performative is set to REQUEST and filter is set to shutdown.
128
129                     // The agent creates the CFactory that manages every message which its
130                     // performative is set to REQUEST and filter is set to shutdown.
131
132                     // We create a factory in order to manage ShutdownAgent orders
133                     class ShutdownAgentFIPA_REQUEST extends FIPA_REQUEST_Participant {
134
135                             @Override
136                             protected String doAction(CProcessor myProcessor) {
137                                     CAgent cAgent = myProcessor.getMyAgent();
138                                     removePublishedServices(cAgent);
139                                     leaveRoles(cAgent);
140                                     deregisterRoles(cAgent);
141                                     deregisterUnits(cAgent);
142                                     return "INFORM";
143                             }
144
145                             @Override
146                             protected void doInform(CProcessor myProcessor, ACLMessage response) {
147                                     CAgent cAgent = myProcessor.getMyAgent();
148                                     response.setSender(cAgent.getAid());
149                                     response.setReceiver(myProcessor.getLastReceivedMessage()
150                                                     .getSender());
151                                     response.setHeader("shutdown", "ShutdownAgent");
152                                     response.setContent("All my published services have been removed and all played
                                            roles have been left.");
153                                     logger.debug("["
154                                                     + myProcessor.getMyAgent().getName()
155                                                     + "] All my published services have been removed and all played
                                                            roles have been left.");
156                                     myProcessor.ShutdownAgent();
157                             }
158
159                             @Override
160                             protected String doReceiveRequest(CProcessor myProcessor,
161                                             ACLMessage request) {
162                                     return "AGREE";
163                             }
164
165                             @Override
166                             protected void doAgree(CProcessor myProcessor,
167                                             ACLMessage messageToSend) {
168                                     messageToSend.setPerformative(ACLMessage.AGREE);
169                                     messageToSend.setHeader("shutdown", "ShutdownAgent");
170                                     messageToSend.setSender(myProcessor.getMyAgent().getAid());
171                                     messageToSend.setReceiver(myProcessor.getLastReceivedMessage()
172                                                     .getSender());
173                                     messageToSend.setHeader("shutdown", "ShutdownAgent");
174                                     messageToSend
175                                                     .setContent("Received \"Shutdown\" order. Removing all my
                                                            published services and leaving all played roles.");
176                                     logger.debug("["
177                                                     + myProcessor.getMyAgent().getName()
178                                                     + "] Received \"Shutdown\" order. Removing all my published
                                                            services and leaving all played roles.");
179                             }
```

```
180                 }
181
182                 ACLMessage template;
183                 MessageFilter shutdownFilter = new MessageFilter("performative = "
184                                 + ACLMessage.getPerformative(ACLMessage.REQUEST)
185                                 + "  AND shutdown = ShutdownAgent");
186
187                 CFactory shutdownTalk = new ShutdownAgentFIPA_REQUEST().newFactory(
188                                 "ShutdownTalk", shutdownFilter, 1, firstProcessor.getMyAgent());
189                 // The template processor is ready. We activate the factory
190                 // as participant. Every message that arrives to the agent
191                 // with the performative set to REQUEST will make the factory
192                 // ShutdownTalk to create a processor in order to manage the conversation.
193                 this.addFactoryAsParticipant(shutdownTalk);
194
195                 // Register services
196
197                 registerService(/* TODO Path to ProficeDescription i.e. "http://localhost:8080/testSFservices/
                        testSFservices/owl/owls/Addition.owl"*/);
198                 // YOUR CODE FOR REGISTER SERVICES STARTS HERE
199                 //
200                 //
201                 // YOUR CODE ENDS HERE
202
203                 // Each agent's conversation is carried out by a CProcessor.
204                 // CProcessors are created by the CFactories in response
205                 // to messages that start the agent's activity in a conversation
206
207                 // An easy way to create CFactories is to create them from the
208                 // predefined factories of package es.upv.dsi.gti_ia.cAgents.protocols
209                 // Another option, not shown here, is that the agent
210                 // designs her own factory and, therefore, a new interaction protocol
211                 class AdditionFIPA_REQUEST extends FIPA_REQUEST_Participant {
212
213                         @Override
214                         protected String doAction(CProcessor myProcessor) {
215
216                                 // YOUR CODE STARTS HERE
217                                 //
218                                 // logger.debug("[" + myProcessor.getMyAgent().getName() + "] -> Nothing to do in
                                        the action. Just return the next state ");
219                                 // return "INFORM";
220                                 //
221                                 // YOUR CODE ENDS HERE
222                         }
223
224                         @Override
225                         protected void doInform(CProcessor myProcessor, ACLMessage response) {
226                                 // YOUR CODE STARTS HERE
227                                 //
228                                 // response.setContent("[" + myProcessor.getMyAgent().getName() + " ] -> Yes, my
                                        number is 666 666 666.");
229                                 // logger.info("[" + myProcessor.getMyAgent().getName() + "] I send the answer to
                                        " + myProcessor.getLastReceivedMessage().getSender().name);
230                                 //
231                                 // YOUR CODE ENDS HERE
232                         }
233
234                         @Override
235                         protected String doReceiveRequest(CProcessor myProcessor,
236                                         ACLMessage request) {
237                                 // YOUR CODE STARTS HERE
238                                 //
239                                 // logger.debug("[" + myProcessor.getMyAgent().getName() + "]-> I always accept
                                        requests.");
240                                 // return "AGREE";
241                                 //
242                                 // YOUR CODE ENDS HERE
243                         }
```

```
244
245                         @Override
246                         protected void doAgree(CProcessor myProcessor,
247                                         ACLMessage messageToSend) {
248                                 messageToSend.setPerformative(ACLMessage.AGREE);
249                                 messageToSend.setSender(myProcessor.getMyAgent().getAid());
250                                 messageToSend.setReceiver(myProcessor.getLastReceivedMessage()
251                                                 .getSender());
252                         }
253                 } // End of class AdditionFIPA_REQUEST
254
255                 // YOUR CODE FOR CREATE PARTICIPANTS MANAGERS FOR CONVERSATIONS STARTS HERE
256                 //
257                 //
258                 // YOUR CODE ENDS HERE
259
260                 // The agent creates the CFactory that manages every message which its
261                 // performative is set to REQUEST and protocol set to REQUEST. In this
262                 // example the CFactory gets the name "TALK", we don't add any
263                 // additional message acceptance criterion other than the required
264                 // by the REQUEST protocol (null) and we limit the number of
265                 // simultaneous
266                 // processors to 1, i.e. the requests will be attended one after
267                 // another.
268                 MessageFilter templateForAddition = new MessageFilter("performative = "
269                                 + ACLMessage.getPerformative(ACLMessage.REQUEST)
270                                 + " AND serviceName = Addition");
271                 CFactory talkFactoryForAddition = new AdditionFIPA_REQUEST()
272                                 .newFactory("AdditionTalk", templateForAddition, 0,
273                                                 firstProcessor.getMyAgent());
274
275                 // Finally the factory is setup to answer to incoming messages that
276                 // can start the participation of the agent in a new conversation
277                 this.addFactoryAsParticipant(talkFactoryForAddition);
278
279                 // YOUR CODE FOR MANAGE CONVERSATIONS STARTS HERE
280                 //
281                 //
282                 // YOUR CODE ENDS HERE
283
284                 // Each agent's conversation is carried out by a CProcessor.
285                 // CProcessors are created by the CFactories in response
286                 // to messages that start the agent's activity in a conversation
287
288                 // An easy way to create CFactories is to create them from the
289                 // predefined factories of package es.upv.dsi.gri_ia.cAgents.protocols
290                 // YOUR CODE FOR CREATE INITIATORS MANAGERS FOR CONVERSATIONS STARTS HERE
291                 //
292                 //
293                 // YOUR CODE ENDS HERE
294
295                 // In order to start a conversation the agent creates a message
296                 // that can be accepted by one of its initiator factories.
297
298         } // End of method execution()
299
300         // This method is executed just before the agent ends its execution
301         @Override
302         protected void finalize(CProcessor firstProcessor,
303                         ACLMessage finalizeMessage) {
304                 CAgent cAgent = firstProcessor.getMyAgent();
305                 logger.info("[" + cAgent.getName() + "] finalize method executed.");
306
307         } // End of method finalize()
308
309         /**
310          * Method to remove all the services that this Agent provides and has
311          * registered in SF.
312          *
```

```
313             * @param cAgent
314             */
315            private void removePublishedServices(CAgent cAgent) {
316                    for (String service : myLocalData.getRegisteredServices()) {
317                            deregisterService(service);
318                    }
319
320            } // End of method removePublishedServices ()
321
322            /**
323             * This method makes the Agent leave all the acquired Roles.
324             *
325             * @param cAgent
326             */
327            private void leaveRoles(CAgent cAgent) {
328                    // Requesting for the list of roles and units in which the agent is in a
329                    // specific moment to leave them.
330
331                    ArrayList<UnitRolePair> agentPlayingRoles = myLocalData
332                                    .getPlayingRolesInReverseOrder();
333                    for (UnitRolePair playingRole : agentPlayingRoles) {
334                            try {
335                                    logger.debug("["
336                                                    + cAgent.getName()
337                                                    + "] leaving role \""
338                                                    + playingRole.getRoleID()
339                                                    + "\" at \""
340                                                    + playingRole.getUnitID()
341                                                    + "\": "
342                                                    + omsProxy.leaveRole(playingRole.getRoleID(),
343                                                                    playingRole.getUnitID()));
344                            } catch (Exception e) {
345                                    logger.error("[" + cAgent.getName()
346                                                    + "] Exception in LeaveRole method: " + e.getMessage());
347                            }
348                    }
349            }
350
351            /**
352             * This method makes the Agent deregister all the created Roles.
353             *
354             * @param cAgent
355             */
356            private void deregisterRoles(CAgent cAgent) {
357                    // Deregister the roles I have created
358                    ArrayList<UnitRolePair> agentCreatedRoles = myLocalData
359                                    .getCreatedRolesInReverseOrder();
360                    for (UnitRolePair createdRole : agentCreatedRoles) {
361                            try {
362                                    logger.debug("["
363                                                    + cAgent.getName()
364                                                    + "] deregistering role \""
365                                                    + createdRole.getRoleID()
366                                                    + "\" at \""
367                                                    + createdRole.getUnitID()
368                                                    + "\": "
369                                                    + omsProxy.deregisterRole(createdRole.getRoleID(),
370                                                                    createdRole.getUnitID()));
371                            } catch (Exception e) {
372                                    logger.error("[" + cAgent.getName()
373                                                    + "] Exception in DeregisterRole method: "
374                                                    + e.getMessage());
375                            }
376                    }
377            }
378
379            /**
380             * This method makes the Agent deregister all the created Units.
381             *
```

```
382             * @param cAgent
383             */
384            private void deregisterUnits(CAgent cAgent) {
385                    // Requesting for the list of created unit by the agent in order
386                    // to deregister them.
387                    for (String unit : myLocalData.getCreatedUnitsInReverseOrder()) {
388                            try {
389                                    logger.debug("[" + cAgent.getName() + "] deregistering unit \""
390                                                    + unit + "\": " + omsProxy.deregisterUnit(unit));
391                            } catch (Exception e) {
392                                    logger.error("[" + cAgent.getName()
393                                                    + "] Exception in DeregisterUnit method: "
394                                                    + e.getMessage());
395                            }
396                    }
397            }
398
399            /**
400             * Method to register the service on SF.
401             *
402             * @param serviceName String with the name used to call the user's service.
403             * @param profile ProfileDescription of the user's service.
404             */
405            private void registerService(String serviceName) {
406                    try {
407                            ArrayList<String> resultRegister = sfProxy
408                                            .registerService(serviceName);
409                            Iterator<String> iterRes = resultRegister.iterator();
410                            String registerRes = "";
411                            while (iterRes.hasNext()) {
412                                    registerRes += iterRes.next() + "\n";
413                            }
414                            logger.debug("[" + this.getName() + "] Result registerService: "
415                                            + registerRes);
416
417                            String[] parts = resultRegister.get(0).split(": ");
418                            String serviceProfile = parts[1].trim();
419
420                            myLocalData.addRegisteredServices(serviceProfile);
421
422                    } catch (THOMASException e) {
423                            logger.error("[" + getName() + "]" + e.getMessage());
424                    }
425            }
426
427            /**
428             * Deregistering the service from SF
429             *
430             * @param serviceName
431             */
432            private void deregisterService(String service) {
433                    try {
434                            String serviceToRemove = service.substring(
435                                            service.lastIndexOf("/") + 1, service.indexOf("."));
436                            logger.debug("[" + getName() + "] deregistering service \""
437                                            + serviceToRemove + "\".");
438
439                            sfProxy.deregisterService(service);
440
441                    } catch (Exception ex) {
442                            logger.error("[" + getName()
443                                            + "] Exception in RemoveProvider method: "
444                                            + ex.getMessage());
445                    }
446
447            }// end of method DeregisterService(String service)
448
449    } // End of class  AdditionAgent
```

APPENDIX

D

ProductAgent.java file

```java
/**
 * This class has been generated using Gormas2Magentix tool.
 *
 * @author Mario Rodrigo - mrodrigo@dsic.upv.es
 *
 */
package EMFGormas_Example;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Hashtable;
import java.util.Iterator;

import EMFGormas_Example.Utils.LocalData;
import EMFGormas_Example.Utils.UnitRolePair;

import es.upv.dsic.gti_ia.cAgents.BeginState;
import es.upv.dsic.gti_ia.cAgents.BeginStateMethod;
import es.upv.dsic.gti_ia.cAgents.CAgent;
import es.upv.dsic.gti_ia.cAgents.CFactory;
import es.upv.dsic.gti_ia.cAgents.CProcessor;
import es.upv.dsic.gti_ia.cAgents.FinalState;
import es.upv.dsic.gti_ia.cAgents.FinalStateMethod;
import es.upv.dsic.gti_ia.cAgents.SendState;
import es.upv.dsic.gti_ia.cAgents.SendStateMethod;
import es.upv.dsic.gti_ia.cAgents.protocols.FIPA_REQUEST_Initiator;
import es.upv.dsic.gti_ia.cAgents.protocols.FIPA_REQUEST_Participant;
import es.upv.dsic.gti_ia.core.ACLMessage;
import es.upv.dsic.gti_ia.core.AgentID;
import es.upv.dsic.gti_ia.core.MessageFilter;
import es.upv.dsic.gti_ia.organization.OMSProxy;
import es.upv.dsic.gti_ia.organization.Oracle;
import es.upv.dsic.gti_ia.organization.Provider;
import es.upv.dsic.gti_ia.organization.SFProxy;
import es.upv.dsic.gti_ia.organization.THOMASException;

public class ProductAgent extends CAgent {

        // ----------------------------------------
        // FIELDS of the class
        // ----------------------------------------
        private OMSProxy omsProxy;
        private SFProxy sfProxy;
```

```
46          private ArrayList<String> results;
47          private Oracle oracle;
48          private String serviceName;
49          private LocalData myLocalData;
50
51          // ----------------------------------------
52          // CONSTRUCTOR of the class
53          // ----------------------------------------
54          public ProductAgent(AgentID aid) throws Exception {
55                  super(aid);
56
57                  if (results == null) {
58                          results = new ArrayList<String>();
59                  }
60                  if (omsProxy == null) {
61                          omsProxy = new OMSProxy(this);
62                  }
63
64                  if (sfProxy == null) {
65                          sfProxy = new SFProxy(this);
66                  }
67
68                  if (myLocalData == null) {
69                          myLocalData = new LocalData();
70                  }
71
72          } // End of constructor
73
74          // ----------------------------------------
75          // METHODS of the class
76          // ----------------------------------------
77
78          // This is the main method of the agent
79          @Override
80          protected void execution(CProcessor firstProcessor,
81                          ACLMessage welcomeMessage) {
82                  omsProxy = new OMSProxy(this);
83                  sfProxy = new SFProxy(this);
84                  String result;
85                  ArrayList<String> resultArrayList;
86                  UnitRolePair desiredRole;
87                  ACLMessage msg;
88                  ArrayList<String> searchInputs = new ArrayList<String>();
89                  ArrayList<String> searchOutputs = new ArrayList<String>();
90                  ArrayList<String> searchKeywords = new ArrayList<String>();
91                  ArrayList<ArrayList<String>> foundServices = new ArrayList<ArrayList<String>>();
92
93                  // Requesting for the list of roles and units in which the agent is in a
94                  // specific moment.
95                  CAgent cAgent = firstProcessor.getMyAgent();
96
97                  try {
98                          //Get the list of roles that agent is playing
99                          ArrayList<UnitRolePair> agentPlayingRoles = Utils
100                                         .queryOMSForRolesPlayedByAnAgent(cAgent, omsProxy, logger);
101                         for (UnitRolePair unitRolePair : agentPlayingRoles) {
102                                 myLocalData.addPlayingRole(unitRolePair);
103                         }
104
105                         desiredRole = new UnitRolePair("operation", "Calculator");
106
107                         if (!myLocalData.getPlayingRoles().contains(desiredRole)) {
108                                 // Acquire Role in main Organization
109                                 result = omsProxy.acquireRole("operation", "Calculator");
110
111                                 if (result.contains("acquired")) {
112                                         // Add the pair Unit - Role to playingRole ArrayList
113                                         myLocalData.addPlayingRole(desiredRole);
114                                 }
```

```
115                                     logger.debug("[" + getName() + "] Entering in 'Calculator': "
116                                                     + result + "\n");
117
118                             } else {
119                                     logger.debug("[" + getName() + "] is inside in 'Calculator'.");
120                             }
121
122                     } catch (Exception e) {
123                             logger.error("[" + getName() + "]" + e.getMessage());
124                     }
125
126                     // The agent creates the CFactory that manages every message which its
127                     // performative is set to REQUEST and filter is set to shutdown.
128
129                     // The agent creates the CFactory that manages every message which its
130                     // performative is set to REQUEST and filter is set to shutdown.
131
132                     // We create a factory in order to manage ShutdownAgent orders
133                     class ShutdownAgentFIPA_REQUEST extends FIPA_REQUEST_Participant {
134
135                             @Override
136                             protected String doAction(CProcessor myProcessor) {
137                                     CAgent cAgent = myProcessor.getMyAgent();
138                                     removePublishedServices(cAgent);
139                                     leaveRoles(cAgent);
140                                     deregisterRoles(cAgent);
141                                     deregisterUnits(cAgent);
142                                     return "INFORM";
143                             }
144
145                             @Override
146                             protected void doInform(CProcessor myProcessor, ACLMessage response) {
147                                     CAgent cAgent = myProcessor.getMyAgent();
148                                     response.setSender(cAgent.getAid());
149                                     response.setReceiver(myProcessor.getLastReceivedMessage()
150                                                     .getSender());
151                                     response.setHeader("shutdown", "ShutdownAgent");
152                                     response.setContent("All my published services have been removed and all played
                                             roles have been left.");
153                                     logger.debug("["
154                                                     + myProcessor.getMyAgent().getName()
155                                                     + "] All my published services have been removed and all played
                                                             roles have been left.");
156                                     myProcessor.ShutdownAgent();
157                             }
158
159                             @Override
160                             protected String doReceiveRequest(CProcessor myProcessor,
161                                             ACLMessage request) {
162                                     return "AGREE";
163                             }
164
165                             @Override
166                             protected void doAgree(CProcessor myProcessor,
167                                             ACLMessage messageToSend) {
168                                     messageToSend.setPerformative(ACLMessage.AGREE);
169                                     messageToSend.setHeader("shutdown", "ShutdownAgent");
170                                     messageToSend.setSender(myProcessor.getMyAgent().getAid());
171                                     messageToSend.setReceiver(myProcessor.getLastReceivedMessage()
172                                                     .getSender());
173                                     messageToSend.setHeader("shutdown", "ShutdownAgent");
174                                     messageToSend
175                                                     .setContent("Received \"Shutdown\" order. Removing all my
                                                             published services and leaving all played roles.");
176                                     logger.debug("["
177                                                     + myProcessor.getMyAgent().getName()
178                                                     + "] Received \"Shutdown\" order. Removing all my published
                                                             services and leaving all played roles.");
179                             }
```

```
180                 }
181
182                 ACLMessage template;
183                 MessageFilter shutdownFilter = new MessageFilter("performative = "
184                                 + ACLMessage.getPerformative(ACLMessage.REQUEST)
185                                 + "  AND shutdown = ShutdownAgent");
186
187                 CFactory shutdownTalk = new ShutdownAgentFIPA_REQUEST().newFactory(
188                                 "ShutdownTalk", shutdownFilter, 1, firstProcessor.getMyAgent());
189                 // The template processor is ready. We activate the factory
190                 // as participant. Every message that arrives to the agent
191                 // with the performative set to REQUEST will make the factory
192                 // ShutdownTalk to create a processor in order to manage the conversation.
193                 this.addFactoryAsParticipant(shutdownTalk);
194
195                 // Register services
196
197                 registerService(/* TODO Path to ProficeDescription i.e. "http://localhost:8080/testSFservices/
                        testSFservices/owl/owls/Product.owl"*/);
198
199                 registerService(/* TODO Path to ProficeDescription i.e. "http://localhost:8080/testSFservices/
                        testSFservices/owl/owls/Square.owl"*/);
200                 // YOUR CODE FOR REGISTER SERVICES STARTS HERE
201                 //
202                 //
203                 // YOUR CODE ENDS HERE
204
205                 // Each agent's conversation is carried out by a CProcessor.
206                 // CProcessors are created by the CFactories in response
207                 // to messages that start the agent's activity in a conversation
208
209                 // An easy way to create CFactories is to create them from the
210                 // predefined factories of package es.upv.dsi.gti_ia.cAgents.protocols
211                 // Another option, not shown here, is that the agent
212                 // designs her own factory and, therefore, a new interaction protocol
213                 class ProductFIPA_REQUEST extends FIPA_REQUEST_Participant {
214
215                         @Override
216                         protected String doAction(CProcessor myProcessor) {
217
218                                 // YOUR CODE STARTS HERE
219                                 //
220                                 // logger.debug("[" + myProcessor.getMyAgent().getName() + "] -> Nothing to do in
                                        the action. Just return the next state ");
221                                 // return "INFORM";
222                                 //
223                                 // YOUR CODE ENDS HERE
224                         }
225
226                         @Override
227                         protected void doInform(CProcessor myProcessor, ACLMessage response) {
228                                 // YOUR CODE STARTS HERE
229                                 //
230                                 // response.setContent("[" + myProcessor.getMyAgent().getName() + " ] -> Yes, my
                                        number is 666 666 666.");
231                                 // logger.info("[" + myProcessor.getMyAgent().getName() + "] I send the answer to
                                        " + myProcessor.getLastReceivedMessage().getSender().name);
232                                 //
233                                 // YOUR CODE ENDS HERE
234                         }
235
236                         @Override
237                         protected String doReceiveRequest(CProcessor myProcessor,
238                                         ACLMessage request) {
239                                 // YOUR CODE STARTS HERE
240                                 //
241                                 // logger.debug("[" + myProcessor.getMyAgent().getName() + "]-> I always accept
                                        requests.");
242                                 // return "AGREE";
```

```
243                                     //
244                                     // YOUR CODE ENDS HERE
245                             }
246
247                             @Override
248                             protected void doAgree(CProcessor myProcessor,
249                                             ACLMessage messageToSend) {
250                                     messageToSend.setPerformative(ACLMessage.AGREE);
251                                     messageToSend.setSender(myProcessor.getMyAgent().getAid());
252                                     messageToSend.setReceiver(myProcessor.getLastReceivedMessage()
253                                                     .getSender());
254                             }
255                     } // End of class ProductFIPA_REQUEST
256
257                     class SquareFIPA_REQUEST extends FIPA_REQUEST_Participant {
258
259                             @Override
260                             protected String doAction(CProcessor myProcessor) {
261
262                                     // YOUR CODE STARTS HERE
263                                     //
264                                     // logger.debug("[" + myProcessor.getMyAgent().getName() + "] -> Nothing to do in
                                                the action. Just return the next state ");
265                                     // return "INFORM";
266                                     //
267                                     // YOUR CODE ENDS HERE
268                             }
269
270                             @Override
271                             protected void doInform(CProcessor myProcessor, ACLMessage response) {
272                                     // YOUR CODE STARTS HERE
273                                     //
274                                     // response.setContent("[" + myProcessor.getMyAgent().getName() + " ] -> Yes, my
                                                number is 666 666 666.");
275                                     // logger.info("[" + myProcessor.getMyAgent().getName() + "] I send the answer to
                                                " + myProcessor.getLastReceivedMessage().getSender().name);
276                                     //
277                                     // YOUR CODE ENDS HERE
278                             }
279
280                             @Override
281                             protected String doReceiveRequest(CProcessor myProcessor,
282                                             ACLMessage request) {
283                                     // YOUR CODE STARTS HERE
284                                     //
285                                     // logger.debug("[" + myProcessor.getMyAgent().getName() + "]-> I always accept
                                                requests.");
286                                     // return "AGREE";
287                                     //
288                                     // YOUR CODE ENDS HERE
289                             }
290
291                             @Override
292                             protected void doAgree(CProcessor myProcessor,
293                                             ACLMessage messageToSend) {
294                                     messageToSend.setPerformative(ACLMessage.AGREE);
295                                     messageToSend.setSender(myProcessor.getMyAgent().getAid());
296                                     messageToSend.setReceiver(myProcessor.getLastReceivedMessage()
297                                                     .getSender());
298                             }
299                     } // End of class SquareFIPA_REQUEST
300
301                     // YOUR CODE FOR CREATE PARTICIPANTS MANAGERS FOR CONVERSATIONS STARTS HERE
302                     //
303                     //
304                     // YOUR CODE ENDS HERE
305
306                     // The agent creates the CFactory that manages every message which its
307                     // performative is set to REQUEST and protocol set to REQUEST. In this
```

```
308                 // example the CFactory gets the name "TALK", we don't add any
309                 // additional message acceptance criterion other than the required
310                 // by the REQUEST protocol (null) and we limit the number of
311                 // simultaneous
312                 // processors to 1, i.e. the requests will be attended one after
313                 // another.
314                 MessageFilter templateForProduct = new MessageFilter("performative = "
315                                 + ACLMessage.getPerformative(ACLMessage.REQUEST)
316                                 + " AND serviceName = Product");
317                 CFactory talkFactoryForProduct = new ProductFIPA_REQUEST().newFactory(
318                                 "ProductTalk", templateForProduct, 0,
319                                 firstProcessor.getMyAgent());
320
321                 // Finally the factory is setup to answer to incoming messages that
322                 // can start the participation of the agent in a new conversation
323                 this.addFactoryAsParticipant(talkFactoryForProduct);
324
325                 MessageFilter templateForSquare = new MessageFilter("performative = "
326                                 + ACLMessage.getPerformative(ACLMessage.REQUEST)
327                                 + " AND serviceName = Square");
328                 CFactory talkFactoryForSquare = new SquareFIPA_REQUEST()
329                                 .newFactory("SquareTalk", templateForSquare, 0,
330                                                 firstProcessor.getMyAgent());
331
332                 // Finally the factory is setup to answer to incoming messages that
333                 // can start the participation of the agent in a new conversation
334                 this.addFactoryAsParticipant(talkFactoryForSquare);
335
336                 // YOUR CODE FOR MANAGE CONVERSATIONS STARTS HERE
337                 //
338                 //
339                 // YOUR CODE ENDS HERE
340
341                 // Each agent's conversation is carried out by a CProcessor.
342                 // CProcessors are created by the CFactories in response
343                 // to messages that start the agent's activity in a conversation
344
345                 // An easy way to create CFactories is to create them from the
346                 // predefined factories of package es.upv.dsi.gri_ia.cAgents.protocols
347                 // YOUR CODE FOR CREATE INITIATORS MANAGERS FOR CONVERSATIONS STARTS HERE
348                 //
349                 //
350                 // YOUR CODE ENDS HERE
351
352                 // In order to start a conversation the agent creates a message
353                 // that can be accepted by one of its initiator factories.
354
355         } // End of method execution()
356
357         // This method is executed just before the agent ends its execution
358         @Override
359         protected void finalize(CProcessor firstProcessor,
360                         ACLMessage finalizeMessage) {
361                 CAgent cAgent = firstProcessor.getMyAgent();
362                 logger.info("[" + cAgent.getName() + "] finalize method executed.");
363
364         } // End of method finalize()
365
366         /**
367          * Method to remove all the services that this Agent provides and has
368          * registered in SF.
369          *
370          * @param cAgent
371          */
372         private void removePublishedServices(CAgent cAgent) {
373                 for (String service : myLocalData.getRegisteredServices()) {
374                         deregisterService(service);
375                 }
376
```

```
377         } // End of method removePublishedServices ()
378
379         /**
380          * This method makes the Agent leave all the acquired Roles.
381          *
382          * @param cAgent
383          */
384         private void leaveRoles(CAgent cAgent) {
385                 // Requesting for the list of roles and units in which the agent is in a
386                 // specific moment to leave them.
387
388                 ArrayList<UnitRolePair> agentPlayingRoles = myLocalData
389                                 .getPlayingRolesInReverseOrder();
390                 for (UnitRolePair playingRole : agentPlayingRoles) {
391                         try {
392                                 logger.debug("["
393                                                 + cAgent.getName()
394                                                 + "] leaving role \""
395                                                 + playingRole.getRoleID()
396                                                 + "\" at \""
397                                                 + playingRole.getUnitID()
398                                                 + "\": "
399                                                 + omsProxy.leaveRole(playingRole.getRoleID(),
400                                                                 playingRole.getUnitID()));
401                         } catch (Exception e) {
402                                 logger.error("[" + cAgent.getName()
403                                                 + "] Exception in LeaveRole method: " + e.getMessage());
404                         }
405                 }
406         }
407
408         /**
409          * This method makes the Agent deregister all the created Roles.
410          *
411          * @param cAgent
412          */
413         private void deregisterRoles(CAgent cAgent) {
414                 // Deregister the roles I have created
415                 ArrayList<UnitRolePair> agentCreatedRoles = myLocalData
416                                 .getCreatedRolesInReverseOrder();
417                 for (UnitRolePair createdRole : agentCreatedRoles) {
418                         try {
419                                 logger.debug("["
420                                                 + cAgent.getName()
421                                                 + "] deregistering role \""
422                                                 + createdRole.getRoleID()
423                                                 + "\" at \""
424                                                 + createdRole.getUnitID()
425                                                 + "\": "
426                                                 + omsProxy.deregisterRole(createdRole.getRoleID(),
427                                                                 createdRole.getUnitID()));
428                         } catch (Exception e) {
429                                 logger.error("[" + cAgent.getName()
430                                                 + "] Exception in DeregisterRole method: "
431                                                 + e.getMessage());
432                         }
433                 }
434         }
435
436         /**
437          * This method makes the Agent deregister all the created Units.
438          *
439          * @param cAgent
440          */
441         private void deregisterUnits(CAgent cAgent) {
442                 // Requesting for the list of created unit by the agent in order
443                 // to deregister them.
444                 for (String unit : myLocalData.getCreatedUnitsInReverseOrder()) {
445                         try {
```

```
446                                 logger.debug("[" + cAgent.getName() + "] deregistering unit \""
447                                                 + unit + "\": " + omsProxy.deregisterUnit(unit));
448                         } catch (Exception e) {
449                                 logger.error("[" + cAgent.getName()
450                                                 + "] Exception in DeregisterUnit method: "
451                                                 + e.getMessage());
452                         }
453                 }
454         }
455
456         /**
457          * Method to register the service on SF.
458          *
459          * @param serviceName String with the name used to call the user's service.
460          * @param profile ProfileDescription of the user's service.
461          */
462         private void registerService(String serviceName) {
463                 try {
464                         ArrayList<String> resultRegister = sfProxy
465                                         .registerService(serviceName);
466                         Iterator<String> iterRes = resultRegister.iterator();
467                         String registerRes = "";
468                         while (iterRes.hasNext()) {
469                                 registerRes += iterRes.next() + "\n";
470                         }
471                         logger.debug("[" + this.getName() + "] Result registerService: "
472                                         + registerRes);
473
474                         String[] parts = resultRegister.get(0).split(": ");
475                         String serviceProfile = parts[1].trim();
476
477                         myLocalData.addRegisteredServices(serviceProfile);
478
479                 } catch (THOMASException e) {
480                         logger.error("[" + getName() + "]" + e.getMessage());
481                 }
482         }
483
484         /**
485          * Deregistering the service from SF
486          *
487          * @param serviceName
488          */
489         private void deregisterService(String service) {
490                 try {
491                         String serviceToRemove = service.substring(
492                                         service.lastIndexOf("/") + 1, service.indexOf("."));
493                         logger.debug("[" + getName() + "] deregistering service \""
494                                         + serviceToRemove + "\".");
495
496                         sfProxy.deregisterService(service);
497
498                 } catch (Exception ex) {
499                         logger.error("[" + getName()
500                                         + "] Exception in RemoveProvider method: "
501                                         + ex.getMessage());
502                 }
503
504         }// end of method DeregisterService(String service)
505
506 } // End of class  ProductAgent
```

APPENDIX

E

JamesAgent.java file

```java
/**
 * This class has been generated using Gormas2Magentix tool.
 *
 * @author Mario Rodrigo - mrodrigo@dsic.upv.es
 *
 */
package EMFGormas_Example;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Hashtable;
import java.util.Iterator;

import EMFGormas_Example.Utils.LocalData;
import EMFGormas_Example.Utils.UnitRolePair;

import es.upv.dsic.gti_ia.cAgents.BeginState;
import es.upv.dsic.gti_ia.cAgents.BeginStateMethod;
import es.upv.dsic.gti_ia.cAgents.CAgent;
import es.upv.dsic.gti_ia.cAgents.CFactory;
import es.upv.dsic.gti_ia.cAgents.CProcessor;
import es.upv.dsic.gti_ia.cAgents.FinalState;
import es.upv.dsic.gti_ia.cAgents.FinalStateMethod;
import es.upv.dsic.gti_ia.cAgents.SendState;
import es.upv.dsic.gti_ia.cAgents.SendStateMethod;
import es.upv.dsic.gti_ia.cAgents.protocols.FIPA_REQUEST_Initiator;
import es.upv.dsic.gti_ia.cAgents.protocols.FIPA_REQUEST_Participant;
import es.upv.dsic.gti_ia.core.ACLMessage;
import es.upv.dsic.gti_ia.core.AgentID;
import es.upv.dsic.gti_ia.core.MessageFilter;
import es.upv.dsic.gti_ia.organization.OMSProxy;
import es.upv.dsic.gti_ia.organization.Oracle;
import es.upv.dsic.gti_ia.organization.Provider;
import es.upv.dsic.gti_ia.organization.SFProxy;
import es.upv.dsic.gti_ia.organization.THOMASException;

public class JamesAgent extends CAgent {

        // ----------------------------------------
        // FIELDS of the class
        // ----------------------------------------
        private OMSProxy omsProxy;
        private SFProxy sfProxy;
```

```
46          private ArrayList<String> results;
47          private Oracle oracle;
48          private String serviceName;
49          private LocalData myLocalData;
50
51          // ----------------------------------------
52          // CONSTRUCTOR of the class
53          // ----------------------------------------
54          public JamesAgent(AgentID aid) throws Exception {
55                  super(aid);
56
57                  if (results == null) {
58                          results = new ArrayList<String>();
59                  }
60                  if (omsProxy == null) {
61                          omsProxy = new OMSProxy(this);
62                  }
63
64                  if (sfProxy == null) {
65                          sfProxy = new SFProxy(this);
66                  }
67
68                  if (myLocalData == null) {
69                          myLocalData = new LocalData();
70                  }
71
72          } // End of constructor
73
74          // ----------------------------------------
75          // METHODS of the class
76          // ----------------------------------------
77
78          // This is the main method of the agent
79          @Override
80          protected void execution(CProcessor firstProcessor,
81                          ACLMessage welcomeMessage) {
82                  omsProxy = new OMSProxy(this);
83                  sfProxy = new SFProxy(this);
84                  String result;
85                  ArrayList<String> resultArrayList;
86                  UnitRolePair desiredRole;
87                  ACLMessage msg;
88                  ArrayList<String> searchInputs = new ArrayList<String>();
89                  ArrayList<String> searchOutputs = new ArrayList<String>();
90                  ArrayList<String> searchKeywords = new ArrayList<String>();
91                  ArrayList<ArrayList<String>> foundServices = new ArrayList<ArrayList<String>>();
92
93                  // Requesting for the list of roles and units in which the agent is in a
94                  // specific moment.
95                  CAgent cAgent = firstProcessor.getMyAgent();
96
97                  try {
98                          //Get the list of roles that agent is playing
99                          ArrayList<UnitRolePair> agentPlayingRoles = Utils
100                                         .queryOMSForRolesPlayedByAnAgent(cAgent, omsProxy, logger);
101                         for (UnitRolePair unitRolePair : agentPlayingRoles) {
102                                 myLocalData.addPlayingRole(unitRolePair);
103                         }
104
105                         desiredRole = new UnitRolePair("student", "School");
106
107                         if (!myLocalData.getPlayingRoles().contains(desiredRole)) {
108                                 // Acquire Role in main Organization
109                                 result = omsProxy.acquireRole("student", "School");
110
111                                 if (result.contains("acquired")) {
112                                         // Add the pair Unit - Role to playingRole ArrayList
113                                         myLocalData.addPlayingRole(desiredRole);
114                                 }
```

```
115                                     logger.debug("[" + getName() + "] Entering in 'School': "
116                                                     + result + "\n");
117
118                             } else {
119                                     logger.debug("[" + getName() + "] is inside in 'School'.");
120                             }
121
122                     } catch (Exception e) {
123                             logger.error("[" + getName() + "]" + e.getMessage());
124                     }
125
126                     // The agent creates the CFactory that manages every message which its
127                     // performative is set to REQUEST and filter is set to shutdown.
128
129                     // The agent creates the CFactory that manages every message which its
130                     // performative is set to REQUEST and filter is set to shutdown.
131
132                     // We create a factory in order to manage ShutdownAgent orders
133                     class ShutdownAgentFIPA_REQUEST extends FIPA_REQUEST_Participant {
134
135                             @Override
136                             protected String doAction(CProcessor myProcessor) {
137                                     CAgent cAgent = myProcessor.getMyAgent();
138                                     removePublishedServices(cAgent);
139                                     leaveRoles(cAgent);
140                                     deregisterRoles(cAgent);
141                                     deregisterUnits(cAgent);
142                                     return "INFORM";
143                             }
144
145                             @Override
146                             protected void doInform(CProcessor myProcessor, ACLMessage response) {
147                                     CAgent cAgent = myProcessor.getMyAgent();
148                                     response.setSender(cAgent.getAid());
149                                     response.setReceiver(myProcessor.getLastReceivedMessage()
150                                                     .getSender());
151                                     response.setHeader("shutdown", "ShutdownAgent");
152                                     response.setContent("All my published services have been removed and all played
                                            roles have been left.");
153                                     logger.debug("["
154                                                     + myProcessor.getMyAgent().getName()
155                                                     + "] All my published services have been removed and all played
                                                            roles have been left.");
156                                     myProcessor.ShutdownAgent();
157                             }
158
159                             @Override
160                             protected String doReceiveRequest(CProcessor myProcessor,
161                                             ACLMessage request) {
162                                     return "AGREE";
163                             }
164
165                             @Override
166                             protected void doAgree(CProcessor myProcessor,
167                                             ACLMessage messageToSend) {
168                                     messageToSend.setPerformative(ACLMessage.AGREE);
169                                     messageToSend.setHeader("shutdown", "ShutdownAgent");
170                                     messageToSend.setSender(myProcessor.getMyAgent().getAid());
171                                     messageToSend.setReceiver(myProcessor.getLastReceivedMessage()
172                                                     .getSender());
173                                     messageToSend.setHeader("shutdown", "ShutdownAgent");
174                                     messageToSend
175                                                     .setContent("Received \"Shutdown\" order. Removing all my
                                                            published services and leaving all played roles.");
176                                     logger.debug("["
177                                                     + myProcessor.getMyAgent().getName()
178                                                     + "] Received \"Shutdown\" order. Removing all my published
                                                            services and leaving all played roles.");
179                             }
```

```
180                 }
181
182                 ACLMessage template;
183                 MessageFilter shutdownFilter = new MessageFilter("performative = "
184                                 + ACLMessage.getPerformative(ACLMessage.REQUEST)
185                                 + "  AND shutdown = ShutdownAgent");
186
187                 CFactory shutdownTalk = new ShutdownAgentFIPA_REQUEST().newFactory(
188                                 "ShutdownTalk", shutdownFilter, 1, firstProcessor.getMyAgent());
189                 // The template processor is ready. We activate the factory
190                 // as participant. Every message that arrives to the agent
191                 // with the performative set to REQUEST will make the factory
192                 // ShutdownTalk to create a processor in order to manage the conversation.
193                 this.addFactoryAsParticipant(shutdownTalk);
194
195                 // Register services
196                 // YOUR CODE FOR REGISTER SERVICES STARTS HERE
197                 //
198                 //
199                 // YOUR CODE ENDS HERE
200
201                 // Each agent's conversation is carried out by a CProcessor.
202                 // CProcessors are created by the CFactories in response
203                 // to messages that start the agent's activity in a conversation
204
205                 // An easy way to create CFactories is to create them from the
206                 // predefined factories of package es.upv.dsi.gti_ia.cAgents.protocols
207                 // Another option, not shown here, is that the agent
208                 // designs her own factory and, therefore, a new interaction protocol
209                 // YOUR CODE FOR CREATE PARTICIPANTS MANAGERS FOR CONVERSATIONS STARTS HERE
210                 //
211                 //
212                 // YOUR CODE ENDS HERE
213
214                 // The agent creates the CFactory that manages every message which its
215                 // performative is set to REQUEST and protocol set to REQUEST. In this
216                 // example the CFactory gets the name "TALK", we don't add any
217                 // additional message acceptance criterion other than the required
218                 // by the REQUEST protocol (null) and we limit the number of
219                 // simultaneous
220                 // processors to 1, i.e. the requests will be attended one after
221                 // another.
222                 // YOUR CODE FOR MANAGE CONVERSATIONS STARTS HERE
223                 //
224                 //
225                 // YOUR CODE ENDS HERE
226
227                 // Each agent's conversation is carried out by a CProcessor.
228                 // CProcessors are created by the CFactories in response
229                 // to messages that start the agent's activity in a conversation
230
231                 // An easy way to create CFactories is to create them from the
232                 // predefined factories of package es.upv.dsi.gri_ia.cAgents.protocols
233                 class SquareFIPA_REQUEST extends FIPA_REQUEST_Initiator {
234                         protected void doInform(CProcessor myProcessor, ACLMessage msg) {
235                                 // YOUR CODE STARTS HERE
236                                 //
237                                 // logger.debug("[" + myProcessor.getMyAgent().getName() + "] " + msg.getSender().
                                          name + " informs me " + msg.getContent());
238                                 //
239                                 // YOUR CODE ENDS HERE
240                         }
241
242                         protected void doRequest(CProcessor myProcessor,
243                                         ACLMessage messageToSend) {
244                                 messageToSend.setHeader("serviceName", "Square");
245                         }
246                 } // End of class SquareFIPA_REQUEST
247                 class ProductFIPA_REQUEST extends FIPA_REQUEST_Initiator {
```

```
248                         protected void doInform(CProcessor myProcessor, ACLMessage msg) {
249                                 // YOUR CODE STARTS HERE
250                                 //
251                                 // logger.debug("[" + myProcessor.getMyAgent().getName() + "] " + msg.getSender().
                                          name + " informs me " + msg.getContent());
252                                 //
253                                 // YOUR CODE ENDS HERE
254                         }
255
256                         protected void doRequest(CProcessor myProcessor,
257                                         ACLMessage messageToSend) {
258                                 messageToSend.setHeader("serviceName", "Product");
259                         }
260                 } // End of class ProductFIPA_REQUEST
261                 class AdditionFIPA_REQUEST extends FIPA_REQUEST_Initiator {
262                         protected void doInform(CProcessor myProcessor, ACLMessage msg) {
263                                 // YOUR CODE STARTS HERE
264                                 //
265                                 // logger.debug("[" + myProcessor.getMyAgent().getName() + "] " + msg.getSender().
                                          name + " informs me " + msg.getContent());
266                                 //
267                                 // YOUR CODE ENDS HERE
268                         }
269
270                         protected void doRequest(CProcessor myProcessor,
271                                         ACLMessage messageToSend) {
272                                 messageToSend.setHeader("serviceName", "Addition");
273                         }
274                 } // End of class AdditionFIPA_REQUEST
275                         // YOUR CODE FOR CREATE INITIATORS MANAGERS FOR CONVERSATIONS STARTS HERE
276                         //
277                         //
278                         // YOUR CODE ENDS HERE
279
280                 // In order to start a conversation the agent creates a message
281                 // that can be accepted by one of its initiator factories.
282                 // Search for the service "Square"
283                 serviceName = "Square";
284                 searchKeywords.clear();
285                 searchKeywords.add(serviceName);
286                 foundServices.clear();
287
288                 do {
289                         // Waiting for services
290                         try {
291                                 Thread.sleep(2 * 1000);
292
293                                 foundServices = sfProxy.searchService(searchInputs,
294                                                 searchOutputs, searchKeywords);
295
296                         } catch (InterruptedException e) {
297                                 logger.error("[" + getName() + "] " + e.getMessage());
298
299                         } catch (THOMASException e) {
300                                 logger.error("[" + getName() + "] " + e.getMessage());
301                         }
302
303                 } while (foundServices.isEmpty());
304
305                 // Request the execution of the service "Square"
306                 try {
307                         String serviceOWLS = sfProxy
308                                         .getService(foundServices.get(0).get(0));
309                         Oracle oracle = new Oracle(serviceOWLS);
310
311                         // Agents or Organizations that can execute the service for me.
312                         ArrayList<Provider> providers = oracle.getProviders();
313                         // Groundings to execute directly the service by myself.
314                         ArrayList<String> providersGroundingWSDL = oracle
```

```
315                                         .getProvidersGroundingWSDL();
316                         // Get service inputs
317                         ArrayList<String> serviceInputs = oracle.getOwlsProfileInputs();
318                         // Put the service inputs values
319                         HashMap<String, String> agentInputs = new HashMap<String, String>();
320
321                         if (!providers.isEmpty()) {
322                                 // Agents or Organizations that can execute the service for me.
323                                 // YOUR CODE STARTS HERE
324                                 //
325                                 // serviceInputs.clear();
326                                 // serviceInputs = oracle.getOwlsProfileInputs();
327                                 //
328                                 // agentInputs.clear();
329                                 // agentInputs = new HashMap<String, String>();
330                                 // for (String input : serviceInputs) {
331                                 //      if (input.equalsIgnoreCase("x")) {
332                                 //          agentInputs.put(input, "5");
333                                 //      } else if (input.equalsIgnoreCase("y")) {
334                                 //              agentInputs.put(input, resultEquation);
335                                 //      } else {
336                                 //              agentInputs.put(input, "0");
337                                 //      }
338                                 // }
339                                 //
340                                 // logger.info("[" + this.getName() + "]" + " Requesting Square Service.");
341                                 //
342                                 // msg = null;
343                                 // msg = new ACLMessage(ACLMessage.REQUEST);
344                                 // msg.setHeader("serviceName", serviceName);
345                                 // msg.setReceiver(new AgentID(providers.get(0).getEntityID()));
346                                 // msg.setProtocol("fipa-request");
347                                 // msg.setSender(getAid());
348
349                                 // content = "";
350                                 // content = st.buildServiceContent(oracle.getServiceName(), agentInputs);
351                                 // msg.setContent(content);
352
353                                 // ///this.send_request(msg);
354
355                                 // // The agent creates the CFactory that creates processors that
356                                 // // initiate REQUEST protocol conversations. In this example
357                                 // // the CFactory gets the name "TALK", we don't add any
358                                 // // additional message acceptance criterion other than the required
359                                 // // by the REQUEST protocol (null) and we do not limit the number of
360                                 // // simultaneous processors (value 0)
361                                 // CFactory talkFactoryForSquare = new SquareFIPA_REQUEST().newFactory("SquareTalk
                                        ", null, msg, 1, firstProcessor.getMyAgent(), 0);
362
363                                 // // The factory is setup to answer start conversation requests from
364                                 // // the agent using the REQUEST protocol.
365
366                                 // this.addFactoryAsInitiator(talkFactoryForSquare);
367
368                                 // // finally the new conversation starts. Because it is synchronous,
369                                 // // the current interaction halts until the new conversation ends.
370                                 // this.startSyncConversation("SquareTalk");
371
372                                 // inputs.clear();
373                                 // st.extractServiceContent(requestResult, inputs);
374                                 // resultEquation = inputs.get("Result");
375                                 //
376                                 // YOUR CODE ENDS HERE
377
378                         } else if (!providersGroundingWSDL.isEmpty()) {
379                                 // Groundings to execute the service directly by myself.
380                                 // YOUR CODE STARTS HERE
381                                 //
382                                 // serviceInputs = oracle.getOwlsProfileInputs();
```

```
383
384                                 // agentInputs.clear();
385
386                                 // agentInputs = new HashMap<String, String>();
387                                 // for (String input : serviceInputs) {
388                                 //      agentInputs.put(input, resultEquation);
389                                 // }
390
391                                 // logger.info("[" + this.getName() + "]" + " Executing Square Service.");
392
393                                 // HashMap<String, Object> resultExecution = st.executeWebService(
                                        providersGrounding.get(0), agentInputs);
394
395                                 // Double resultContent = (Double) resultExecution.get("Result");
396
397                                 // logger.info("[" + this.getName() + "] Final result: " + resultContent + ".");
398                                 //
399                                 // YOUR CODE ENDS HERE
400                         } else {
401                                 //no providers for this service
402                                 logger.info("[" + this.getName() + "]"
403                                                 + " No providers found for Square Service.");
404                         }
405                 } catch (THOMASException e) {
406                         logger.error("[" + getName() + "] " + e.getMessage());
407                 }
408                 // Search for the service "Product"
409                 serviceName = "Product";
410                 searchKeywords.clear();
411                 searchKeywords.add(serviceName);
412                 foundServices.clear();
413
414                 do {
415                         // Waiting for services
416                         try {
417                                 Thread.sleep(2 * 1000);
418
419                                 foundServices = sfProxy.searchService(searchInputs,
420                                                 searchOutputs, searchKeywords);
421
422                         } catch (InterruptedException e) {
423                                 logger.error("[" + getName() + "] " + e.getMessage());
424
425                         } catch (THOMASException e) {
426                                 logger.error("[" + getName() + "] " + e.getMessage());
427                         }
428
429                 } while (foundServices.isEmpty());
430
431                 // Request the execution of the service "Product"
432                 try {
433                         String serviceOWLS = sfProxy
434                                         .getService(foundServices.get(0).get(0));
435                         Oracle oracle = new Oracle(serviceOWLS);
436
437                         // Agents or Organizations that can execute the service for me.
438                         ArrayList<Provider> providers = oracle.getProviders();
439                         // Groundings to execute directly the service by myself.
440                         ArrayList<String> providersGroundingWSDL = oracle
441                                         .getProvidersGroundingWSDL();
442                         // Get service inputs
443                         ArrayList<String> serviceInputs = oracle.getOwlsProfileInputs();
444                         // Put the service inputs values
445                         HashMap<String, String> agentInputs = new HashMap<String, String>();
446
447                         if (!providers.isEmpty()) {
448                                 // Agents or Organizations that can execute the service for me.
449                                 // YOUR CODE STARTS HERE
450                                 //
```

```
451                                 // serviceInputs.clear();
452                                 // serviceInputs = oracle.getOwlsProfileInputs();
453                                 //
454                                 // agentInputs.clear();
455                                 // agentInputs = new HashMap<String, String>();
456                                 // for (String input : serviceInputs) {
457                                 //      if (input.equalsIgnoreCase("x")) {
458                                 //          agentInputs.put(input, "5");
459                                 //      } else if (input.equalsIgnoreCase("y")) {
460                                 //              agentInputs.put(input, resultEquation);
461                                 //      } else {
462                                 //              agentInputs.put(input, "0");
463                                 //      }
464                                 // }
465                                 //
466                                 // logger.info("[" + this.getName() + "]" + " Requesting Product Service.");
467                                 //
468                                 // msg = null;
469                                 // msg = new ACLMessage(ACLMessage.REQUEST);
470                                 // msg.setHeader("serviceName", serviceName);
471                                 // msg.setReceiver(new AgentID(providers.get(0).getEntityID()));
472                                 // msg.setProtocol("fipa-request");
473                                 // msg.setSender(getAid());
474
475                                 // content = "";
476                                 // content = st.buildServiceContent(oracle.getServiceName(), agentInputs);
477                                 // msg.setContent(content);
478
479                                 // ///this.send_request(msg);
480
481                                 // // The agent creates the CFactory that creates processors that
482                                 // // initiate REQUEST protocol conversations. In this example
483                                 // // the CFactory gets the name "TALK", we don't add any
484                                 // // additional message acceptance criterion other than the required
485                                 // // by the REQUEST protocol (null) and we do not limit the number of
486                                 // // simultaneous processors (value 0)
487                                 // CFactory talkFactoryForProduct = new ProductFIPA_REQUEST().newFactory("
                                        ProductTalk", null, msg, 1, firstProcessor.getMyAgent(), 0);
488
489                                 // // The factory is setup to answer start conversation requests from
490                                 // // the agent using the REQUEST protocol.
491
492                                 // this.addFactoryAsInitiator(talkFactoryForProduct);
493
494                                 // // finally the new conversation starts. Because it is synchronous,
495                                 // // the current interaction halts until the new conversation ends.
496                                 // this.startSyncConversation("ProductTalk");
497
498                                 // inputs.clear();
499                                 // st.extractServiceContent(requestResult, inputs);
500                                 // resultEquation = inputs.get("Result");
501                                 //
502                                 // YOUR CODE ENDS HERE
503
504                             } else if (!providersGroundingWSDL.isEmpty()) {
505                                 // Groundings to execute the service directly by myself.
506                                 // YOUR CODE STARTS HERE
507                                 //
508                                 // serviceInputs = oracle.getOwlsProfileInputs();
509
510                                 // agentInputs.clear();
511
512                                 // agentInputs = new HashMap<String, String>();
513                                 // for (String input : serviceInputs) {
514                                 //      agentInputs.put(input, resultEquation);
515                                 // }
516
517                                 // logger.info("[" + this.getName() + "]" + " Executing Product Service.");
518
```

```
519                                     // HashMap<String, Object> resultExecution = st.executeWebService(
                                                providersGrounding.get(0), agentInputs);
520
521                                     // Double resultContent = (Double) resultExecution.get("Result");
522
523                                     // logger.info("[" + this.getName() + "] Final result: " + resultContent + ".");
524                                     //
525                                     // YOUR CODE ENDS HERE
526                             } else {
527                                     //no providers for this service
528                                     logger.info("[" + this.getName() + "]"
529                                                     + " No providers found for Product Service.");
530                             }
531                     } catch (THOMASException e) {
532                             logger.error("[" + getName() + "] " + e.getMessage());
533                     }
534                     // Search for the service "Addition"
535                     serviceName = "Addition";
536                     searchKeywords.clear();
537                     searchKeywords.add(serviceName);
538                     foundServices.clear();
539
540                     do {
541                             // Waiting for services
542                             try {
543                                     Thread.sleep(2 * 1000);
544
545                                     foundServices = sfProxy.searchService(searchInputs,
546                                                     searchOutputs, searchKeywords);
547
548                             } catch (InterruptedException e) {
549                                     logger.error("[" + getName() + "] " + e.getMessage());
550
551                             } catch (THOMASException e) {
552                                     logger.error("[" + getName() + "] " + e.getMessage());
553                             }
554
555                     } while (foundServices.isEmpty());
556
557                     // Request the execution of the service "Addition"
558                     try {
559                             String serviceOWLS = sfProxy
560                                             .getService(foundServices.get(0).get(0));
561                             Oracle oracle = new Oracle(serviceOWLS);
562
563                             // Agents or Organizations that can execute the service for me.
564                             ArrayList<Provider> providers = oracle.getProviders();
565                             // Groundings to execute directly the service by myself.
566                             ArrayList<String> providersGroundingWSDL = oracle
567                                             .getProvidersGroundingWSDL();
568                             // Get service inputs
569                             ArrayList<String> serviceInputs = oracle.getOwlsProfileInputs();
570                             // Put the service inputs values
571                             HashMap<String, String> agentInputs = new HashMap<String, String>();
572
573                             if (!providers.isEmpty()) {
574                                     // Agents or Organizations that can execute the service for me.
575                                     // YOUR CODE STARTS HERE
576                                     //
577                                     // serviceInputs.clear();
578                                     // serviceInputs = oracle.getOwlsProfileInputs();
579                                     //
580                                     // agentInputs.clear();
581                                     // agentInputs = new HashMap<String, String>();
582                                     // for (String input : serviceInputs) {
583                                     //      if (input.equalsIgnoreCase("x")) {
584                                     //          agentInputs.put(input, "5");
585                                     //      } else if (input.equalsIgnoreCase("y")) {
586                                     //              agentInputs.put(input, resultEquation);
```

```
587                                     //      } else {
588                                     //              agentInputs.put(input, "0");
589                                     //      }
590                                     // }
591                                     //
592                                     // logger.info("[" + this.getName() + "]" + " Requesting Addition Service.");
593                                     //
594                                     // msg = null;
595                                     // msg = new ACLMessage(ACLMessage.REQUEST);
596                                     // msg.setHeader("serviceName", serviceName);
597                                     // msg.setReceiver(new AgentID(providers.get(0).getEntityID()));
598                                     // msg.setProtocol("fipa-request");
599                                     // msg.setSender(getAid());
600
601                                     // content = "";
602                                     // content = st.buildServiceContent(oracle.getServiceName(), agentInputs);
603                                     // msg.setContent(content);
604
605                                     // ///this.send_request(msg);
606
607                                     // // The agent creates the CFactory that creates processors that
608                                     // // initiate REQUEST protocol conversations. In this example
609                                     // // the CFactory gets the name "TALK", we don't add any
610                                     // // additional message acceptance criterion other than the required
611                                     // // by the REQUEST protocol (null) and we do not limit the number of
612                                     // // simultaneous processors (value 0)
613                                     // CFactory talkFactoryForAddition = new AdditionFIPA_REQUEST().newFactory("
                                            AdditionTalk", null, msg, 1, firstProcessor.getMyAgent(), 0);
614
615                                     // // The factory is setup to answer start conversation requests from
616                                     // // the agent using the REQUEST protocol.
617
618                                     // this.addFactoryAsInitiator(talkFactoryForAddition);
619
620                                     // // finally the new conversation starts. Because it is synchronous,
621                                     // // the current interaction halts until the new conversation ends.
622                                     // this.startSyncConversation("AdditionTalk");
623
624                                     // inputs.clear();
625                                     // st.extractServiceContent(requestResult, inputs);
626                                     // resultEquation = inputs.get("Result");
627                                     //
628                                     // YOUR CODE ENDS HERE
629
630                             } else if (!providersGroundingWSDL.isEmpty()) {
631                                     // Groundings to execute the service directly by myself.
632                                     // YOUR CODE STARTS HERE
633                                     //
634                                     // serviceInputs = oracle.getOwlsProfileInputs();
635
636                                     // agentInputs.clear();
637
638                                     // agentInputs = new HashMap<String, String>();
639                                     // for (String input : serviceInputs) {
640                                     //      agentInputs.put(input, resultEquation);
641                                     // }
642
643                                     // logger.info("[" + this.getName() + "]" + " Executing Addition Service.");
644
645                                     // HashMap<String, Object> resultExecution = st.executeWebService(
                                            providersGrounding.get(0), agentInputs);
646
647                                     // Double resultContent = (Double) resultExecution.get("Result");
648
649                                     // logger.info("[" + this.getName() + "] Final result: " + resultContent + ".");
650                                     //
651                                     // YOUR CODE ENDS HERE
652                             } else {
653                                     //no providers for this service
```

```
654                                 logger.info("[" + this.getName() + "]"
655                                                 + " No providers found for Addition Service.");
656                         }
657                 } catch (THOMASException e) {
658                         logger.error("[" + getName() + "] " + e.getMessage());
659                 }
660
661         } // End of method execution()
662
663         // This method is executed just before the agent ends its execution
664         @Override
665         protected void finalize(CProcessor firstProcessor,
666                         ACLMessage finalizeMessage) {
667                 CAgent cAgent = firstProcessor.getMyAgent();
668                 logger.info("[" + cAgent.getName() + "] finalize method executed.");
669
670         } // End of method finalize()
671
672         /**
673          * Method to remove all the services that this Agent provides and has
674          * registered in SF.
675          *
676          * @param cAgent
677          */
678         private void removePublishedServices(CAgent cAgent) {
679                 for (String service : myLocalData.getRegisteredServices()) {
680                         deregisterService(service);
681                 }
682
683         } // End of method removePublishedServices ()
684
685         /**
686          * This method makes the Agent leave all the acquired Roles.
687          *
688          * @param cAgent
689          */
690         private void leaveRoles(CAgent cAgent) {
691                 // Requesting for the list of roles and units in which the agent is in a
692                 // specific moment to leave them.
693
694                 ArrayList<UnitRolePair> agentPlayingRoles = myLocalData
695                                 .getPlayingRolesInReverseOrder();
696                 for (UnitRolePair playingRole : agentPlayingRoles) {
697                         try {
698                                 logger.debug("["
699                                                 + cAgent.getName()
700                                                 + "] leaving role \""
701                                                 + playingRole.getRoleID()
702                                                 + "\" at \""
703                                                 + playingRole.getUnitID()
704                                                 + "\": "
705                                                 + omsProxy.leaveRole(playingRole.getRoleID(),
706                                                                 playingRole.getUnitID()));
707                         } catch (Exception e) {
708                                 logger.error("[" + cAgent.getName()
709                                                 + "] Exception in LeaveRole method: " + e.getMessage());
710                         }
711                 }
712         }
713
714         /**
715          * This method makes the Agent deregister all the created Roles.
716          *
717          * @param cAgent
718          */
719         private void deregisterRoles(CAgent cAgent) {
720                 // Deregister the roles I have created
721                 ArrayList<UnitRolePair> agentCreatedRoles = myLocalData
722                                 .getCreatedRolesInReverseOrder();
```

```
723                 for (UnitRolePair createdRole : agentCreatedRoles) {
724                         try {
725                                 logger.debug("["
726                                                 + cAgent.getName()
727                                                 + "] deregistering role \""
728                                                 + createdRole.getRoleID()
729                                                 + "\" at \""
730                                                 + createdRole.getUnitID()
731                                                 + "\": "
732                                                 + omsProxy.deregisterRole(createdRole.getRoleID(),
733                                                                 createdRole.getUnitID()));
734                         } catch (Exception e) {
735                                 logger.error("[" + cAgent.getName()
736                                                 + "] Exception in DeregisterRole method: "
737                                                 + e.getMessage());
738                         }
739                 }
740         }
741
742         /**
743          * This method makes the Agent deregister all the created Units.
744          *
745          * @param cAgent
746          */
747         private void deregisterUnits(CAgent cAgent) {
748                 // Requesting for the list of created unit by the agent in order
749                 // to deregister them.
750                 for (String unit : myLocalData.getCreatedUnitsInReverseOrder()) {
751                         try {
752                                 logger.debug("[" + cAgent.getName() + "] deregistering unit \""
753                                                 + unit + "\": " + omsProxy.deregisterUnit(unit));
754                         } catch (Exception e) {
755                                 logger.error("[" + cAgent.getName()
756                                                 + "] Exception in DeregisterUnit method: "
757                                                 + e.getMessage());
758                         }
759                 }
760         }
761
762         /**
763          * Method to register the service on SF.
764          *
765          * @param serviceName String with the name used to call the user's service.
766          * @param profile ProfileDescription of the user's service.
767          */
768         private void registerService(String serviceName) {
769                 try {
770                         ArrayList<String> resultRegister = sfProxy
771                                         .registerService(serviceName);
772                         Iterator<String> iterRes = resultRegister.iterator();
773                         String registerRes = "";
774                         while (iterRes.hasNext()) {
775                                 registerRes += iterRes.next() + "\n";
776                         }
777                         logger.debug("[" + this.getName() + "] Result registerService: "
778                                         + registerRes);
779
780                         String[] parts = resultRegister.get(0).split(": ");
781                         String serviceProfile = parts[1].trim();
782
783                         myLocalData.addRegisteredServices(serviceProfile);
784
785                 } catch (THOMASException e) {
786                         logger.error("[" + getName() + "]" + e.getMessage());
787                 }
788         }
789
790         /**
791          * Deregistering the service from SF
```

```
792             *
793             * @param serviceName
794             */
795            private void deregisterService(String service) {
796                    try {
797                            String serviceToRemove = service.substring(
798                                            service.lastIndexOf("/") + 1, service.indexOf("."));
799                            logger.debug("[" + getName() + "] deregistering service \""
800                                            + serviceToRemove + "\".");
801
802                            sfProxy.deregisterService(service);
803
804                    } catch (Exception ex) {
805                            logger.error("[" + getName()
806                                            + "] Exception in RemoveProvider method: "
807                                            + ex.getMessage());
808                    }
809
810            }// end of method DeregisterService(String service)
811
812    } // End of class  JamesAgent
```

APPENDIX

F

Utils.java file

```java
/**
 * This class has been generated using Gormas2Magentix tool.
 *
 * @author Mario Rodrigo - mrodrigo@dsic.upv.es
 *
 */
package EMFGormas_Example;

import java.util.ArrayList;
import java.util.Collections;

import org.apache.log4j.Logger;

import es.upv.dsic.gti_ia.cAgents.CAgent;
import es.upv.dsic.gti_ia.organization.OMSProxy;
import es.upv.dsic.gti_ia.organization.THOMASException;

/**
 * Utilities class for manage everything related to Units, Roles, etc. in
 * THOMAS.
 *
 * @author mrodrigo
 *
 */
public class Utils {

	/**
	 * Class to save what roles are played by an Agent. These roles are saved as
	 * pairs (RoleID, UnitID).
	 *
	 * @author mrodrigo
	 *
	 */
	static class UnitRolePair {

		// ----------------------------------------
		// FIELDS of the class
		// ----------------------------------------
		private String roleID;
		private String unitID;

		// ----------------------------------------
		// CONSTRUCTOR of the class
		// ----------------------------------------
		/**
```

```
46                  * Constructor of the class.
47                  *
48                  * @param unitID
49                  * @param roleID
50                  */
51                 public UnitRolePair(String roleID, String unitID) {
52                         this.roleID = roleID.toLowerCase();
53                         this.unitID = unitID.toLowerCase();
54
55                 }
56
57                 // ----------------------------------------
58                 // METHODS of the class
59                 // ----------------------------------------
60                 /**
61                  * @return the roleName
62                  */
63                 public String getRoleID() {
64                         return roleID;
65                 }
66
67                 /**
68                  * @param roleID
69                  *            the roleName to set
70                  */
71                 public void setRoleID(String roleID) {
72                         this.roleID = roleID;
73                 }
74
75                 /**
76                  * @return the unitName
77                  */
78                 public String getUnitID() {
79                         return unitID;
80                 }
81
82                 /**
83                  * @param unitID
84                  *            the unitName to set
85                  */
86                 public void setUnitID(String unitID) {
87                         this.unitID = unitID;
88                 }
89
90                 /*
91                  * (non-Javadoc)
92                  *
93                  * @see java.lang.Object#hashCode()
94                  */
95                 @Override
96                 public int hashCode() {
97                         final int prime = 31;
98                         int result = 1;
99                         result = prime * result
100                                        + ((roleID == null) ? 0 : roleID.hashCode());
101                        result = prime * result
102                                        + ((unitID == null) ? 0 : unitID.hashCode());
103                        return result;
104                }
105
106                /*
107                 * Compare current UnitRolePair with specified UnitRolePair
108                 *
109                 * @see java.lang.Object#equals(java.lang.Object)
110                 */
111                @Override
112                public boolean equals(Object obj) {
113                        if (this == obj)
114                                return true;
```

```
115                         if (obj == null)
116                                 return false;
117                         if (!(obj instanceof UnitRolePair))
118                                 return false;
119                         UnitRolePair other = (UnitRolePair) obj;
120                         if (roleID == null) {
121                                 if (other.roleID != null)
122                                         return false;
123                         } else if (!roleID.equals(other.roleID))
124                                 return false;
125                         if (unitID == null) {
126                                 if (other.unitID != null)
127                                         return false;
128                         } else if (!unitID.equals(other.unitID))
129                                 return false;
130                         return true;
131                 }
132
133         } // End of class PlayingRole
134
135         /**
136          * Class to mantain local data about what roles are played and or create by
137          * an Agent. It also contains a list with the started agents and another
138          * with the created units.
139          *
140          * @author mrodrigo
141          *
142          */
143         static class LocalData {
144
145                 // ----------------------------------------
146                 // FIELDS of the class
147                 // ----------------------------------------
148                 /**
149                  * ArrayList to save the {@link UnitRolePair}s played by each Agent.
150                  * This ArrayList will be used to properly leave them at shutdown time.
151                  */
152                 private ArrayList<UnitRolePair> playingRoles;
153
154                 /**
155                  * ArrayList to save the {@link UnitRolePair}s created by each Agent.
156                  * This ArrayList will be used to properly deregister them at shutdown
157                  * time.
158                  */
159                 private ArrayList<UnitRolePair> createdRoles;
160
161                 /**
162                  * ArrayList to save the {@link CAgents} started by each Agent. This
163                  * ArrayList will be used to properly shutdown them at shutdown time.
164                  */
165                 private ArrayList<CAgent> startedAgents;
166
167                 /**
168                  * ArrayList to save the Units started by each Agent. This ArrayList
169                  * will be used to properly deregister them at shutdown time.
170                  */
171                 private ArrayList<String> createdUnits;
172
173                 /**
174                  * ArrayList to save the services registered by each Agent. This
175                  * ArrayList will be used to properly deregister them at shutdown time.
176                  */
177                 private ArrayList<String> registeredServices;
178
179                 // ----------------------------------------
180                 // CONSTRUCTOR of the class
181                 // ----------------------------------------
182                 /**
183                  * Constructor of the class
```

```
184                  */
185                 public LocalData() {
186
187                         if (playingRoles == null) {
188                                 playingRoles = new ArrayList<UnitRolePair>();
189                         }
190
191                         if (createdRoles == null) {
192                                 createdRoles = new ArrayList<UnitRolePair>();
193                         }
194
195                         if (startedAgents == null) {
196                                 startedAgents = new ArrayList<CAgent>();
197                         }
198
199                         if (createdUnits == null) {
200                                 createdUnits = new ArrayList<String>();
201                         }
202
203                         if (registeredServices == null) {
204                                 registeredServices = new ArrayList<String>();
205                         }
206                 }
207
208                 // ----------------------------------------
209                 // METHODS of the class
210                 // ----------------------------------------
211                 /**
212                  * Method to get an ArrayList with the roles that the agent is playing.
213                  * The first position in the array corresponds to the last role
214                  * acquired.
215                  *
216                  * @return the playingRoles.
217                  */
218                 protected ArrayList<UnitRolePair> getPlayingRolesInReverseOrder() {
219                         ArrayList<UnitRolePair> reverseCollection = playingRoles;
220                         Collections.reverse(reverseCollection);
221                         return reverseCollection;
222
223                 }
224
225                 /**
226                  * Method to get an ArrayList with the roles that the agent is playing.
227                  *
228                  * @return the playingRoles.
229                  */
230                 protected ArrayList<UnitRolePair> getPlayingRoles() {
231                         return playingRoles;
232                 }
233
234                 /**
235                  * Method to add the provided {@link UnitRolePair} as parameter to the
236                  * ArrayList of roles that this Agent is playing.
237                  *
238                  * @param playingRole
239                  *            the playingRole to add.
240                  */
241                 protected void addPlayingRole(UnitRolePair playingRole) {
242                         if (playingRole != null) {
243                                 this.playingRoles.add(playingRole);
244                         }
245                 }
246
247                 /**
248                  * Method to retrieve the created roles in reverse order. The first in
249                  * list is the last created.
250                  *
251                  * @return the createdRoles.
252                  */
```

```
253                 protected ArrayList<UnitRolePair> getCreatedRolesInReverseOrder() {
254                         ArrayList<UnitRolePair> reverseCollection = createdRoles;
255                         Collections.reverse(reverseCollection);
256                         return reverseCollection;
257                 }
258 
259                 /**
260                  * Method to retrieve the created roles inside the unitID provided as
261                  * parameter. The first in the list is the last created.
262                  *
263                  * @param unitID
264                  * @return the created roles in this unit.
265                  */
266                 protected ArrayList<String> getCreatedRolesInUnit(String unitID) {
267                         ArrayList<String> roles = new ArrayList<String>();
268                         for (UnitRolePair unitRolePair : createdRoles) {
269                                 if (unitRolePair.unitID.equalsIgnoreCase(unitID)) {
270                                         roles.add(unitID);
271                                 }
272                         }
273                         // Change the order, the first will be the last created.
274                         Collections.reverse(roles);
275 
276                         return roles;
277                 }
278 
279                 /**
280                  * Method to retrieve the created roles .
281                  *
282                  * @return the createdRoles.
283                  */
284                 protected ArrayList<UnitRolePair> getCreatedRoles() {
285                         return createdRoles;
286                 }
287 
288                 /**
289                  * Method to add the provided {@link UnitRolePair} as parameter to the
290                  * ArrayList of roles that this Agent has created.
291                  *
292                  * @param createdRole
293                  *            the createdRole to add.
294                  */
295                 protected void addCreatedRole(UnitRolePair createdRole) {
296                         if (createdRole != null) {
297                                 this.createdRoles.add(createdRole);
298                         }
299                 }
300 
301                 /**
302                  * Method to retrieve the started Agents list in reverse order. The
303                  * first in list is the last started.
304                  *
305                  * @return the startedAgents.
306                  */
307                 protected ArrayList<CAgent> getStartedAgentsInReverseOrder() {
308                         ArrayList<CAgent> reverseCollection = startedAgents;
309                         Collections.reverse(reverseCollection);
310                         return reverseCollection;
311                 }
312 
313                 /**
314                  * Method to retrieve the started Agents list.
315                  *
316                  * @return the startedAgents.
317                  */
318                 protected ArrayList<CAgent> getStartedAgents() {
319                         return startedAgents;
320                 }
321 
```

```
322                 /**
323                  * Method to add the provided {@link CAgent} as parameter to the
324                  * ArrayList of agents that this Agent has started.
325                  *
326                  * @param startedAgent
327                  *            the startedAgent to add.
328                  */
329                 protected void addStartedAgent(CAgent startedAgent) {
330                         if (startedAgent != null) {
331                                 this.startedAgents.add(startedAgent);
332                         }
333                 }
334
335                 /**
336                  * Method to retrieve the created Units list in reverse order. The first
337                  * in this list is the last created.
338                  *
339                  * @return the createdUnits
340                  */
341                 protected ArrayList<String> getCreatedUnitsInReverseOrder() {
342                         ArrayList<String> reverseCollection = createdUnits;
343                         Collections.reverse(reverseCollection);
344                         return reverseCollection;
345                 }
346
347                 /**
348                  * Method to retrieve the created Units list.
349                  *
350                  * @return the createdUnits
351                  */
352                 protected ArrayList<String> getCreatedUnits() {
353                         return createdUnits;
354                 }
355
356                 /**
357                  * Method to add the provided {@link CAgent} as parameter to the
358                  * ArrayList of agents that this Agent has started.
359                  *
360                  * @param startedAgent
361                  *            the startedAgent to add.
362                  */
363                 protected void addCreatedUnit(String createdUnit) {
364                         if (createdUnit != null) {
365                                 this.createdUnits.add(createdUnit);
366                         }
367                 }
368
369                 /**
370                  * Method to add the provided service as parameter to the ArrayList of
371                  * services that this Agent has registered.
372                  *
373                  * @param registeredService
374                  *            the registeredServices to add
375                  */
376                 protected void addRegisteredServices(String registeredService) {
377                         if (registeredService != null) {
378                                 this.registeredServices.add(registeredService);
379                         }
380                 }
381
382                 /**
383                  * Method to retrieve the registered Services list.
384                  * @return the registeredServices
385                  */
386                 protected ArrayList<String> getRegisteredServices() {
387                         return registeredServices;
388                 }
389
390         } // End of class LocalData
```

```
391
392         // ----------------------------------------
393         // METHODS of the class
394         // ----------------------------------------
395         /**
396          * Method to get an ArrayList with the roles that the agent is playing. The
397          * first position in the array corresponds to the last role acquired.
398          *
399          * @param searchedAgent
400          * @param omsProxy
401          * @return ArrayList<PlayingRole>
402          */
403         protected static ArrayList<UnitRolePair> queryOMSForRolesPlayedByAnAgent(
404                         CAgent searchedAgent, OMSProxy omsProxy, Logger logger) {
405
406                 ArrayList<ArrayList<String>> agentRoles;
407                 ArrayList<UnitRolePair> agentPlayingRoles = new ArrayList<UnitRolePair>();
408                 UnitRolePair playingRole;
409
410                 try {
411                         agentRoles = omsProxy.informAgentRole(searchedAgent.getName());
412
413                         for (ArrayList<String> pair : agentRoles) {
414                                 String roleName = pair.get(0);
415                                 String unitName = pair.get(1);
416                                 if (roleName != null && unitName != null) {
417                                         playingRole = new UnitRolePair(roleName, unitName);
418                                         agentPlayingRoles.add(playingRole);
419                                 }
420                         }
421
422                 } catch (THOMASException e) {
423                         // This exception is launched if service can not found the agent.
424                         // In this case, the web service provides an error with
425                         // this description: "Not found. The agent <Agent Name> not exists".
426                         // If description is diferent, we show the stack trace. In other
427                         // cases,
428                         // the execution can continue.
429                         if (e.getMessage()
430                                         .contains(searchedAgent.getName() + " not exists")) {
431                                 logger.warn("[" + searchedAgent.getName() + "] "
432                                                 + e.getMessage());
433
434                         } else {
435                                 logger.error("[" + searchedAgent.getName() + "] "
436                                                 + e.getMessage());
437                         }
438                 }
439
440                 // This method lets reverse the order of elements of
441                 // agentPlayingRoles. So, the agent can leave the roles in inverse
442                 // order that he acquires them.
443                 Collections.reverse(agentPlayingRoles);
444                 return agentPlayingRoles;
445         }
446
447 } // End of class Utils
```

APPENDIX

G

Constants.java file

```java
/**
 * This class has been generated using Gormas2Magentix tool.
 *
 * @author Mario Rodrigo - mrodrigo@dsic.upv.es
 *
 */
package EMFGormas_Example;

public class Constants {

	public enum AccessibilityType {
		EXTERNAL("external"), INTERNAL("internal");

		private final String value;

		private AccessibilityType(String value) {
			this.value = value;
		}

		@Override
		public String toString() {
			return value;
		}

	}

	static enum PositionType {
		CREATOR("creator"), MEMBER("member"), SUPERVISOR("supervisor"), SUBORDINATE(
				"subordinate");

		private final String value;

		private PositionType(String value) {
			this.value = value;
		}

		@Override
		public String toString() {
			return value;
		}

	}

	static enum VisibilityType {
		PUBLIC("public"), PRIVATE("private");
```

```
46
47                  private final String value;
48
49                  private VisibilityType(String value) {
50                          this.value = value;
51                  }
52
53                  @Override
54                  public String toString() {
55                          return value;
56                  }
57
58          }
59
60          static enum UnitType {
61                  FLAT("flat"), TEAM("team"), HIERARCHY("hierarchy");
62
63                  private final String value;
64
65                  private UnitType(String value) {
66                          this.value = value;
67                  }
68
69          }
70
71  }
```