Electronic Institution

Scenery & Reputation

Daniel Dinis Teixeira – ddtxra@gmail.com

1/14/2008

CONTENTS

TABLE OF FIGURES

OBJECTIVES

The main objectives of this project were to create a scenery with domotics components and build a service of reputation to classify the enterprise agents. In addition to that some changes have been done related to the Ontology Service. Also the way of how the requests are done was modified and the type of file in use changed to OWL.

SUMMARY

This report begins to explain how to define components in OWL files and use the software to edit. After that, a description of how the requests are done and the Ontology Service works is presented. The next chapter describes the scenery created with domotics components. Succeeding that, the reputation service and trust and the evaluation function is described. Finally, some conclusions and possible improvements are discussed. At the end, there are annexes describing some tricks with the software used to edit OWL files and also examples of configuration files.

COMPONENTS FILES

TOOLS IN USE

The components provided by suppliers and the ones requested by clients are defined in OWL [1] files. OWL [2] is a language for defining and instantiating Web Ontologies. To edit these files Protégé [3] version 3.3.1 was used. The version of OWL used was the 1.0, since Protégé doesn't support OWL 1.1 at present. To make easy the parsing of these files, the OWL API [4] was used. The OWL API is a Java interface and implementation for the W3C Web Ontology Language OWL. The latest version of the API is focused towards OWL 1.1, however the version used was the OWL 1.0, because of the limitation of Protégé. Both versions, 1.0 and 1.1, ca be downloaded at Source Forge [5]

ONTOLOGIES

One of the main characteristics of the scenery is to have suppliers and clients with different ontologies. The OWL files can be divided in two types. On the one hand a type to define ontologies. This type of file has basically the classes and attributes. On the other hand a type used to create components following one of those definitions. The files which define ontologies can be hosted in either an http server or in the system file and are not used directly by any agent. The other type of files imports one of those ontologies to instantiate components.

In the scenery, two ontologies are defined: domotics1 and domotics2. The client, Request1, imports the ontology defined in domotics1. Concerning suppliers, the

figure 1 shows that three of them (Supply1, Supply2 and Supply3) share the ontology defined by domotics1 and the two remainders share the domotics2. This means, since the client imports the ontology defined in domotics1 that Supply5 and Supply4 need to ask the Ontology Service to provide a mapping for their classes.

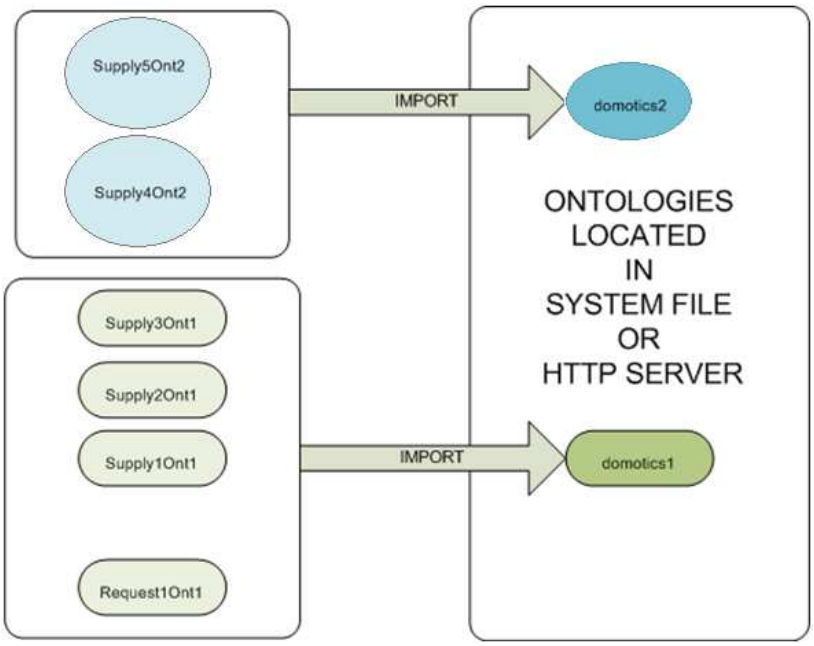

FIGURE 1 - IMPORT SCHEMA

It is important to define correctly the location for files which defines the ontology. Like mentioned before, the location can be both in system file or http server, like the following figure illustrates.

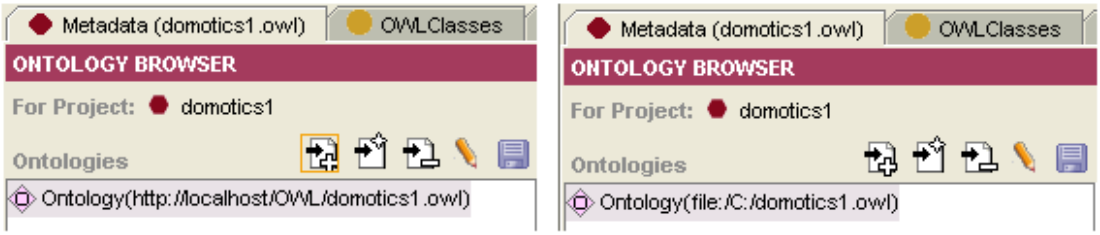

FIGURE 2 – ONTOLOGY LOCATION

The files which contain the components need to import the ontologies, however they don't need to have the correct location specified, since they won't be imported by any other file. So the default ontology can be used (http://www.owl-ontologies.com/date...) .

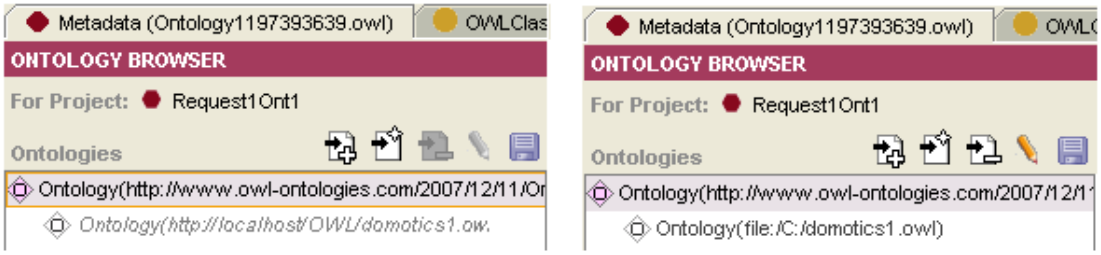

FIGURE 3 – IMPORT ONTOLOGY

CLASSES

Components are instantiated from a class. The classes can be organized as a hierarchy. The following figure shows that the classes defined in the scenery. The mapping classes don't have to be necessary organized following the same hierarchy. For instance, Siren and Alarm are both classes, however Alarm is a Subclass of Security which is a subclass of Domotics, but Siren is just a subclass of Domotics.

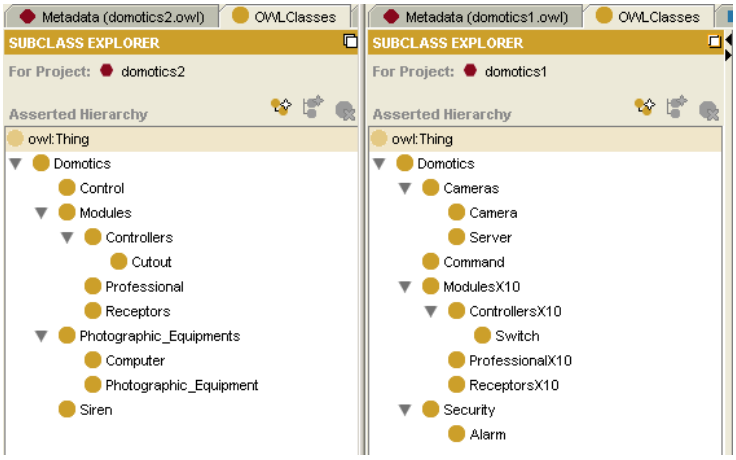

FIGURE 4 – CLASSES

ATTRIBUTES

Before describing the process to create the attributes, an explanation about the different type of attributes' values (allowed, acceptable and preferable) can be required. To understand those concepts an example of an easy attribute will be presented: The color of a car.

Allowed Values: These are all the values that a color can have for the car (yellow, blue, red, green, black, white….)

Acceptable Values: Supposing that a client wants his car either red or black and no other color is acceptable for this client. So only these two colors are the acceptable values.

Preferable Values: Among those two colors, the color that the client really likes is red. The black is acceptable, but won't make him happier. So only red is the preferable value. It is important to notice that a preferable value needs to be contained in the acceptable ones.

The allowed values are common to every agent and then are defined in the definition of the ontology (domotics1 and domotics2). The other values depends on the agents them-selves, some can prefer red others can prefer black, therefore the acceptable and preferable values are defined in the components files.

The following figure shows the different attributes that are defined in the scenery. Each attribute (acceptable values) has a sub-attribute (preferable values), which have the same name but with the suffix *_pref*. The figure 5 shows attributes in order, i.e. *vision_angle* matches with *sight_grade*, *reach* with *range* and so one…

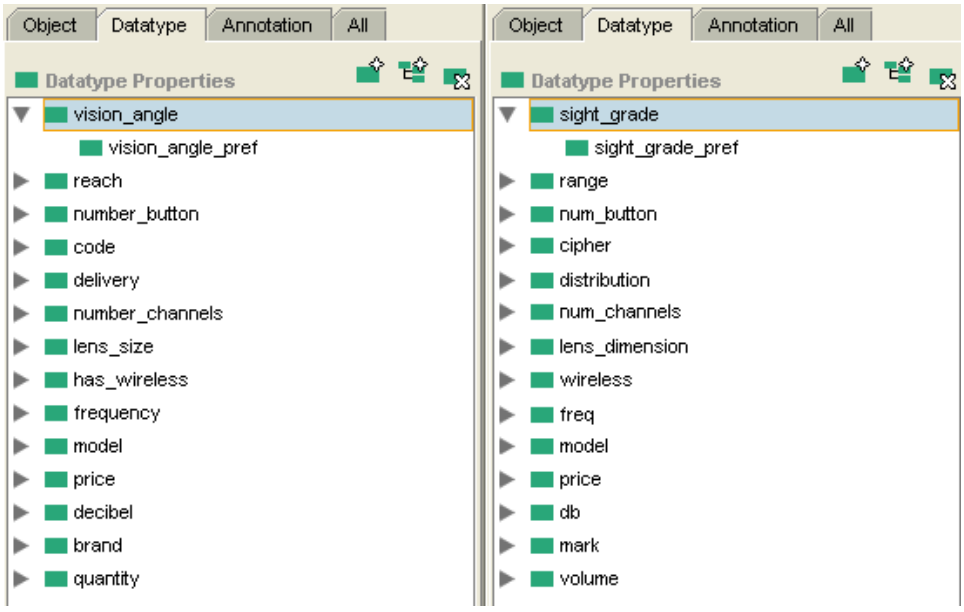

FIGURE 5 - DATATYPES

The attributes need to be assigned to one or several classes. An attribute which is assigned to a class, is automatically assigned to its subclasses. The following figure shows that the attribute decibel belongs to the class Alarm. After that, the range must be chosen. The platform is able to negotiate attributes which are Boolean, Strings, Integer or Floats. As soon as the attributes are matched by the Ontology Service, the *allowed values* of the attributes are considerate to be the same so it can be useful for some attributes to define the *allowed values* to avoid possible mistakes. In the following figure the allowed values are 90db and 110db, which means that all the alarms have either 90db or 110db, therefore *Siren* defined in the other ontology need to allow these same values in order to negotiate *Alarms* with *Sirens*.

FIGURE 6 - DATATYPES DOMAIN

COMPONENTS

Once the ontology is defined it can imported and components can be created. In annex some tricks with Protégé are described, which can help creating an instance more easily.

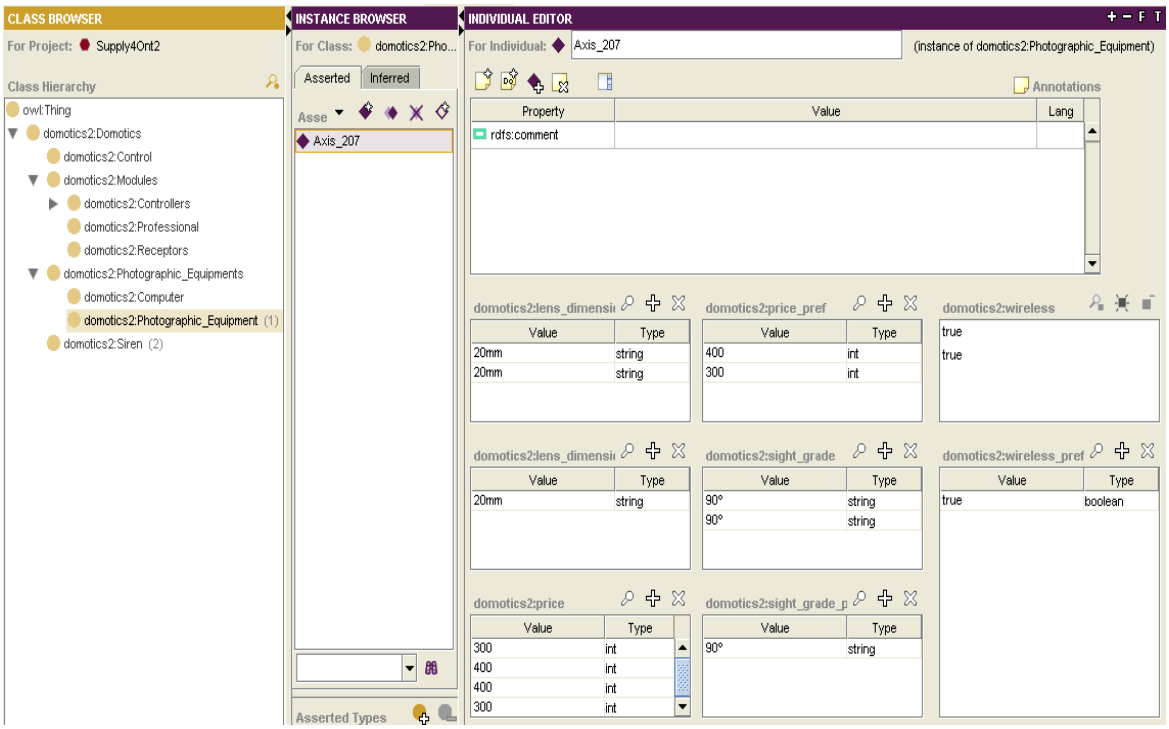

FIGURE 7 - COMPONENTS

CALL FOR PROPOSAL

When a client requests a component, he delegates the process of negotiation to the negotiation mediator giving him information about the components he wants. Only information regarding the class and acceptable values are provided, preferable values are only known by the agent his-self. As soon as a supplier receives a CFP, he tries to find a component of the same class among his components. If he finds one, he checks if the attributes of that component meets the requirements of the request, i.e. the values of the attributes of that component need to be contained within the acceptable values requested by the agent. If this is the case, he starts negotiating that component, otherwise he continues checking components of the same class until finding one. If he finds the class, but any component is acceptable then he refuses the proposal. If he doesn't find the class, he calls the Ontology Service in order to see if there is a class with a different name which satisfies the request. The figure 8 shows that Supply4 need to call the Ontology Service because he can't find any Switch in his components.

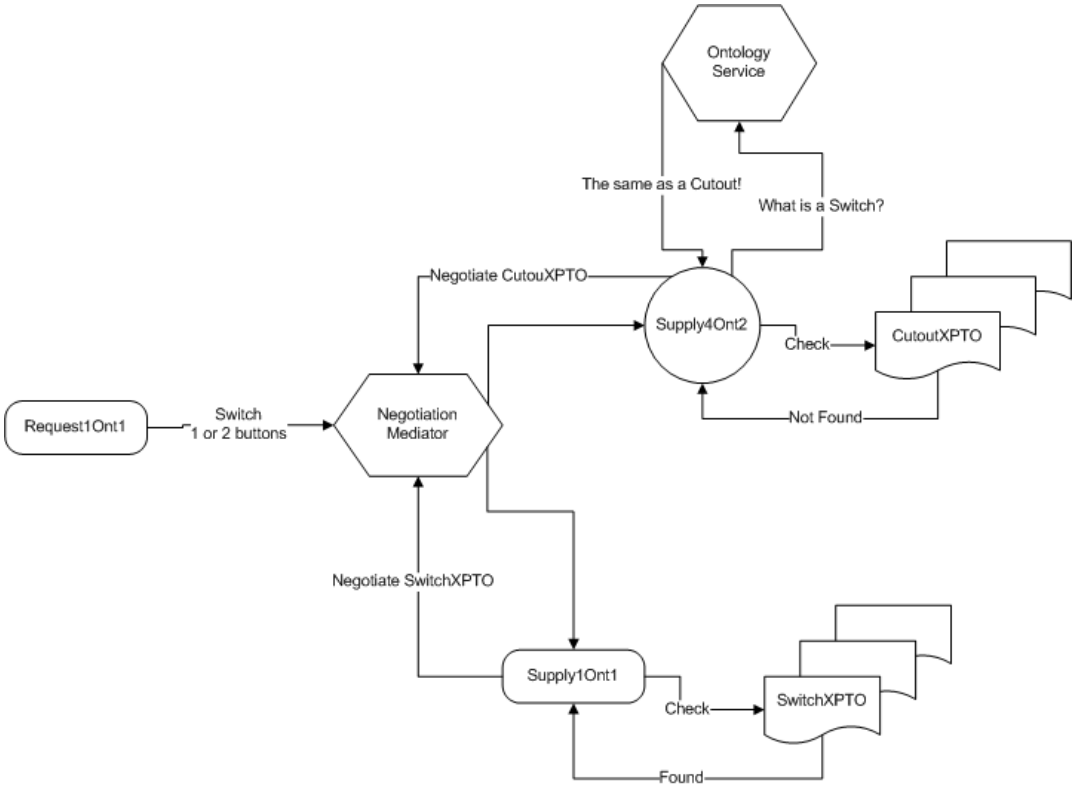

FIGURE 8 - CFP

ONTOLOGY SERVICE

The Ontology Service tries to match components with different names, but before trying to match components, a pre-selection is done regarding the components prices. Only components with prices within the interval of the request are selected, like the figure 9 shows.

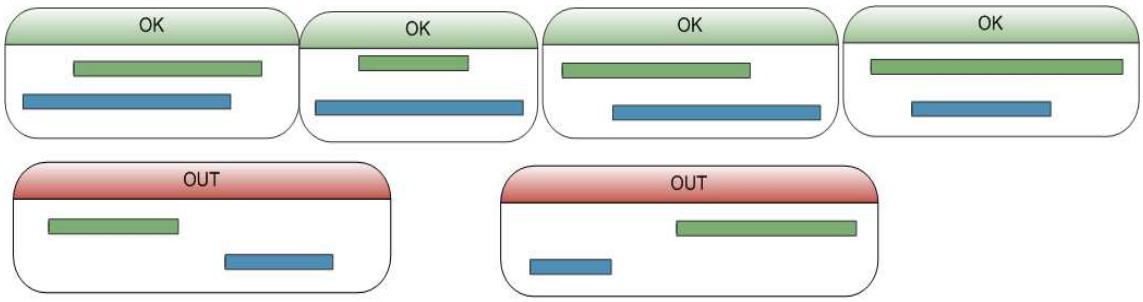

FIGURE 9 - PRESELECTION

Once the pre-selection is done, the service looks at the name of the class and the name of its attributes. There are two ways to compare names. One of them is using N-Gram and the other is using Wordnet. Wordnet is a service that uses socket therefore the result is provided after a long time. It is necessary to reduce the access to Wordnet, so that the first try is done using N-Gram and only if the result with N-Gram is bad the Wordnet is called. Some modifications have been done related to the Wordnet Pearl Server. Instead of calling this server for each sense of a name, it is called only once and the Server work out all the senses and returns the best result. Others improvements are related with the cache to minimize accessing the WordNet server.

The name of attributes can be made up of several names, but it is important to consider that they have to be separated by underscores. For instance *vision_angle* and *sight_grade* are a perfect match, however for WordNet it is impossible to match them if they are not inputted as separated names.

Example:

- Photographic_Equipment (2 names) ←→ Camera (1 name)

In this case, since the number of names that composed each attribute are not the same, the input to WordNet will be Photographic_Equipment and Camera. WordNet is able to substitute the underscore with a space and knows the name *Photographic Equipment* which will return a perfect match with *Camera*.

- Sight_Grade (2 names) ←→ Vision_Angle (2 names)

Here, both attributes have the same number of names, so *sight* with *vision* will be a first input and *grade* with *angle* will be the second one. It is necessary to keep the same order.

SCENERY

The scenery is related to domotic's components. To make this scenery more realistic, the components are based on the components of Central Casa [6]. In this scenery there are a total of six agents. One of them, Request1, is the client. This one requests four different components: a *Command*, a *Switch*, an *Alarm* and a *Camera*. The other five agents are suppliers. It is possible to see in the figure below that Supply1, Supply2 and Supply3 share the same ontology as Request1 (rectangles), however supply 4 and supply 5 don't (circles). Therefore when Supply4 and Supply5 receive a CFP with the ontology of rectangles, they won't be successful in finding components of that class, because those classes are unknown, even though they could have a component which would be a perfect match. To achieve that match those agents need to ask the Ontology Service to provide them information about what is a *Command, Switch, Alarm* and *Camera* in his own ontology. That is actually what happen, when Supply4 or Supply5 asks what is a *Command* the ontology service will reply him that is the same as a *Control* and then it will try to find a perfect match. On the other hand the suppliers who share the same ontology as the client don't need to ask the Ontology Service, they will go through the components who belongs to the class requested and then will check if the attributes are acceptable for both, if it finds a components which respect that then it will negotiate it. For instance, if request1 asks for a component with the brand Panasonic and Sony the supply1 will go search in his cameras until finding a camera Sony or Panasonic.

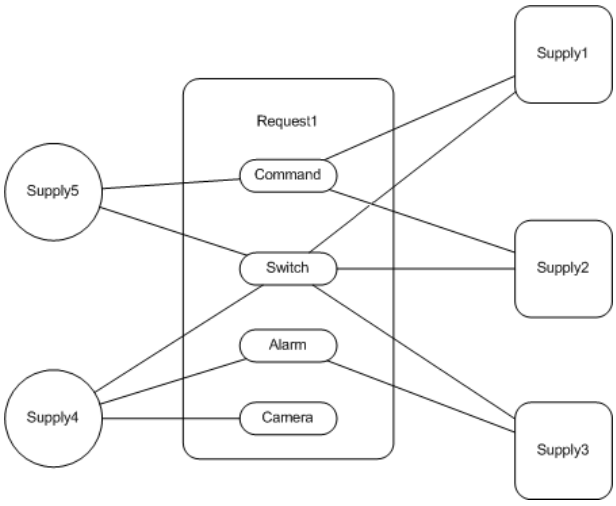

FIGURE 10 - COMPONENTS

CUSTOMER

The customer is represented by the tab “Request” in the GUI of an enterprise agent. The four components are then described for Request1Ont1.

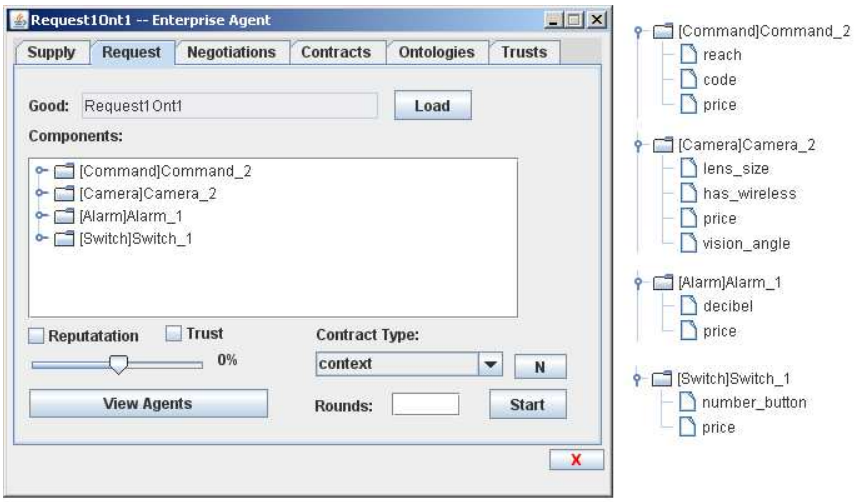

FIGURE 11 - REQUEST GUI

It is possible to click on each of the attributes of the components and a new window will be displayed showing the acceptable and preferable values of the attribute like the figure 12 shows. The figure shows that the agent wants a command with a reach of 30m or 50m, but his preferable value is, in fact, only 30m.

FIGURE 12 – ATTRIBUTE DETAILS

SUPPLIERS

If an agent has something to supply, then the components will be displayed on the tab “Supply”. The following figure shows the different components supplied by the suppliers according to the figure 10. The first column shows Supply1, Supply2 and Supply3 which share the same ontology with the customer. However the second column shows a different ontology.

Like for the customer GUI, it is possible to expand all the components to see its attributes and clicking on the attributes, the attributes details will be shown.

Ontology 1 (Rectangles) Ontology2 (Circles)

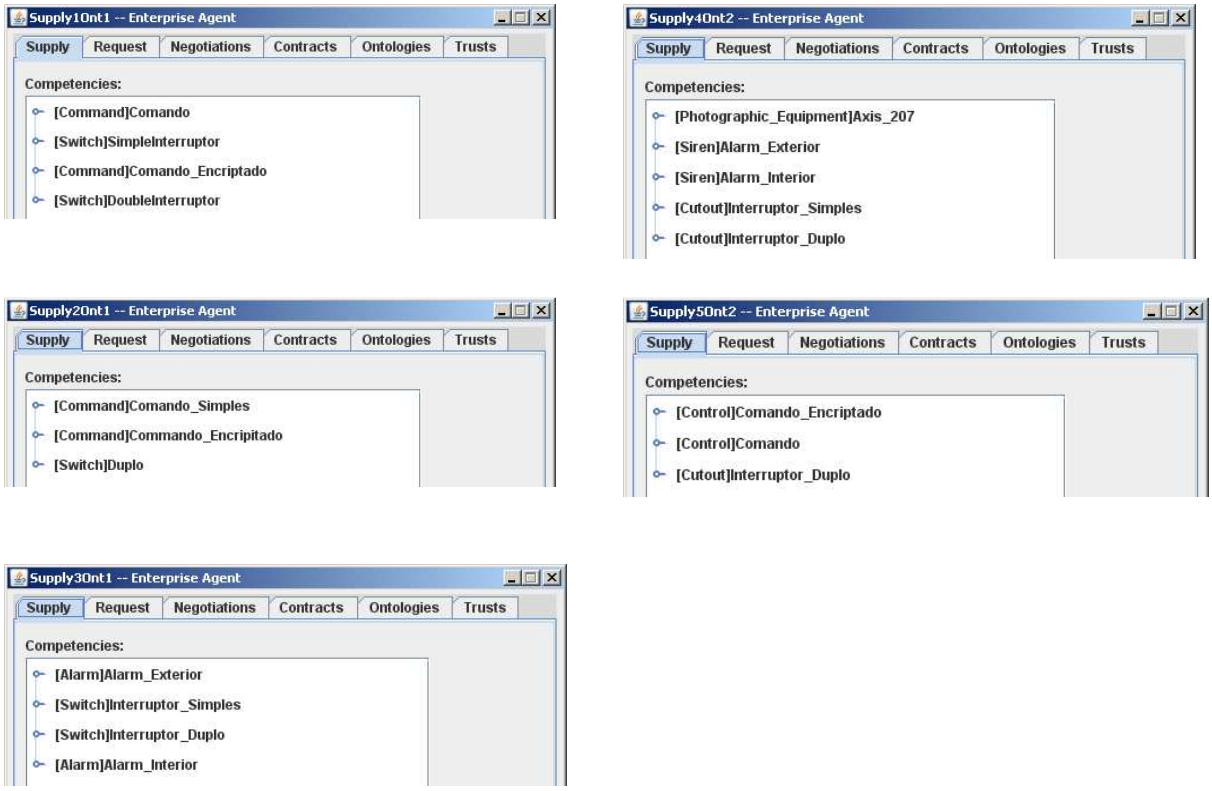

FIGURE 13 - SUPPLY GUI

Due to the fact that Supply4 and Supply5 don't share the same ontology of the customer they will need to contact the Ontology Service. After contacting the Ontology Service Supply4 and Supply5 will have part of the following mappings:

Ontology Rectangles		Ontology Circle		Confidence		
Camera		Photographic_Equipment		0,81	0,81	0,80 = High-Confidence
	price		price	1,00	0,80	
	has_wireless		wireless	0,62 (N-gram)		
	lens_size		lens_dimension	0,85 (Wordnet)		
	Vision_angle		sight_grade	0,73 (Wordnet)		

Ontology Rectangles		Ontology Circle		Confidence		
Alarm		Siren		0,81 (Wordnet)	0,81	0,90 = High-Confidence
	price		price	1,00	1,00	
	decibel		db	1,00 (Wordnet)		

Ontology Rectangles		Ontology Circle		Confidence		
Switch		Cutout		0,81(Wordnet)	0,81	0,82 = High-Confidence
	price		price	1,00	0,82	
	num_button		number_button	0,64 (N-gram)		

Ontology Rectangles		Ontology Circle		Confidence		
Command		Control		1,00 (Wordnet)	1,00	0,97 = High-Confidence
	price		price	1,00	0,94	
	cipher		code	0,81 (WordNet)		
	range		reach	1,00 (WordNet)		

This information can be viewed in the enterprise agent GUI, clicking on the Ontologies tab. It is possible to see in the following figure that Supply4 has a mapping for the class Alarm, Camera and Switch and Supply 5 has a mapping for the class Switch and Command.

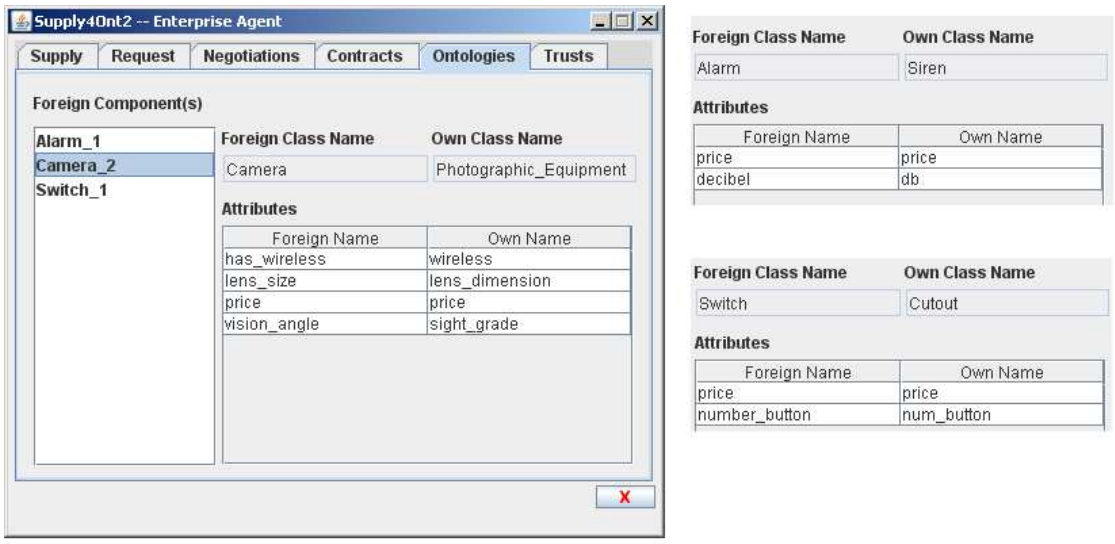

FIGURE 14 - ONTOLOGIES GUI SUPPLY4

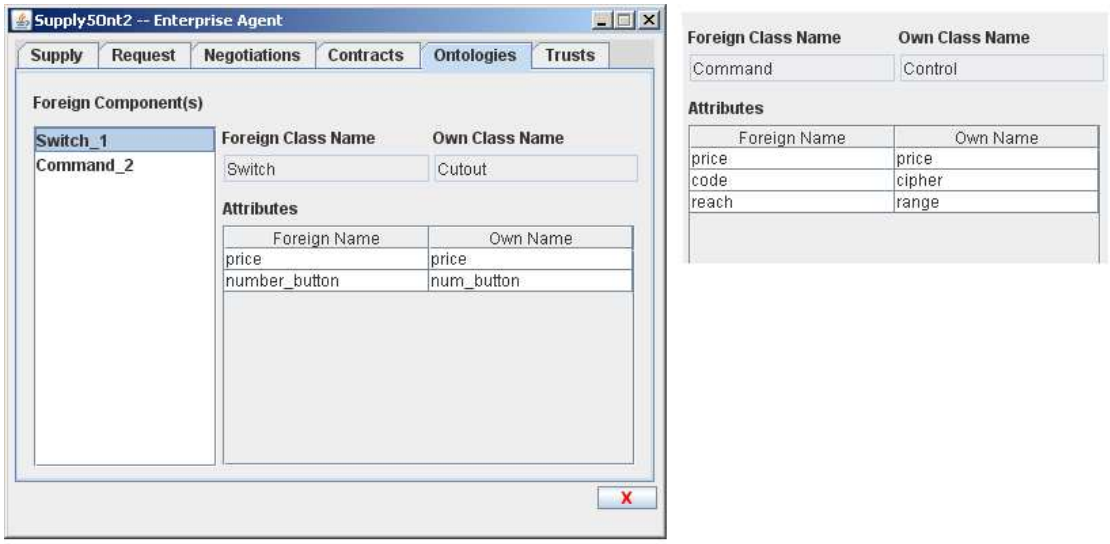

FIGURE 15 - ONTOLOGIES GUI SUPPLY5

For all the classes the final result is equal or over 80 %, which means a high confidence. Therefore the components from both ontologies will be negotiated. Looking again at the figure 13, it is possible to see that 3 agents have commands or controls, 5 agents have switches or cutouts, 2 agents have alarms or sirens and only 1 agent has a camera or photographic equipment. In fact, the figure 16, which represents the state of negotiations, shows the correct number of agents negotiating the components.

Component	Agents Negotiating	Current Winner	Round	Utility
Command_2	3	Supply1Ont1	10/10 --> OVER	29.999998
Camera_2	1	Supply4Ont2	10/10 --> OVER	2.8922763
Alarm_1	2	Supply3Ont1	10/10 --> OVER	1.4962795
Switch_1	5	Supply5Ont2	10/10 --> OVER	8.935559

FIGURE 16 - NEGOTIATIONS

Clicking on the Switch component, the window of utility will be displayed. In that window it is easy to notice that the first winner of the negotiation was Supply2, however after the 6th round Supply5 overtakes. This happens because Supply5 has been flexible with the attribute price for much more time.

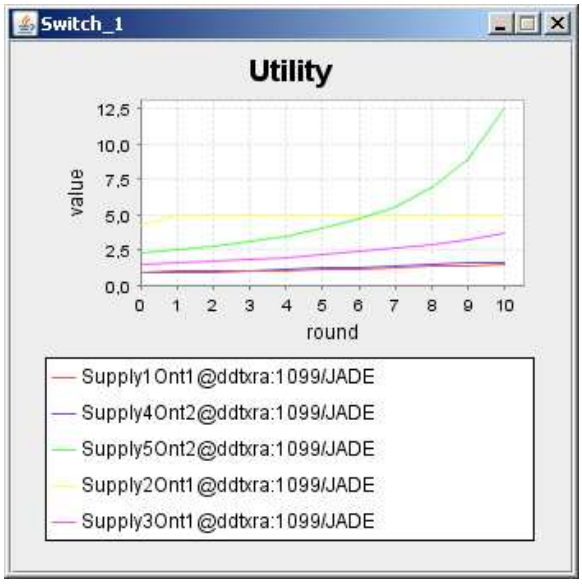

FIGURE 17 - UTILITY

REPUTATION

Each agent has to fulfill obligations when involved in a contract. Each contract can have several obligations concerning the supplier as well as the client. So what happens if one of these agents fulfills or violates obligations? That's where the reputation comes in. If an obligation is fulfilled, then the agent will have more reputation, on the other hand if an agent violates then his reputation score decreases.

The reputation is a result from all the contracts negotiated. When an obligation is fulfilled or violated, the reputation service is notified and stores the record. It happens in the society that agents can have good relations with some agents and bad relations with others. For instance, if an agent A has a good relationship with an agent B, but a very bad with the agent C, the agent B can trust the agent A, but the agent C cannot. The reputation service gives an average of all the contracts, however the trust gives a score related to a relationship between only two agents.

The following picture shows two contracts. One contract between agent Request1Ont1 and Supply1Ont1 and another one between Request1Ont1 and Suppl4Ont2. Supposing that no other contracts have been done before, the Reputation Service will have information about these three agents. Regarding to the trust, Supply1 will have information in his trusts records concerning Request1, but any information about Supply4. In order to have information about that agent, the agent needs to ask the reputation service or make a contract with him.

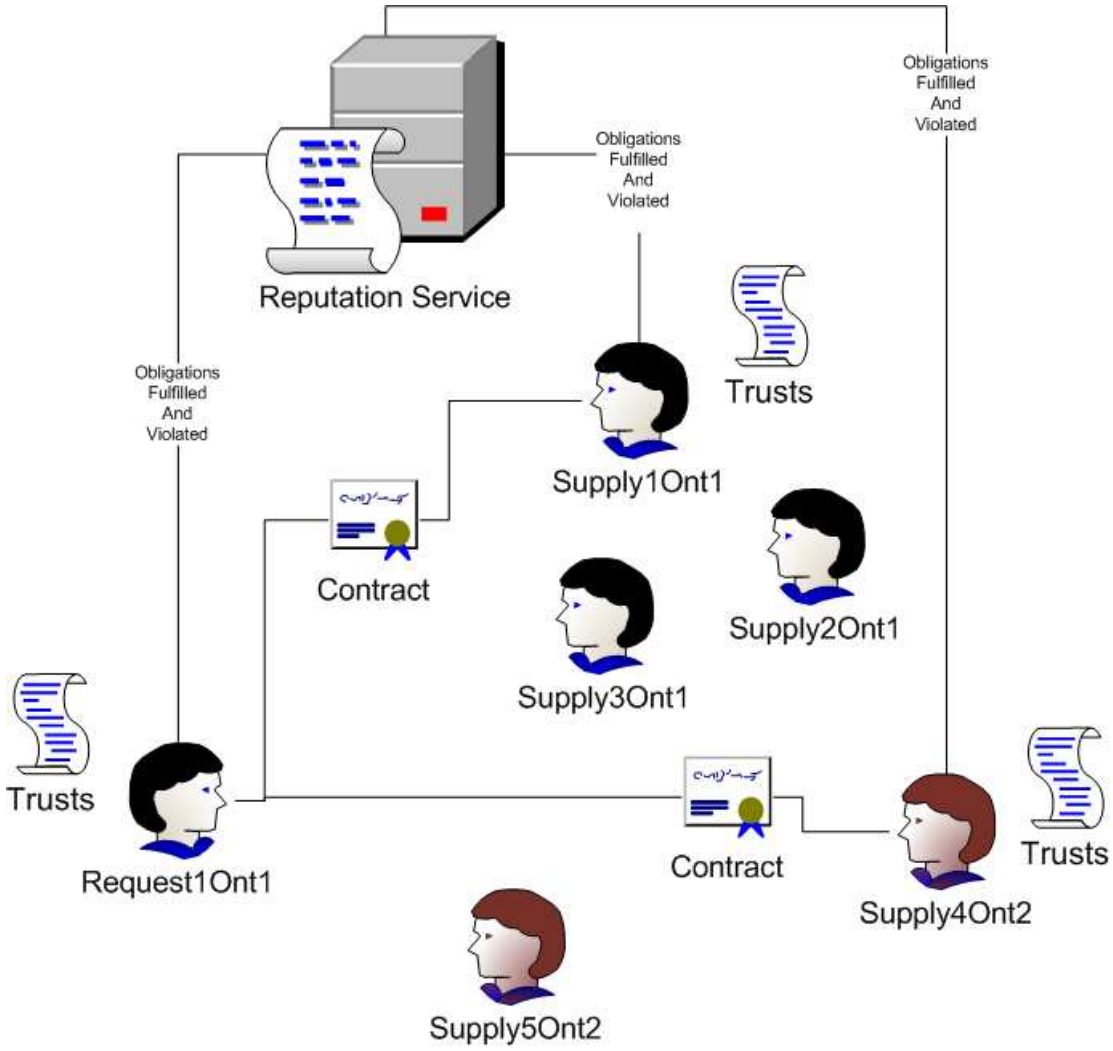

FIGURE 18 - REPUTATION AND TRUST

EVALUATION

The score is worked out according to the following characteristics:

- The score has to increase if a fulfilled obligation is added and decrease in case of a violated obligation.
- The violation of an obligation can have a weight more important than a fulfilled one.
- Some agents trust more quickly than others.

The function used to follow those characteristics is the following:

$$x = (premium * number\_fulfilled\_obligations) \; - \; (penalty * number\_violated\_obligations)$$

$$.f(x) = \begin{cases} -\frac{\text{flexibility} * \text{x}}{flexibility * x + 1} \; (x < 0) \\ \frac{\text{flexibility} * \text{x}}{flexibility * x + 1} \; (x \geq 0) \end{cases}$$

Note: The value of the flexibility is divided by 10, i.e. if the value in the GUI or in the file is 5, the value introduced in the function is 0.5.

This function is represented in the figure below. It is possible to see that if the flexibility is 1,0 the inclination is more vertical than if the value is 0,1. It means that for a flexibility of 0,1 the confidence will move more slowly. After 20 fulfilled obligations and zero violated the result is below 70 % with a flexibility of 0,1, however if the flexibility is 1,0 a score of 75% is reached after the 3rd fulfilled obligation.

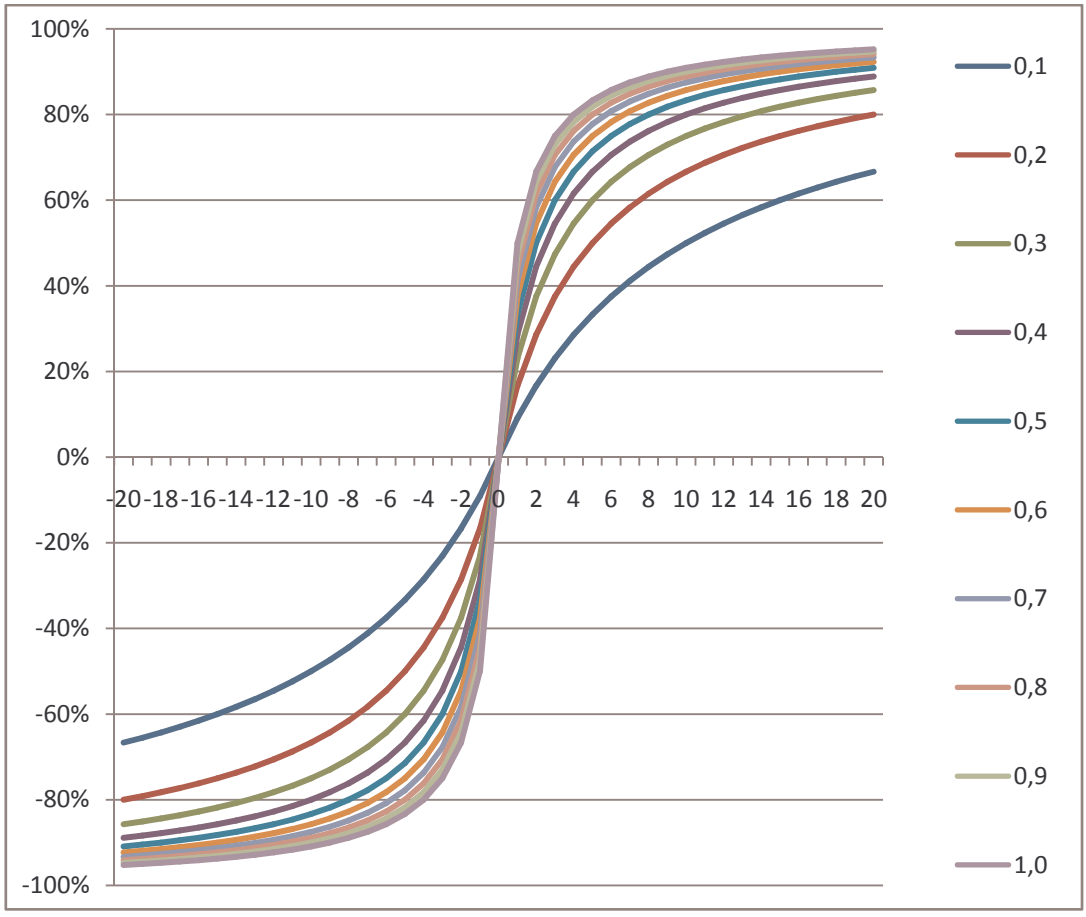

FIGURE 19 - GRAPH FLEXIBILITY

INTERFACES

REPUTATION

The reputation GUI, as well as the trust tab of the enterprise agents GUI, has several controls that change the parameters and consequently the score of an agent. The flexibility can be defined from 1 to 10. The relation between premium and penalty can also be selected. If the value is 1:1 it means that a violated and a fulfilled obligation have the same weight. However if the value selected is 1:2 / 1:4, it means that a violated obligation is weighted two / four times more than a fulfilled one. For instance, if an agent A has 80 % in score and the relation between premium and penalty is 1:4 and if one violated obligation is added, then the agent A needs to fulfill 4 obligations to have 80 % again in score.

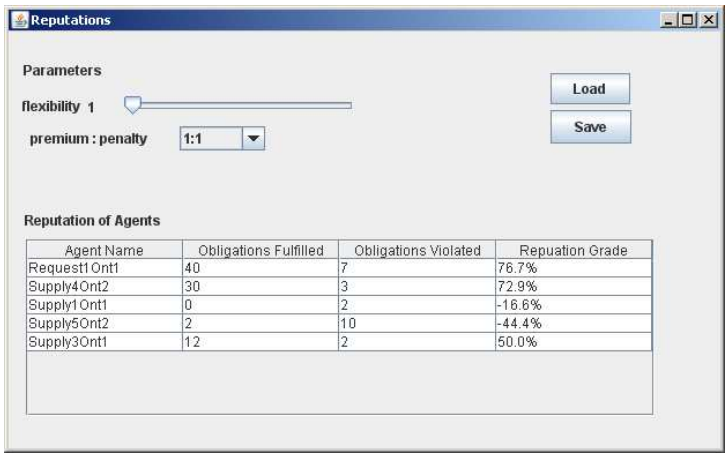

FIGURE 20 - REPUTATION GUI

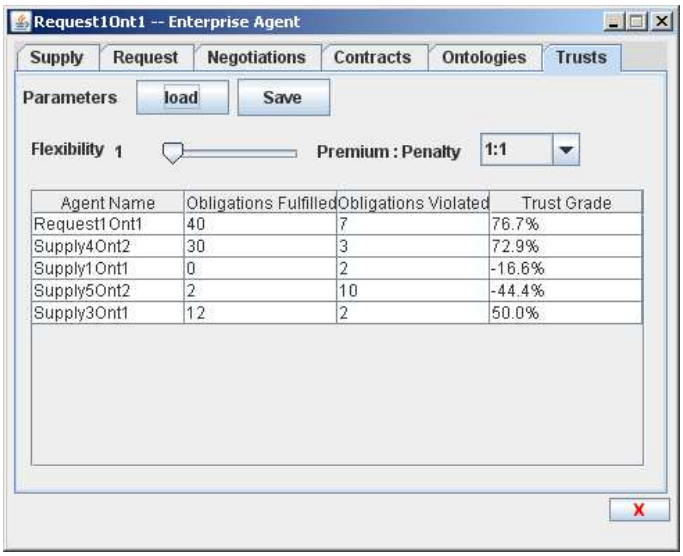

FIGURE 21 - TRUST TAB

SUPPLY

When an agent A receives a CFP for a component that an agent B has requested, he (agent A) can choose if he wants to negotiate his component with the agent B or not. The agent A has four possibilities related to his supplied components. He can choose to supply with all the agents, in this case he has to select none of the two checkboxes. If he wants to choose a minimum value based on his trusts, then he just has to select the trust checkbox. In case of wanting to negotiate based on the reputation service he just has to select the reputation box. If the agent selects both of the checkboxes he looks first at his trust records, if he finds the agent there he uses this value, otherwise he will use the value of the reputation service.

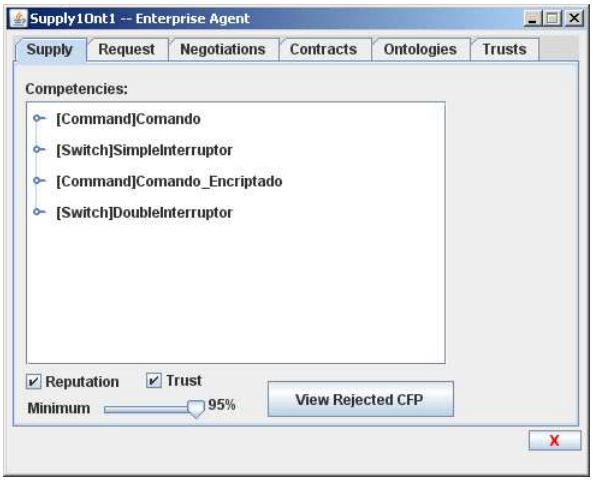

FIGURE 22 - SUPPLY GUI TRUST AND REPUTATION

There is a button *view rejected CFP* that will show the CFP that the agent has refused. For Instance the following figure shows that agent Supply5Ont2 has rejected 4 CFP from agent Request1Ont1 for four different components (Command_2,Camera_2,Alarm_1,Switch_1). The agent refused theses CFP because he defined as minimum value 49 % and the agent had only 28.5 % according to his trust. The figure 23 shows exactly that, it is possible to see that there is a field named *Criterium,* which gives information of who decided not to enter in the negotiation.

The *Criterium* can have different values:

- TRUST (The information is based on trusts of the agent supply)
- REPUTATION (The information is based on the reputation service)
- N/A TRUST (It happens when the agent who supplies doesn't have any information about the agent who requests)
- N/A REPUTATION (It happens when the reputation service doesn't have any information about the agent who requests)

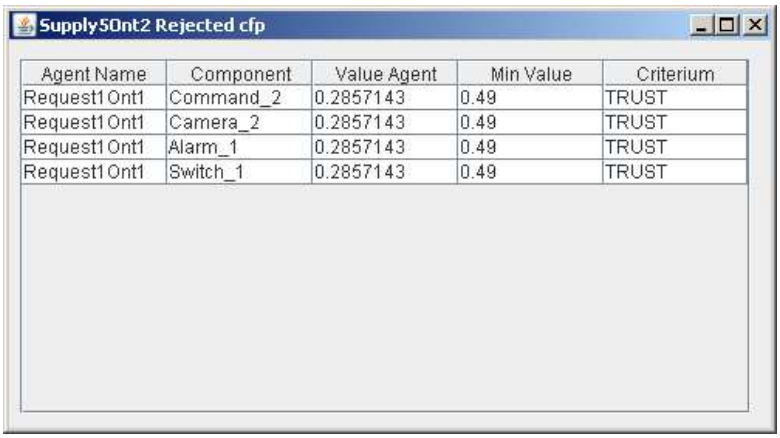

FIGURE 23 - REJECTED CFP

REQUEST

An agent when requests component(s) can choose a minimum value, like the figure shows (78 %). The agent can choose between selecting reputation or trust. If he chooses trust, he will negotiate components only with agents who have a trust value over that minimum value. If he chooses reputation, the same will happen but the records used to select the agents are the ones from the reputation service. In case of none of the two checkboxes are selected, he negotiates with all the agents registered as *enterprise agents* in DF.

It is possible to see the agents with whom the negotiation will proceed clicking on the button *View Agents*.

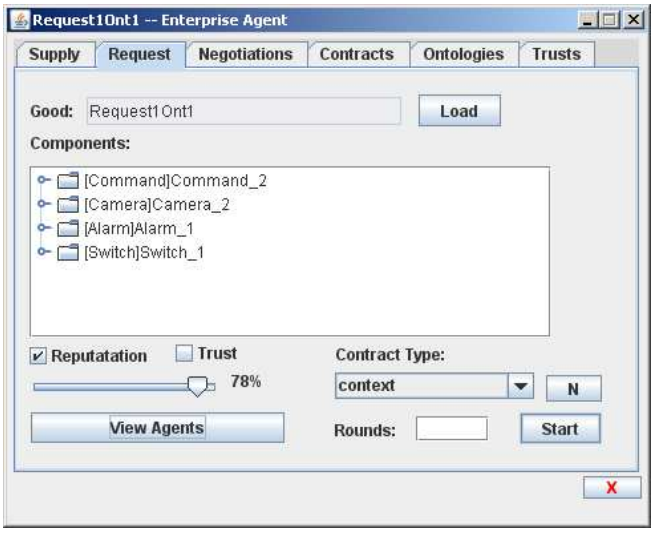

FIGURE 24 – REQUEST GUI TRUST AND REPUTATION

FIGURE 25 - TRUSTED AGENTS GUI

Conclusions and Possible Improvements

All the main objectives proposed have been achieved, however not all the ideas could be implemented. E-Commerce is a fascinating theme that can be explored in so many ways that is impossible to say that the system is perfect, because so many interesting features can be added.

Concerning the reputation service, this one could be extended to all the agents and not only the enterprise agents. The banks, the notaries, the negotiation mediators all of them can be dishonest. The score could be also differentiated, for instance an enterprise agent can be a very bad supplier, but an excellent client. A mechanism of friendship could be added based on "the friend of my friend is my friend". If an agent A blindly trusts an agent B who blindly trusts an agent C, then the agent A could blindly trust the agent C. This transitive property could be added building in this way a real network of friendships as well as swindlers. The minimum value could be applied depending on the component or even in the price (for an higher price a greater confidence is required). The initial value of an unknown client could also vary like the flexibility and the relation premium:penalty.

The Ontology Service could take advantage of the fact that ontologies can be imported from a global localization. For instance after one successful mapping, the Ontology Service could store that http://host123/domotics1.owl#Switch equals http://host345/domotics.owl#Cutout. Therefore next time a Switch from that Ontology is requested there is no need to try a mapping with all classes. The enterprise agents themselves could store this information avoiding using the Ontology Service again for the same class.

Those are some ideas that could be added, but there are much more besides them. At present the platform is truly interesting and tends to approach more and more a real world where the majority of identities are present. Nowadays enterprises still have some apprehension in using these platforms. However, when recognizable identities are present, such as banks and notaries, and a reputation evaluation is implemented the confidence starts to rise among suppliers and customers. There is no doubt that in a few years the use of this kind of platform will increase hugely.

REFERENCES

[1] OWL, http://www.w3.org/2004/OWL/

[2] OWL, http://en.wikipedia.org/wiki/Web_Ontology_Language

[3] Protégé, http://protege.stanford.edu/

[4] OWL API, http://owlapi.sourceforge.net/

[5] OWL API SourceForge, https://sourceforge.net/project/showfiles.php?group_id=90989

[6] Central Casa, http://www.centralcasa.com/

ANNEX A – TRICKS WITH PROTÉGÉ

Protégé 3.3.1 has some problems in creating instances. The first problem concerns the repetition of the sub-attribute values. It is possible to see in the right figure below that, the attribute code, has the values *false*, *true* and *false* again. *False* is repeated twice, this is due to the fact that the preferable value is also *false*. This repetition appears just in design, in OWL code the attribute code has only two values: *false* and *true*.

The other problem is that the range of the sub-attribute can be defined in the ontology, however when one tries to introduce the value in a sub-attribute, the default range is *string*. So in case of types different from *string* a red boundary will be shown until choosing the correct type. In the following figure, it is possible to see for the attribute *code* (Type: boolean) that if a new value wants to be defined in *code_pref* a red boundary will be shown around the attribute *code*. So it is necessary to change *string* to *boolean* in order to introduce a valid value.

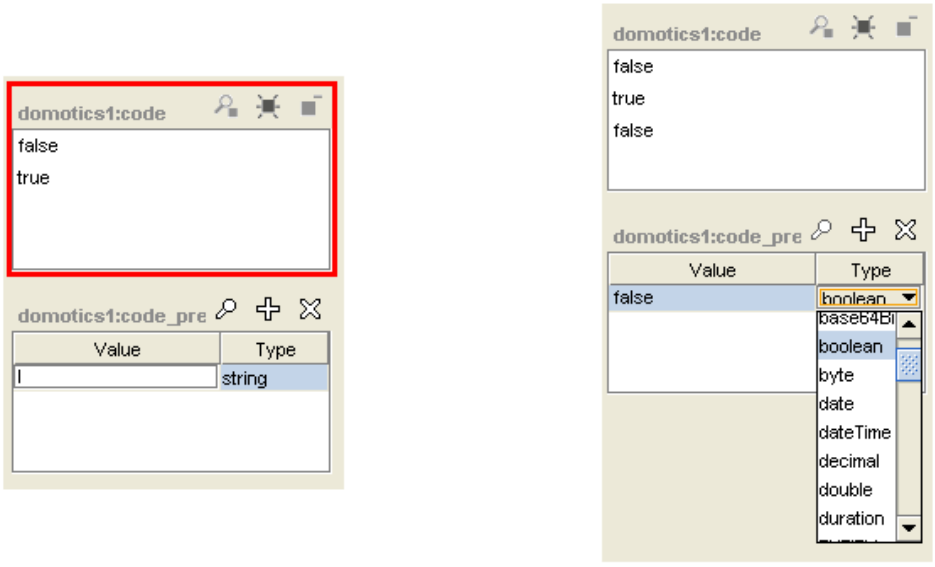

FIGURE 26 – PREFERABLE VALUES

ANNEX B – CONFIGURATION FILES

COMPONENTS

Regarding to the configuration file, the configuration of requested components must be done with the property *good*. In case of supplier the property *components* should be used. An agent can be a supplier and a client at the same time, the configuration of the agent simply need both arguments. In the example below, there is an example of a file.

REPUTATION

The reputation service can be launched with the argument *reputation_file*, which will load historical records of obligations and contracts. The same can be done with enterprise agents, however with the argument *trust_file* instead.

EXAMPLE OF A CONFIGURATION FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ei-config SYSTEM "ei-agents.dtd">
<ei-config>
<jade-parameters>
        <parameter>java.security.policy=policy.txt</parameter>
        <parameter>jade.security.authentication.logincallback=CmdLine</parameter>
        <parameter>jade.security.authentication.loginmodule=NT</parameter>
        <parameter>java.security.auth.login.config=jaas.conf</parameter>
        <parameter>owner=rneves:rneves</parameter>
        <parameter>imtp=jade.imtp.rmi.RMISSLIMTPManager</parameter>
</jade-parameters>
<institutional-agents>
        <agent name="negmed">
                <class>ei.service.negotiation.qnegotiation.QFNegotiationMediator</class>

        <argument>handler_class=ei.service.negotiation.qnegotiation.QFNegotiationHandler<
/argument>
                <argument>service=IS-negotiation-mediator</argument>
                <argument>contract_xsd=D:\rneves\rational                        projs\EI
Code\contract.xsd</argument>
        </agent>
        <agent name="normenv">
                <class>ei.normenv.NormativeEnvironment</class>
                <argument>service=IS-normative-environment</argument>
                <argument>jess_file=ei.clp</argument>
        </agent>
        <agent name="reputation">
                <class>ei.service.reputation.Reputation</class>
                <argument>service=IS-reputation</argument>
                <argument>reputation_file=Cenario_Domotica/reputation.xml</argument>
        </agent>
        <agent name="bank">
                <class>ei.agent.role.BankAgent</class>
        </agent>
        <agent name="delivery-tracker">
                <class>ei.agent.role.DeliveryTrackerAgent</class>
        </agent>
        <agent name="messenger">
                <class>ei.agent.role.MessengerAgent</class>
        </agent>
```

```
        <agent name="Request1Ont1">
                <class>ei.agent.enterpriseagent.AskOntoMapAgent</class>
                <argument>good=Cenario_Domotica/Request1Ont1.owl</argument>
                <argument>external_contract_viewer=C:\Program                Files\Mozilla
Firefox\firefox.exe</argument>
                <argument>trust_file=Cenario_Domotica/trustsRequest1.xml</argument>
        </agent>
        <agent name="Supply1Ont1">
                <class>ei.agent.enterpriseagent.qnegotiation.QEnterpriseAgent</class>
                <argument>components=Cenario_Domotica/Supply1Ont1.owl</argument>
                <argument>external_contract_viewer=C:\Program                Files\Mozilla
Firefox\firefox.exe</argument>
                <argument>trust_file=Cenario_Domotica/trustsSupply1.xml</argument>
        </agent>
        <agent name="Supply2Ont1">
                <class>ei.agent.enterpriseagent.qnegotiation.QEnterpriseAgent</class>
                <argument>components=Cenario_Domotica/Supply2Ont1.owl</argument>
                <argument>external_contract_viewer=C:\Program                Files\Mozilla
Firefox\firefox.exe</argument>
                <argument>trust_file=Cenario_Domotica/trustsSupply2.xml</argument>
        </agent>
        <agent name="Supply3Ont1">
                <class>ei.agent.enterpriseagent.AskOntoMapAgent</class>
                <argument>components=Cenario_Domotica/Supply3Ont1.owl</argument>
                <argument>external_contract_viewer=C:\Program                Files\Mozilla
Firefox\firefox.exe</argument>
                <argument>trust_file=Cenario_Domotica/trustsSupply3.xml</argument>
        </agent>
        <agent name="Supply4Ont2">
                <class>ei.agent.enterpriseagent.AskOntoMapAgent</class>
                <argument>components=Cenario_Domotica/Supply4Ont2.owl</argument>
                <argument>external_contract_viewer=C:\Program                Files\Mozilla
Firefox\firefox.exe</argument>
                <argument>trust_file=Cenario_Domotica/trustsSupply4.xml</argument>
        </agent>
        <agent name="Supply5Ont2">
                <class>ei.agent.enterpriseagent.AskOntoMapAgent</class>
                <argument>components=Cenario_Domotica/Supply5Ont2.owl</argument>
                <argument>external_contract_viewer=C:\Program                Files\Mozilla
Firefox\firefox.exe</argument>
                <argument>trust_file=Cenario_Domotica/trustsSupply5.xml</argument>
        </agent>
        <agent name="osa">
                <class>ei.service.ontology.OntologyMapping</class>
                <argument>service=IS-ontology-mapping</argument>
                <argument>wordnet_host=127.0.0.1</argument>
                <argument>wordnet_port=6180</argument>
                <argument>wordnet_file=WordNetSimilarity.dat</argument>
        </agent>
        <agent name="notary">
                <class>ei.service.notary.Notary</class>
                <argument>service=IS-notary</argument>
        </agent>
        <agent name="rma">
                <class>jade.tools.rma.rma</class>
        </agent>
</institutional-agents>
</ei-config>
```

ANNEX C – REPUTATION RECORDS

REPUTATION FILE

In order to keep historical records of obligations, one type of file has been created. This file is similar to the configuration file, it is also specified in XML. In this file, the several parameters are defined:

- Flexibility – Which is a value from 1 to 10.
- Premium – Which is always 1.
- Penalty – Which can be 1, 2 or 4.

After specifying parameters, records of agents are stored. Each agent has a name, number of contracts, number of fulfilled obligations and number of violated ones.

TRUST FILE

The same type of file can be used to save and load trusts of enterprise agents. Obviously the number of contracts and the other values are related to contracts done with the agent himself. For instance, if agent A did a total of 100 contracts, 20 contracts with agent B and 80 with agent C. The reputation service will store 100 related to that agent, however agent B will store 20 and the agent C will store 80.

EXAMPLE OF FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE reputation SYSTEM "reputation.dtd">
<reputation>
	<parameters>
		<flexibility>1</flexibility>
		<premium>1</premium>
		<penalty>1</penalty>
	</parameters>
	<values>
		<agent>
			<name>Supply1Ont1</name>
			<n_contracts>1</n_contracts>
			<n_fulfilled_obligations>0</n_fulfilled_obligations>
			<n_violated_obligations>2</n_violated_obligations>
		</agent>
		<agent>
			<name>Supply4Ont2</name>
			<n_contracts>21</n_contracts>
			<n_fulfilled_obligations>30</n_fulfilled_obligations>
			<n_violated_obligations>3</n_violated_obligations>
		</agent>
		<agent>
			<name>Request1Ont1</name>
			<n_contracts>3</n_contracts>
			<n_fulfilled_obligations>40</n_fulfilled_obligations>
			<n_violated_obligations>7</n_violated_obligations>
		</agent>
	</values>
</reputation>
```