# INVOLVING WEB-TRADING AGENTS & MAS

## An Implementation for Searching and Recovering Environmental Information

L. Iribarne, N. Padilla, J. A. Asensio, F. Muñoz and J. Criado

*Applied Computer Group, University of Almeria, Spain*

Keywords:     Software agents, Web-trading, Ontology, MAS, WIS, EMS.

Abstract:     The *Web-based Information Systems* appear to facilitate the access of user(s) to different kind of information geographically distributed in different regions (both data and users). In a web system, we have the possibility to use components called *traders* that improve the interoperability with agents or even trading systems. This paper describes details for a *Web Trading Agent* in a *Multi-Agent System*, which implements a distributed information system, called SOLERES. Besides, it presents the communication system based on *ontologies* through which the system agents communicate with each other and with the trading agent inside the SOLERES system (i.e., an Environmental Management System). This architecture is based on a "*Query-Searching/Recovering-Response*" model. For the implementation it uses: an *interface user agent* and the *trader agent*, "processes" ontologies for the agent communication, "data" ontologies for the information storage, SPARQL notation for queries and the JADE platform for the implementation.

## 1  INTRODUCTION

*Web-based Information Systems* (WIS) are an example of distributed information systems that facilitate the information access, decision-making, etc. (Lytras, 2005). WIS facilitates the access of users to different kind of information geographically distributed in different regions (both data and users). In this type of system, interactions are made between "agents" of the system (Web components, subsystems, humans) working in the same ambient (computing space), or even through trading (i.e., mediation components).

Mediation components (i.e., *traders*) enrich the component interoperability in WIS, so system agents must use a common vocabulary to allow the communication between them. This interaction is formally defined by means of ontologies.

Environmental Management Systems (EMS) are an example of WIS. In this kind of information systems, the software agents are applied in four different contexts:

(a) *information management context*: for searching, filtering, recovering and spreading information;

(b) in *control and supervision processes context*: monitoring an element or an activity;

(c) in *cooperative or work group applications context*: to flow information for the member interaction; and

(d) as *personal assistants*: to represent a user if he knows his preferences beforehand.

The SOLERES system (a spatio-temporal environmental management system based in neuronal networks, agents and software components; available at http//:www.ual.es/acg/soleres) is our particular EMS system, which follows a WIS approach. This system has been designed inside a Multi-Agent System (MAS) based on a trader agent paradigm (Asensio *et al.*, 2008). The implementation follows a "*QuerySearching/RecoveringResponse*" model, and it uses:

(a) an *interface user agent* and a *trader agent*,

(b) "processes" ontologies for the agent communication,

(c) "data" ontologies for the information storage,

(d) SPARQL notation for queries and,

(e) the JADE platform for the implementation.

In the next section we will present a review of the SOLERES system, and specifically the web trading agent, showing some communication implementation details by using Multi-Agent System approaches (Shiyong *et al.*, 2007) (Tweedale et al., 2007). After that, we conclude the paper with some future work.

## 2 SOLERES TRADING SERVICE

SOLERES follows a "*QuerySearching / Recoverin-gResponse*" model (QS/RR) for the implementation of the *web-based trading agent* using four contexts (Figure 1) (Russ and Jones, 2006). On this model, the queries made by a user are interpreted by using the interface agent, which interacts with the user interface and performs the query to the SPARQL language. The monitoring process is made by the control agent named "IMIAgent" which receives the query. This intermediary agent asks the trader agent ("TradingAgent") to find out the data localization (searching). Later, the intermediary agent recovers the information (recovering) through the resource agent and replies to the interface agent which made the query (response).
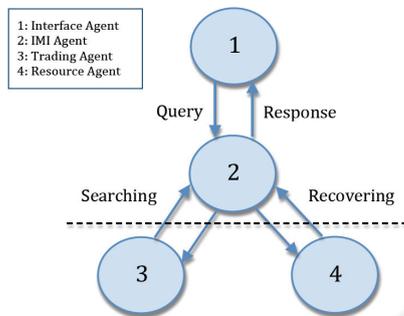


Figure 1: QuerySearching/RecoveringResponse Paradigm.

In an object-oriented programming (OOP) view, a trading service (*trader*) is a software object that serves as a mediator between objects providing certain capacities (exporters) and objects requiring a dynamical use of these capacities (importers).

A trading object (ISO/IEC, 1996) uses five interfaces for the interaction with client objects (exporters and importers): **Register**: allows clients to export service offers in the trader**. Lookup**: allows clients to inquire trader about the service offers stored in the trading service. **Link**: allows the trader to be connected to other traders for the propagation of request in a network. **Admin**: allows administrators to configure the trader. **Proxy**: is used to allow legacy systems properties in federated trader system.

In our work, we use traditional traders' behavior (ISO/IEC, 1996) (Iribarne *et al.*, 2004) to adapt it in order to replace objects (or software components) by software knowledge-based agents. We use the JADE platform for the MAS implementation of SOLERES, since it simplifies the process through a middleware and provides a tool set as support for checking and debugging. Besides, the system manages two kinds of ontologies, both of them written in OWL format:

data and functional. The first type is used to represent the environmental information stored in the documents of the trader. The second one is referred to the trader functionality, that is, the agent tasks.

The trading service represents the main agent in the SOLERES system architecture. As we can see in the Figure 2, basically two mainly components take part in the model: the user or user group represented by an interface agent, and the trading service which has the access to the information.
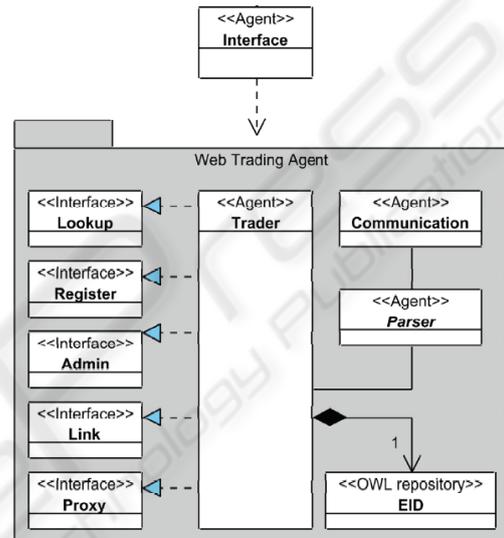


Figure 2: Web Trading Agent architecture.

## 3 IMPLEMENTATION DETAILS

The implementation follows the QS/RR model, in which appear two roles basically. These roles are the **client** and the **server**, having in mind that server will be waiting the reception of query messages sent by clients, solving next them and answering with the resultant information to the clients.

In our system, these roles have been developed by means of two agent objects: "AppletAgentQI" and "TraderAgent" in Figure 3. The first one is created from the class "ClientContainer" and the second one from the class "ServerContainer", being both of them an "AgentContainer" agent class. These containers are created by the launcher classes "QIApplet" and "Trader", respectively. Besides, in our system exist: the "SPARQLParser" (an agent for translating the interface queries to SPARQL), the interface "Lookup" (one interface of the trader) and the EID repository (i.e., the OWL repository containing de metadata information of the system).
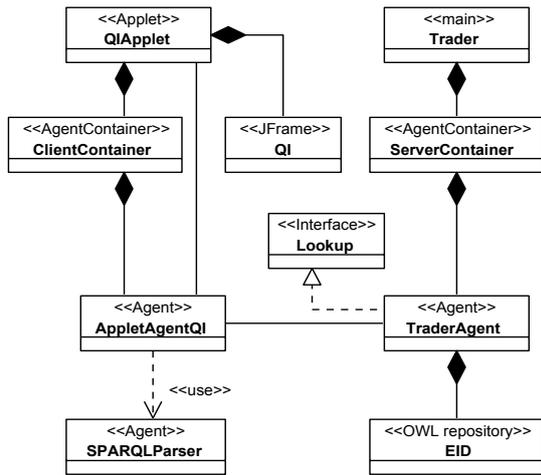
Figure 3: Class diagram of the example.



Figure 4: Server process initialization.



Figure 5: Client process initialization.



Figure 6: JADE Platform after initialization.

To execute the example, it is necessary to follow the next steps. First, we must execute the server process on the computer containing the hosting service. This process will create the agent into an agent container of the JADE platform, which ("Trader-Agent") will initialize its behaviour waiting for response sub-behaviour for the query messages (Figure 4). On the other hand, each user will access to our application through a web page. This web page will initialize an applet that represents the user interface. The applet creates an interface agent and adds its behaviour (Figure 5).

Once we have launched both processes, both agents are created living in a JADE Platform and contained in an independent container (Figure 6). This feature allows us to check the communication between users executing the application in different places, by means of sending and receiving messages from autonomous containers and JADE Platform.

Figure 7 describes the communication query process of the web trading agent, since the client user or the client user group have finished the query in the query user interface until results are shown.

In this implementation example, two main agents appears in the process, i.e., an interface agent ("AppletQIAgent") and a trader agent ("TraderAgent"), using the Lookup ontology to establish the communication. It is necessary to emphasize that the example does not implement the monitoring agent ("IM-IAgent") to simplify the query process.

Query process follows the next steps. The client user builds the query making use of the user interface (steps #1 and #2). Figure 8 shows the query user interface window. We used a tree representation to show the data to be inquired ("Tree Model") and the query ("QueryTree").
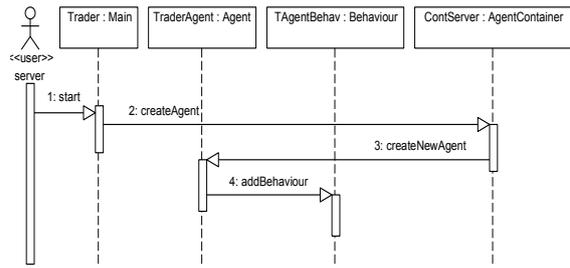
Besides, queries can be manual or supervising. A manual query allows the user to write directly the query into the "Query" section. The supervising one makes possible to build the query selecting the variables from the interface, values and operators. Once the query is built, the user sends it from the user interface to the interface agent (steps #3 and #4). This agent sends the query (written in the interface language) to the SPARQL agent in order to translate it (step #5).

Then, the parser agent converts the query to the SPARQL language (#step 6). The query is stored in a text file in the server directory that contains the trading agent (step #7). Then, the "Lookup" ontology communication message is built (steps #8, #9, #10, #11 and #12).
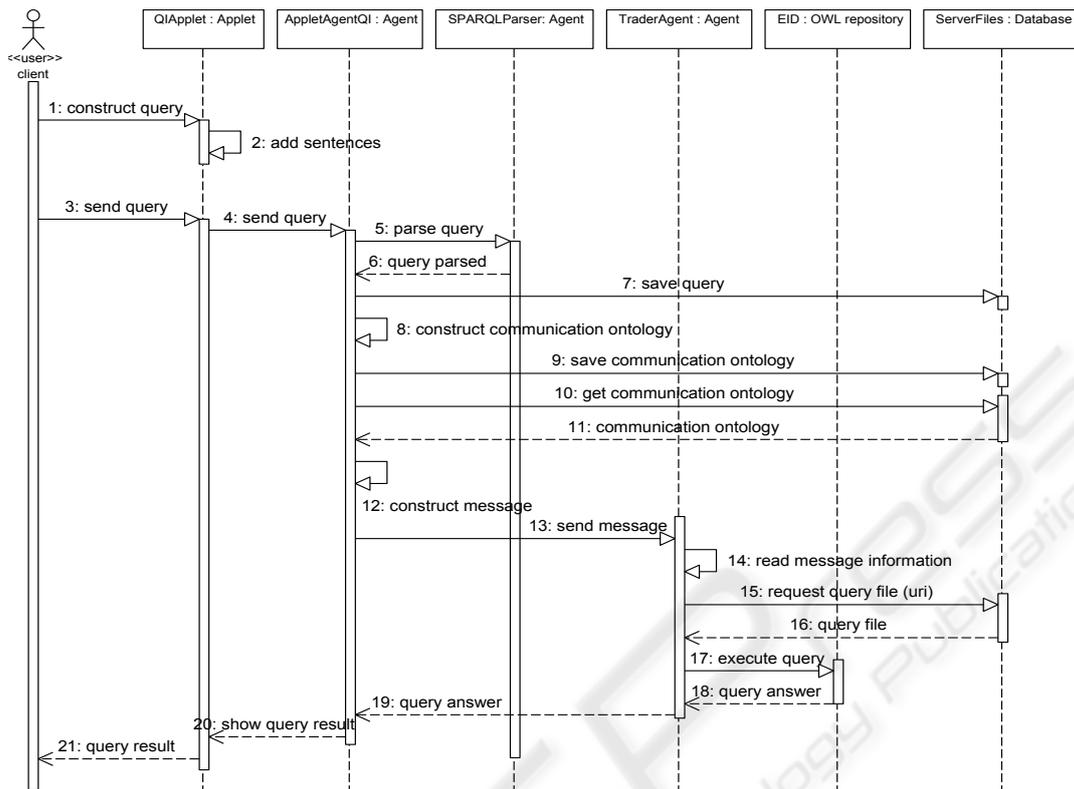
270

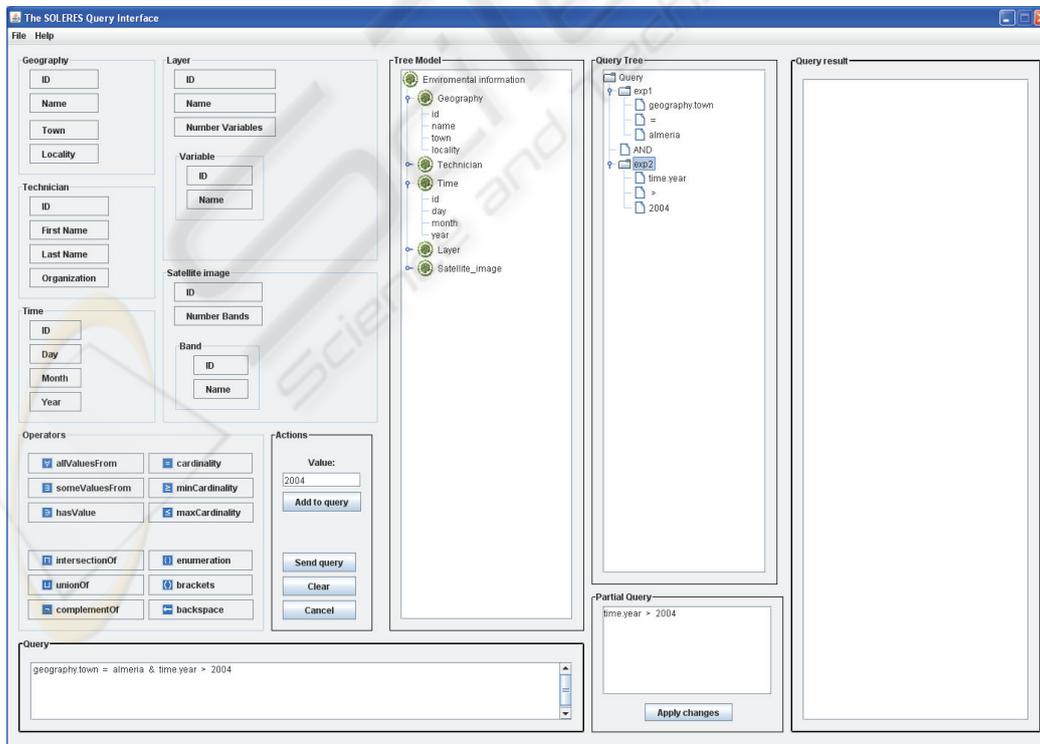Figure 7: Communication process.



Figure 8: The Query User Interface.

For the communication ontology (Lookup) implementation, we used the XML format; The client fills the necessary fields with the correspondent information before send the message (Figure 9). The server (trader) will receive the XML content file and will extract the data from the fields it needs. These operations have been possible making use of the Jdom, Saxon, and other XML libraries.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
- <Lookup>
  - <Concept>
    - <QueryForm>
        <id>12345</id>
        <uri>D:/jade/add-ons/jadeapplets/files/query.txt</uri>
        <type>trading-query</type>
        <source>UI-Agent</source>
        <target>Trader-Agent</target>
      </QueryForm>
    - <PolicySeq>
        <value />
      </PolicySeq>
    - <OfferSeq>
        <id />
        <uri />
      </OfferSeq>
      . . .
```

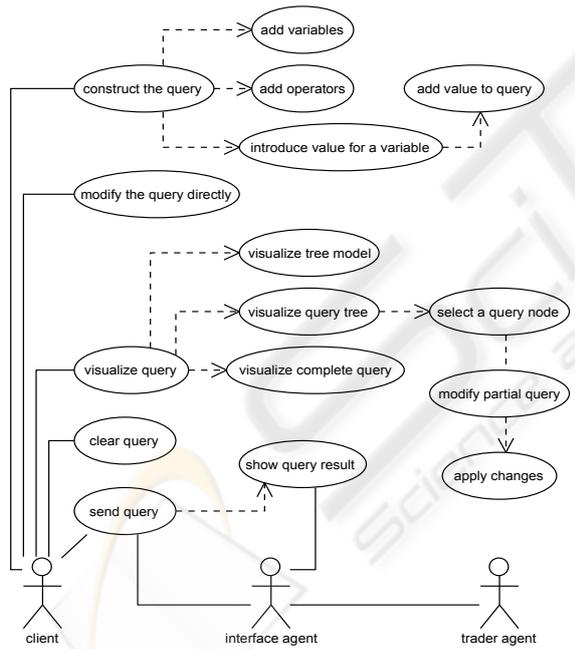Figure 9: The XML Lookup ontology file.



Figure 10: Use Case Diagram.

The next step is to send the message from the interface agent to the trading agent (#13). The trading agent extracts the information from the message: query path, source agent, etc. (#14), and later recover the query file content (#15 and #16) and execute it to the EID repository (#17 and #18). The execution of the SPARQL query to the OWL file is possible by the use of the Jena library. The query result is sent from the trader agent to the interface agent (#19) and it shows this information in the user interface into the "Query Result" field (#20 and #21). Figure 10 summarizes, through a use case diagram, the functional options of the user interface and the connection with the other actors.

We included to the software agents behaviours for autonomous and intelligent functionality in order to solver the task they have been developed for. The interface agent has the task of recovering the query from the user interface and sending it to the agent who can solve it. Later, the query result is obtained and shown to the user. This complex behaviour is divided in sub-behaviours as shown in Figures 11 and 12.

On the other hand, the trader agent has the response of the user messages main task. This task has been divided in four main sub-tasks corresponding with the sub-behaviours we have implemented for the agent (see Figure 12).
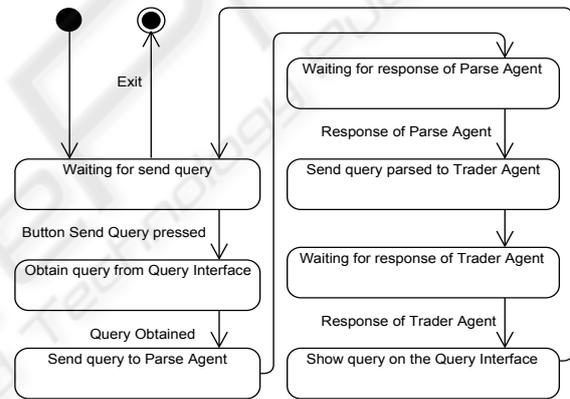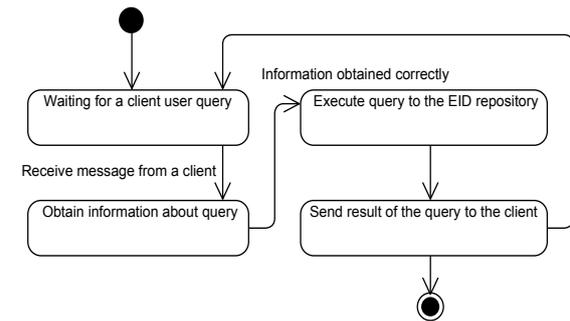


Figure 11: Interface agent behaviour.



Figure 12: Trader agent behaviour.

A simple query example involved in the QS/RR process is shown in the next code. To explain this situation, let's suppose the following query in natural language: "*We wish the classifications made in Almeria that describes lithology variables, specifi-*

*cally about the sand area*". This request is represented in the Query section (Figure 8) as follows:

```
?classif: geo.name = "Almeria" & layer.name
= "lithology" & layer.variable.name = "sand"
```

The SPARQL translation for this example is:

```
PREFIX onto : <http://…/acg/soleres/owl#>
SELECT ?classif
WHERE {
 ?classif onto:classification_shows_geo ?g.
 ?g onto:geography_name "Almeria".
 ?classif onto:classification_uses_layer ?l.
 ?l onto:layer_has_variable ?v.
 ?l onto:layer_name "lithology".
 ?v onto:variable_name "sand".
}
ORDER BY ?year
```

The obtained result by the query process is the classification "classif". At this record, we can find information about a classification, like the identification number, the name of the classification, the layer and the satellite images it uses or the start and end time of the classification. This kind of information is an OWL environmental information that agents of the SOLERES-MAS manage.

## 4   CONCLUSIONS

In this paper we have presented an implementation of the web trading agent methodology for "searching and recovering" process used by SOLERES system.

The web service reviewed is based on the "QuerySearching/RecoveringResponse" model issue. It uses the SPARQL language to represent the queries, and the OWL notation to describe data and functional ontologies.

In this work we also show a MAS structure case study for requiring the information of trading agent from an interface agent. The communication is made through our developed Lookup ontology.

As a future work, we are interested in federation of web-trading agents in MAS by means of the extension of the trader's Link interface. We are also studying the application of this kind of trading agent behavior for evolvable user interface development.

More implementation details of the Web Trading agent, "processes" and "data" ontologies are available at http://www.ual.es/acg/soleres/wt.

## ACKNOWLEDGEMENTS

## REFERENCES

Anghel, C., Salomie, I., 2003. JADE Based solutions for knowledge assessment in eLearning Environments. TILAB & University of Limerick.

Asensio, J., Iribarne, L., Padilla, N., Ayala, R., 2008. Implementing trading agents for adaptative and evolutive COTS components architectures. In *Proceedings of the International Conference on e-Business*. Porto, Portugal, pp. 259-262.

Iribarne, L., Troya, J., Vallecillo, A., 2004. A trading service for COTS components. The Computer Journal 47(3): 342-357.

ISO/IEC DIS 13235-1: IT, 1996. Open Distributed Processing: ODP Trading Function Part 1: Spec.

Lytras, M., 2005. Semantic web and information systems: An agenda based on discourse with community leaders. International Journal of Semantic Web and Information Systems 1(1).

Odell, J., Van Dyke Parunak, H., Bauer, B., 2001. Agent UML: A formalism for specifying multiagent software systems. International Journal of Software Engineering and Knowledge Engineering 11(3): 207-230.

Polleres, A., 2007. From SPARQL to rules (and back). In *Proceedings of the International Conference on World Wide Web (WWW)*, pp. 786-796.

Russ, M., Jones, J., 2006. Knowledge-based strategies and information system technologies: preliminary findings. International Journal of Knowledge and Learning 2(1): 154-179.

Shiyong, D., Xueqiang, F., Jie, Y., Runbo, M., 2007. Research for the Communication and Cooperation Mechanism of Multi-Agent System. In: Electronic Measurement and Instruments, 2007. ICEMI'07.

Tweedale, J., Ichalkaranje, N., Sioutis, C., Jarvis, B., Consoli, A., Phillips-Wren, G., 2007. Innovations in multiagent systems. Journal of Network and Computer Applications 30(3): 1089-1115.