Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 243–249

Information Retrieval Using an Ontological Web-Trading Model

José Andrés Asensio
University of Almería
Applied Computing Group
04120, Almería, Spain
Email: jacortes@ual.es

Nicolás Padilla
University of Almería
Applied Computing Group
04120, Almería, Spain
Email: npadilla@ual.es

Luis Iribarne
University of Almería
Applied Computing Group
04120, Almería, Spain
Email: luis.iribarne@ual.es

***Abstract*—One of the biggest problems facing Web-based Information Systems (WIS) is the complexity of the information searching/retrieval processes, especially the information overload, to distinguish between relevant and irrelevant content. In an attempt to solve this problem, a wide range of techniques based on different areas has been developed and applied to WIS. One of these techniques is the information retrieval. In this paper we described an information retrieval mechanism (only for structured data) with a client/server implementation based on the Query-Searching/Recovering-Response (QS/RR) model by means of a trading model, guided and managed by ontologies. This mechanism is part of SOLERES system, an Environmental Management Information System (EMIS).**

I. Introduction

Nowadays, *Web-based Information Systems* (WIS) have become popular as they favour universal access to the information, helping their users to analyze the information from different viewpoints and support group work, decision-making, etc. However, one of the biggest problems of this kind of systems is the complexity of the information searching/retrieval processes, largely due to the huge amount of information they manage.

Their users depend on web sites, digital libraries, engines and other information searching/retrieval systems [1], [2] to help them in this tedious process and, even so, they deal with an overload of information in which they must distinguish between the relevant and irrelevant content. In an attempt to solve this problem, a wide range of techniques based on different areas has been developed and applied: information retrieval, information filtering, studies on information search behavior, etc. Of all these techniques, we focused on the information retrieval in a client/server model for Web systems. In this context, the term "information retrieval" refers to a set of techniques that satisfy the users' information requirements [3].

The main WIS information retrieval mechanism, based on the client/server model, is *the Query-Searching/Recovering-Response* (QS/RR), showed in Figure 1. On one hand, the term "Query" refers to the whole process of creating and formulating the client's request. The term "Searching" refers to the process of locating the data sources (repositories, data storage or databases, regardless of the model) where the information is found, and the term "Recovering" refers to the process of locating, identifying and selecting the data from these sources. Finally the term "Response" refers to the whole process of formulation, preparation and creation of the response by the server to the client. The "Query-Searching" pair is a process that goes from the client to the server. The "Recovering-Response" pair goes from the server to the client.

Fig. 1. Overview of the QS/RR mechanism.

A solution to QS/RR mechanism is the UDDI (*Universal Description Discovery and Integration*) specification and WSDL (*Web-Services Definition Language*) for SOA (*Service Oriented Architecture*). They are based on client/server implementations for Web systems. Nevertheless, these techniques allow us to respect a *subscribe/publish/response* model (a QS/RR information retrieval approach) for locating WSDL documents (i.e., XML specifications of web-services) and connecting web services in WIS, but not for different types of information (non-WDSL information). Traders [4] are another solution for open and distributed systems that extend the OMA (*Object Management Architecture*) ORB (*Object-Request Broker*) mechanism. From the viewpoint of the *Open Distributed Processing* (ODP), a Trader (also called trading service, trading function or mediator) is the software object that mediates between objects that offer certain capacities or services and other objects that demand their use dynamically. As is shown in Figure 2, objects that offer their services are called "exporters" and provide the Trader with a description (extra-functional aspects) and an interface (functional aspects) of their service, whereas objects that demand these services are called "importers" and ask the Trader for services with certain characteristics. The function of the Trader, therefore, consists of checking the characteristics required in the descriptions of the services

This work has been supported by the EU (FEDER) and the Spanish Ministry MINECO under grant of the TIN2010-15588, and also by the JUNTA DE ANDALUCÍA excellent project TIC-6114. http://acg.ual.es.

offered (stored in a local repository) and indicating the importer the interfaces of the selected services for his interaction with the exporter.

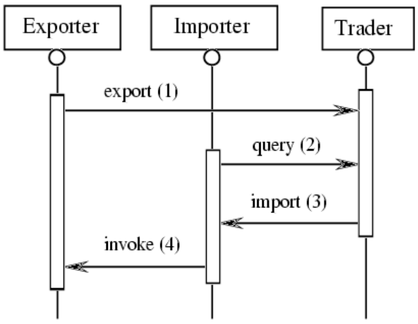

Fig. 2. Roles of the ODP Trader.

There is a large number of studies in which the trading service follows the ODP specification. For instance, in [5] a trading service called DOKTrader is presented, which acts on a federated database system called *Distributed Object Kernel* (DOK). Another example is found in [6]. This study concentrates on the creation of a framework to develop distributed applications for a *Common Open Service Market* (COSM), making use of a *Service Interface Description Language* (SIDL) to describe the services manipulated by the trader. These approaches of the ODP trading implementations have several shortcomings like component interactions, object communications or language description, which have been improved using **ontologies**.

The use of ontologies in trading services has spread, especially in web information services. Ontologies are being used to describe the services offered as well as communication primitives employed by system components. In [7] authors present the design of a market managed by ontologies. Within this system, an ontological communication language is used to represent queries, offers and agreements. Furthermore, in [8], ontologies are used to describe information shared by different system components. To achieve greater operability and autonomous, many systems have chosen to encapsulate the trader object within a software agent. In [9] the MinneTAC agent is described, like a trading agent developed to participate in the Trading Agent Competition (TAC). Through the description of this agent, implementation of a trader as a software agent is shown to maximize benefits from scenarios that require cooperation and negotiation between the trader and the rest of the system components, as well as systems that require communication among various trading objects, making use of ontologies to represent information shared by the agents, whether to describe data and the relationships among variables, as is the case in [10], or defining communication primitives and interaction among agents [11].

In this paper, we propose the Ontological Web Trading (OWT) model that implements a mechanism for solving the complexity of information retrieval in the SOLERES system by means of a trading model for WIS, guided and managed by ontologies. OWT has been implemented in this system as a software agent. SOLERES [12] is an Environmental Management Information System (EMIS) based on satellite images, neural networks, cooperative systems, multi-agent architectures and commercial components. This multi-agent system implements a user Information Retrieval mechanism that implements the QS/RR model and uses the SPARQL query language and the OWL ontology description language to operate. In this system, the ontologies are used in two different contexts: (a) to represent the application domain information itself (**data ontology**), and (b) to request services between agents during their interaction (**service ontologies**). Although a trader agent has five interfaces (i.e., Lookup, Register, Link, Proxy and Admin), this paper discusses only the service and data ontology design features of the Lookup interface, which is used for searching and recovering information. This information should be only structured data. All research work presented here is part of a complete design strategy for *Ontology-Driven Software Engineering* (ODSE) that we are developing in SOLERES.

The remainder of the paper is organized as follows. Section 2 shows the SOLERES system architecture. Section 3 identifies the requirements that an ontological trading service should meet for open and distributed environments as well as the operation models it may carry out. Section 4 describes the Web Trading Agent. Section 5 shows the Lookup ontology used by such agent. We end with some conclusions and prospects for future work in section 6.

II. A Case Study: The SOLERES System

This section presents the main SOLERES system architecture (Figure 3), a spatio-temporal information system for environmental management (an example of EMIS). The general idea of the system is a framework for integrating the disciplines above for "Environmental information" as the application domain, specifically ecology and landscape connectivity. The system has two main subsystems, SOLERES-HCI and SOLERES-KRS. The first is the framework specialized in human-computer interaction. This subsystem is beyond the scope of this article and will not be described. On the other hand, SOLERES-KRS is used to manage environmental information. Examining Figure 3, the `IMI Agent` is like a gateway between the user interface and the rest of the modules, and is responsible for the management of user demands.

Given the magnitude of the information available in the information system, and that this information may be provided by different sources, at different times or even by different people, the environmental information (i.e., the knowledge) can be distributed, consulted, and geographically located in different ambients (i.e., locations, containers, nodes or domains) called Environmental Process Units (`EPU`). Thus the system is formed by a cooperative group of knowledge-based EPUs. These groups operate separately by using an agent to find better solutions (queries on ecological maps).

We accomplished the distributed cooperation of these EPUs by developing a Web Trading Agent (WTA) based on the ODP trader specification and extended to agent behavior.

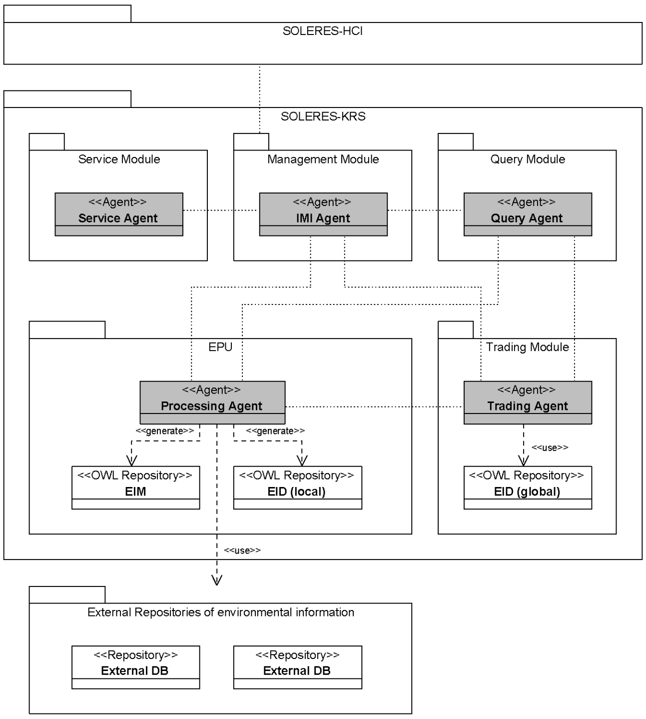

Fig. 3. SOLERES architecture.

Our trading agent mediates between HCI requests and EPU services. EPUs manage two local repositories of environmental information. One of these repositories contains metadata of the information in the domain itself (i.e., basically information related to ecological classifications and satellite images), called *Environmental Information Map* data: EIM documents (`EIM`). This information is extracted from external databases (`External DB` repository in the Figure). The EIM documents are specified by an ontology in OWL [13] (`<<OWL Repository>>`). These EIM documents are the first level of information in SOLERES-KRS.

The second repository contains metadata called Environmental Information metaData, or EID documents (`EID`). These documents contain the most important EIM metadata that could be used by the information retrieval service, and further, incorporate other new metadata necessary for agent management itself. To a certain extent, an EID document represents a "template" with the basic metadata from the EIM document. The EID documents have also been specified by an ontology to accomplish open distributed system requirements. EID documents represent the second level of information in the KRS subsystem. Each EPU keeps its own EID document (or sets of documents) locally and also registers them with the Web Trading Agent (WTA). This way, the WTA has an overall repository of all the EID documents from all EPUs in an ambient and can thereby offer an information search service, as described in the following sections.

III. Requirements and Trading models for OWT

The OWT model developed here is based on the traditional functionality of a trading service, adapted to the management of any type of information (not only on services) through ontologies. Next we will identify a set of requirements necessary for the design of an ontological trading service for open and distributed environments. Later on, we will describe the trading models implemented in our model.

I. Requirements for OWT

For the design of an OWT model, we established a set of properties or requirements that must be met. Table I shows a list of such properties from the ODP standard constraints.

Table I. OWT properties.

Property	Name
#1	Heterogeneous data model
#2	Federation
#3	Composition and adaptation of services
#4	Weak pairing
#5	Usage of heuristics and metrics
#6	Extensible and scalable
#7	"Storage and forwarding" policy
#8	Delegation
#9	Push and pull storage model

Property **#1** (Heterogeneous data model) means that a trading service should be able to work with different data models and platforms and should not be restricted to just one data model. Thus it should be able to mediate with different protocols of access to information and adapt to the evolution of current and future models.

Property **#2** is related to the federation. For the cooperation among traders there should exist a federation among trading services by using different strategies. For instance, a "repository-based" federation strategy allows more than one service to read and write on the same repository, each being unaware of the presence of others inside the federation, and thus allowing a scalable approach.

Current trading services use "one-to-one" pairing according to the clients' demands and availability of services stored in the repositories they can access. Nevertheless, the ontological trading service should also provide "one-to-many" pairing linking (property **#3**), where a client's query should be satisfied through the composition of two or more instances of metadata available in the repositories.

In the trading service processes, especially those working for open systems (like Internet) where methods and operations refer to the services offered, it is essential to consider the kind of pairing imposed (weak or accurate) (property **#4**), as services are chosen randomly, in an unstandarized way and without agreement. That is why a trading service, when getting the list of chosen metadata during the information searching/retrieval processes, should allow using partial pairing to select (from repositories) those metadata that completely adapt to the request for information or just to a part of it.

Property **#5** points out that a trading service should allow users to specify heuristics and metrics functions when searching for metadata, especially for weak pairing. Thus, among other aspects, the trading service would return results organized according to a search conditions.

Property **#6** defines the extensibility and scalability characteristics of a trading service. Here the trading service should consider any piece of information on services (or metadata) such as data of creators, marketing information and so on, and allow users to independently include new pieces of information for metadata they export (register). In turn, it should be able to use the new piece of information as part of the exported metadata.

In view of a client metadata query, a trading service should retrieve a result. Such result can refer either to a list of chosen metadata that satisfy the query or to a “fail” message if there is no search result. In the latter case, we should also be able to require a trading service to compulsorily satisfy the query or, if that is not the case, store it with the information available by that time and postpone the response until one (or several) metadata providers register (export) a metadata that satisfies the client query. This “response-query” behavior is called behavior “on hold” or “storage-and-forwarding” behavior (property **#7**).

Regarding the previous property, a trading service should also allow delegating (property **#8**) (complete or partial) queries to other trading services if the trading service itself were not able to satisfy such queries.

Property #9 defines the push and pull storage models of a trading service. A push model is the model in which exporters directly get in touch with the trading service to register their metadata. An alternative for metadata registration, suitable for trading services, which work in open and distributed environments on a broad scale, consists of making use of a pull storage model. Here, exporters do not get in touch with traders but rather publish metadata on their websites so that the trading services themselves later on “track” the network in search of new metadata.

Now that the requirements demanded of the trading service have been identified, the OWT model operations in the query process can be described.

II. OWT Model Operations

Let us now see how the OWT model operates in the query process, since an object or component makes a query until the results are retrieved.

This model is a trading-based version of the three-level client/server model. It is comprised basically of a series of elements <I,T,D>, each of which intervenes on a different level, depending on the treatment of the query. Level 1 (L1) is like the client side. Queries are generated and dealt with by an interface object (I). Level 3 (L3) is the server side. System data (D) reside on this level. In our case, these are the EIM repositories with the environmental information. Level 2 (L2) is the middleware that enables the source information to be located. This is the level where the trader objects (T) operate. Associated with the trader (T), the EID repositories with the source environmental information metadata (EIM) also reside there. All three objects use the Lookup ontology (described later) to communicate between them. As the premise for their functioning, an interface object must be associated with a trader object. However, a trader object can also be associated with one or more external data sources or resources, in our case, with the environmental source data (which reside in the EPU, as discussed above). This “trader-information source” association arises from the production of environmental information, where each EPU has an associated trader in which a subset (metadata) of environmental information generated by it is registered. On the other hand, each trader can be associated with one or more traders in federations.

In this three-level architecture, three operating scenarios are possible: *Trading Reflection*, *Trading Delegation*, and *Trading Federation*. Figure 4 shows the three levels (L1, L2, L3), where the three basic objects (I,T,D) reside, and the three scenarios permissible in OWT, as described below.

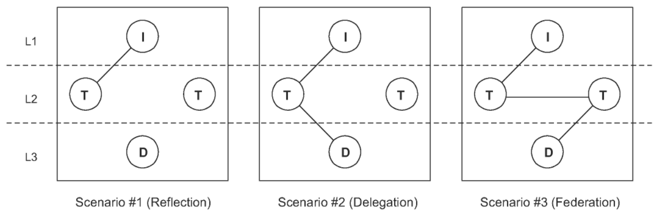

Fig. 4. Operational models of trading.

The **Trading Reflection** scenario in which the query may be solved directly by the trader. The query is generated on the interface and the information can be reached by the metadata that reside in the repository associated with the trader. In this case, the model <I,T> pair intervenes.

The **Trading Delegation** scenario indirectly mediates with the trader. The query is partly resolved by the trader. A query is generated on the interface level that goes on to the trading level (T). The trader locates the data source (or sources) (D), inferring this information from its metadata repository. Therefore, the trader delegates the query to the outside data source (D). In this case, the object series is <I,T,D>.

Finally, the **Trading Federation** scenario is a case in which two or more trader objects are able to federate. As in the cases above, the query remains preset on the interface. This query is passed on to the associated trader object. It can propagate the query to another federated trader object, who locates the external data source (D). In this case the object series intervening is <I,T,T,D>.

For design reasons, the three basic OWT model levels <I,T,D> have been implemented by agents using the JADE platform in the following way. The interface (I) was implemented by means of two agents: the Interface Agent and the IMI Agent. The trading level (T) was implemented by using two other agents: Query Agent and Trading Agent (WTA). The data level (D) was implemented by means of a Resource Agent. From the work perspective presented here, we are interested in the information searching/retrieval processes, so that the explanation concentrates only on the WTA and the Lookup ontology used for it.

IV. Web Trading Agent

This section describes the internal structure of our Trading Agent and some details about its design and implementation. It should be emphasized that this agent, like all SOLERES system agents, was modeled, designed and implemented based on run-time management of the ontologies used. The trader therefore manages two kinds of ontologies, data and service (or process):

(a) The first is related to the ecological information repositories the trader can access. The information is distributed in different OWL repositories on two levels, as described in *Section II-A*. Some of them contain environmental metadata (EIM repositories) and others contain metadata from the first (EID repositories). A trader manages an EID repository.

(b) The second kind of ontology refers to trader functionality, that is, actions it can do and demand from others. In this case, behavior and interaction protocols must also be defined. These definitions set the operating and interaction rules for agents, governing how the functions the trader provides and demands to work (behavior) are used and the order they are called up in (protocols/choreography).

Figure 5 shows a data ontology from an EID repository (described formally in UML). Let us recall that the application domain to be modeled is ecological information (a type of environmental information) on cartographic maps and satellite images. Advanced algorithms based on neuronal networks find correlations between satellite and cartographic information. For the calculation of this correlation, prior treatment of the satellite images and maps is necessary (an image classification, `Classification`).

A cartographic map stores its information in layers (`Layer`), each of which is identified by a set of variables (`Variable`). For instance, we are using cartographic maps classified in 4 layers (climatology, lithology, geomorphology and soils) with over a hundred variables (e.g., scrubland surface, pasture land surface, average rainfall, etc.).

Satellite images work almost the same way. The information is also stored in layers, but here they are called bands. An example of satellite images is the LANDSAT image, which has 7 bands (but no variables stored in this case). Finally, both the cartographic and satellite classifications have geographic information associated (`Geography`), which is made at a given time (`Time`) by a technician or group of technicians (`Technician`).

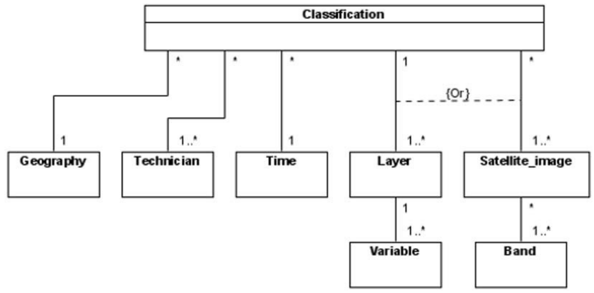

Fig. 5. Ontology of the EID metadata that traders use.

As a complement and formalization of this conceptual model, Table II shows the complete assertions of the eight ontology entities expressed in OCL (*Object Constraint Language*). As an example, we can describe two assertions. The assertion #2 for the `Classification` entity shows it has two required properties, `Classification_id` and `Classification_name`. This entity (classification) is related: (i) either with at least one `Layer` or `Satellite_image` entity (never with both entities simultaneously) through the `classification_shows_layer` or `classification_uses_satellite_image` relationships, respectively; (ii) always with one `Geography` entitythrough the `classification_shows_geography` relationship; (iii) with at least one `Technician` entity through `classification_is_made_by_technician`; and (iv) also with two `Time` entities, `classification_starts_time` and `classification_ends_time`. Analogously, the assertion #4 for the `Layer` entity indicates that it has two

TABLE II. EID ONTOLOGY ASSERTIONS IN OCL.

#	Entity	Assertions
#1	Band	(band_id **exactly** 1) **and** (band_is_shown_by_satellite_image **min** 0) and (band_name **exactly** 1)
#2	Classification	(classification_id **exactly** 1) **and** ((classification_shows_layer **min** 1) **or** (classification_ uses_satellite_image **min** 1)) **and** (classification_ends_time **exactly** 1) **and** (classification_ is_made_by_technician **min** 1) **and** (classification_name **exactly** 1) **and** (classification_ shows_geography **exactly** 1) **and** (classification_starts_time **exactly** 1)
#3	Geography	(geography_id **exactly** 1) **and** (geography_is_shown_by_classification **min** 0) **and** (geography_ locality **exactly** 1) **and** (geography_name **exactly** 1) **and** (geography_town **exactly** 1)
#4	Layer	(layer_id **exactly** 1) **and** (layer_has_variable **min** 1) **and** (layer_is_shown_by_classification **exactly** 1) **and** (layer_name **exactly** 1) **and** (layer_observations **max** 1)
#5	Satellite_image	(satellite_image_id **exactly** 1) **and** (satellite_image_is_used_by_classification **min** 0) **and** (satellite_image_shows_band **min** 1)
#6	Technician	(technician_id **exactly** 1) **and** (technician_first_name **exactly** 1) **and** (technician_last_name **exactly** 1) **and** (technician_makes_classification **min** 0) **and** (technician_organization **max** 1)
#7	Time	(time_id **exactly** 1) **and** (time_day **exactly** 1) **and** (time_month **exactly** 1) **and** (time_year **exactly** 1) **and** (time_is_started_by_classification **min** 0)
#8	Variable	(variable_id **exactly** 1) **and** (variable_name **exactly** 1) **and** (variable_is_had_by_layer **exactly** 1)

required properties, `layer_id` and `layer_name`, as well as another optional, `layer_observations`, and it is always related with `layer_is_shown_by_classification` and, at least with one `Variable` through `layer_has_variable`.

The functionality of our trader [14], [15] is divided into three clearly differentiated components (see Figure 6): (a) a component that manages the agent-communication mechanism (**Communication**); (b) a parser that codes and decodes the trading ontology-based messages exchanged (**Parser**); and (c) trading itself (**Trader**).

The third component is inspired by the ODP specification, which indicates how offers and demands must be implemented among objects in a distributed environment and proposes grouping all the different functionalities that a trader may include. Although the standard specifies five trader interfaces (i.e., Lookup, Register, Admin, Link and Proxy), its specification does not demand a trader to implement these five interfaces to work. In fact, we have only developed ontologies for the Lookup, Register, Admin and Link interfaces, but none has been implemented for the last one yet. The Lookup interface offers the search-information in a repository under certain query criteria. The Register interface enables objects in this repository to be inserted, modified and deleted. The Admin interface can modify the main parameters of the trader configuration, and finally, the Link interface makes trading agent federation possible.

As previously explained, this paper focuses on identifying and explaining how ontologies appear and intervene in the Web Trading Agent. Of the interfaces implemented, we only explain here the Lookup interface works, because it takes part in the search, which is the primary subject of this article.

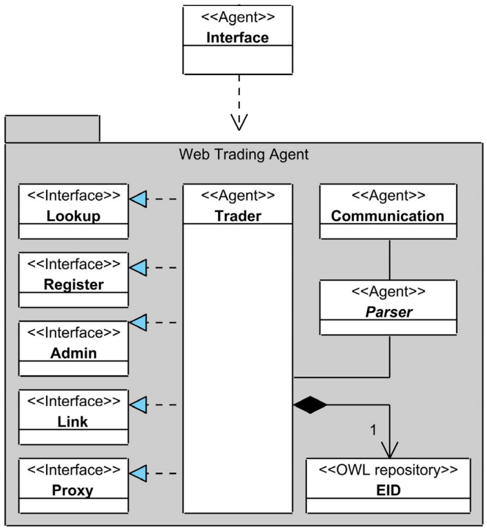

Fig. 6. Web Trading Agent view.

V. THE LOOKUP ONTOLOGY IN OWT

The Lookup ontology (Figure 7) is used between system objects. The trader uses the `Query` action and the `QueryForm` concept. The `QueryForm` concept expresses the query in a specific language, whose properties, among others are: an `id` (a query identifier) and an `uri` (reference to the file where the query is stored). In addition, there could be a set of query policies (`Policy`) through the `PolicySeq` concept, and each "policy" is represented by means of a tuple (name, value). For instance, some of the tuples implemented are:

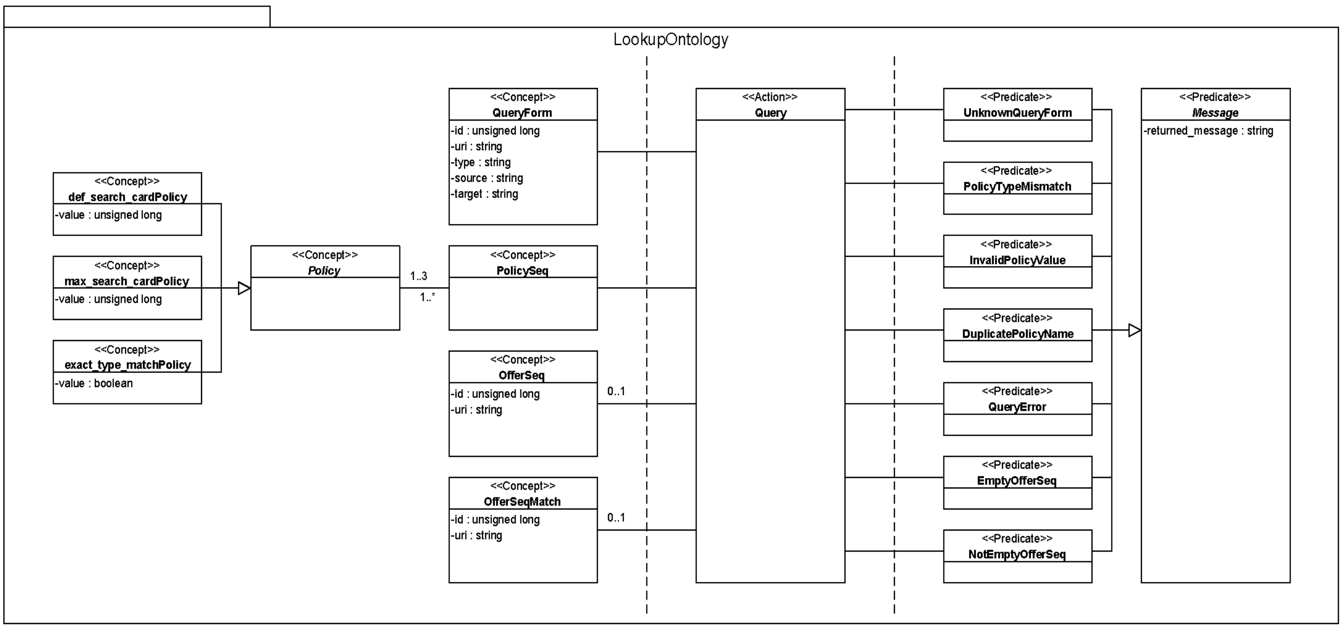

Fig. 7. Lookup Ontology metamodel expressed in UML.

def_search_cardPolicy or max_search_cardPolicy, indicating the number of records to be located by default, and the maximum number of records to be located in the query, respectively. It is possible some exceptions.

Thus, UnknownQueryForm indicates that the query cannot be answered because the file specified in the uri is not accessible; PolicyTypeMismatch indicates that the type of value specified is not appropriate for the Policy; InvalidPolicyValue indicates that the Policy value specified is not within the permissible value range for that Policy; DuplicatePolicyName indicates that more than one value for the same Policy has been specified in the PolicySeq; and QueryError indicates that an error has occurred during the query. If there is no exception and the query is successfully executed, either the EmptyOfferSeq predicate is used when no record is returned by the query, or the NotEmptyOfferSeq predicate, when it is. This, in turn, uses the OfferSeq concept to represent the set of records located in the query, the properties of which are the query "id" and the file "uri" where the found records are stored.

VI. Conclusion

Today, web-based EMIS greatly facilitate information search and retrieval, favoring user cooperation and decision-making. Their design requires the use of standardized methods and techniques that provide a common vocabulary to represent the knowledge in the system and a capability for mediation to allow interaction (communication, negotiation, coordination, etc.) of its components. Ontologies are able to provide that shared vocabulary, and trading systems can improve the interoperability of open and distributed system.

The present paper shows how traditional traders, properly extended to operate in WIS, are a good solution for information retrieval. For that we have introduced Ontological Web-Trading (OWT), an extension of the traditional ODP trading service to support ontological information retrieval issues on Web-based EMIS, as is the case of the SOLERES system.

Future work will focus on the implementation of SOLERES-HCI (Human-Computer Interaction). This subsystem of our EMIS is defined by means of the Computer Supported Cooperative Work (CSCW) paradigm [16] and implemented by using an innovative technology of intelligent agents and multi-agent architectures. Furthermore, we are working on this subsystem and studying how to decompose the user tasks into actions that will have to be performed by the SOLERES-KRS subsystem for retrieval of the information requested and the ontology mapping problems involved.

Finally, we would like to study, develop and incorporate new evaluation and validation techniques, such as measuring the precision of data returned to queries, response time in executing the query, usability, etc.

References

[1] Goh, D., Foo, S., 2007. Social information retrieval systems: Emerging technologies and applications for searching the web effectively. Idea Group Reference.
[2] Gama, J., May, M., 2011. Ubiquitous Knowledge Discovery. Intell. Data Anal. 15 (1), 1.
[3] Ramos, A. C., Gensel, J., Villanova-Oliver, M., and Martin, H. (2005). Adapted information retrieval in web information systems using PUMAS. In AOIS, pages 243-258.
[4] Trader, I., 1996. ISO/IEC DIS 13235-1: IT–Open Distributed Processing– ODP Trading Function–Part 1: Specification.
[5] Craske G, Tari Z, Kumar K. R. (1999) DOK-Trader: A CORBA Persistent Trader with Query Routing Facilities. Proc. of the Int. Symp. on Distributed Objects and Applications, pp. 230
[6] Merz M, Müller K, Lamersdorf W. (1994) Service Trading and Mediation in Distributed Computing Systems. Conf. on Distributed Computing Systems, pp. 450–450.
[7] Busse S, Kutsche R. D., Leser U, H Weber (1999) Federated Information Systems: Concepts, Terminology and Architectures. Technical Report 99-9, Technical University of Berlin.
[8] Lamparter S, Schnizler B. (2006) Trading Services in Ontology-driven Markets. Proc. of the 2006 ACM Symp. on Applied Computing, pp. 1679–1683.
[9] Tsai WT, Huang Q, Xu J, Chen Y, Paul R. (2007) Ontology-based Dynamic Process Collaboration in Service-Oriented Architecture. Proc. Service-Oriented Comp. & App., pp. 39–46
[10] Collins J, Ketter W, Gini M. (2009) Flexible decision control in an autonomous trading agent. Electronic Commerce Research and Applications.
[11] Ziming Z, Liyi Z (2007) An integrated approach for developing e-commerce system. Proc. Of the Wireless Communications, Networkig and Mobile Computing, pp. 3596–3599.
[12] Iribarne, L., Padilla, N., Asensio, J., Criado, J., Ayala, R., Almendros, J., Menenti, M., 2011. An Open-Environmental Ontology Modeling. IEEE Trans. on SMC Part A 41 (4), 730–745.
[13] Padilla, N., Iribarne, L., Asensio, J., Muñoz, F., Ayala, R., 2008. Modelling an Environmental Knowledge-Representation System. Lecture Notes in Computer Science (LNCS), 5288: 70-78. Springer. DOI: 10.1007/978-3-540-87781-3_8.
[14] Asensio, J., Iribarne, L., Padilla, N., Ayala, R., 2008. Implementing trading agents for adaptable and evolutive COTS components architectures. In: Proceedings of the International Conference on e-Business, Porto, Portugal. pp. 259–262.
[15] Iribarne, L., Troya, J., Vallecillo, A., 2004. A trading service for COTS components. The Computer Journal 47 (3), 342–357.
[16] Pendharkar, P., 2007. The theory and experiments of designing cooperative intelligent systems. Decision Support Systems 43 (3), 1014–1030.