Knowledge Sharing by Information Retrieval in the Semantic Web

Neyir Sevilmis[1], André Stork[1,†], Tim Smithers[3], Jorge Posada[3], Massimiliano Pianciamore[2], Rui Castro[4], Ivan Jimenez[3], Gorka Marcos[3], Marco Mauri[2], Paolo Selvini[2], Bruno Thelen[5], and Vincenzo Zecchino[6]

[1] Fraunhofer Institut für Graphische Datenverarbeitung, Darmstadt
[2] CEFRIEL, Milan
[3] VICOMTech, Donostia / San Sebastián
[4] Centro de Computação Gráfica, Guimarães
[5] Schenck Pegasus GmbH, Darmstadt
[6] Italdesign – Giugiaro SpA, Moncalieri (Torino)
[†]`ist-wide@igd.fhg.de`

Abstract. Effective and efficient information retrieval, knowledge sharing and combining has become an essential part of more and more professional tasks and work flows in different kinds of projects. Our aim is to investigate the use of emerging Semantic Web technologies, tools, and standards in the support of effective information retrieval in real multi-disciplinary activities, such as innovative product design. This paper presents an approach to knowledge sharing and information support that has been developed and adopted, the information system architecture that is being developed to test both this approach, and the Semantic Web techniques that are used in its implementation, some early results, and a discussion of related work in information systems and Semantic Web techniques and tools.

1 Introduction

Unquestionable, the internet is developing towards the Semantic Web. Semantic Web technology promises to improve on one of the main usages of the internet: knowledge exchange via information retrieval. But how shall a Semantic Web-based system look like to best support different users in retrieving information and knowledge generated by others in the Semantic Web? This was the deriving question, motivating us to design, implement, and test an approach that explores Semantic Web technologies for improving on today's limited search and retrieval possibilities on the internet. The developments have been done in the context of the product development process within the car industry. The domain knowledge, use and test cases have been developed with real data and real users from two companies, namely ItalDesign Giugiaro SpA and Schenck Pegasus GmbH.

[†] Corresponding Author.

The main requirements imposed by the scenario - typical for the Semantic Web - have been to support different kinds of users accessing various heterogeneous information sources in a semantic way. These requirements entailed a number of secondary questions: How to support users in developing complex queries in a terminology that each type of user is familiar with? How to process these queries so they can be ‘understood’ by different information sources? How to deal with complex queries? What to do with the results from heterogeneous sources? How to present the results in a meaningful way? And from the technological point of view: What kinds of Semantic Web technologies are best suited to answer those questions? How to they perform with real world ontologies and data? Are the current tools appropriate for end-users to model their domain?

This paper tries to answer to those questions by introducing an architecture and describing its implementation based on the experience we gathered in two rounds of user testing. Although there have been comparable attempts to approach the information management issue in heterogeneous environments [15, 16], to our knowledge none of them comprises both: the full scope of the approach that we introduce here and the use of latest Semantic Web technology and tools, e.g. OWL and OWL reasoners such as RACER.

The paper is structured as follows: first the basic concepts are described. Secondly, the role of the components of the architecture is overviewed. The main part of the paper is dedicated to the semantic information retrieval process, made up by seven steps: the graphical interactive user query development, semantic query processing using a domain ontology, planning distributed query execution on heterogeneous information sources, the retrieval from Semantic Information Sources, the result collection and adaptation, the result preparation process of semantically enriched results, and finally the domain knowledge enhanced result presentation.

Afterwards, based on our experience we point out further research needs from a holistic point of view, here we see another contribution of the current paper to the Semantic Web community.

2 Concepts

In this section, we introduce the basic concepts and items/entities involved in our approach. Some of them are related with the architecture, others with the way we are modelling the domain and how we distinguish amongst the role of the components of the system whereas other are definitions useful in describing the various kinds of information and information structures involved.

2.1 3-Layered Architecture

The basic assumption that drove the conception of the architecture is that the near future of the Semantic Web will be determined by a mixture of information sources with various levels of semantic richness in the internet and intranet. Those information sources basically form the lowest level of the architecture – the Content Level (see Figure 1).

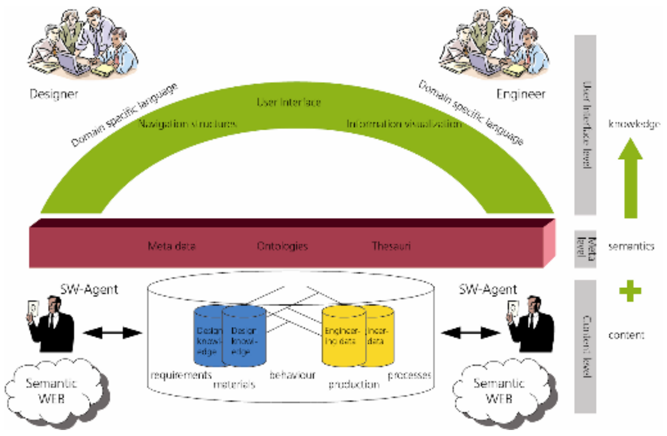

Fig. 1. 3-layered architecture

The second assumption is that domain knowledge and process knowledge shall reside in central components doing the semantic processing of queries and results – the Meta Level.

And thirdly, users of different disciplines along the domain - in our case stylists, designers, engineers involved in the product design process of cars – shall access information and share knowledge via different instances of a User Interface that communicates with the Meta Level and presents retrieved results.

These assumptions immediately entail that the modelling of information sources and the modelling of the domain ontology is performed independently, giving rise to the need of terminology mapping and query adaptation.

2.2 Semantic Information Sources

As already mentioned, we believe that for the near future the developing Semantic Web will provide information sources with different levels of semantic richness.

But, how an ideal Semantic Information Source shall look like? And what kind of functionality shall it contain or provide? Plus what kind of information shall a Semantic Information Source return to enable semantic reasoning?

From a data-centric view, for us a Semantic Information Source contains the following layers: Schema, Annotation and Content.

On the top-most layer a conceptual data schema describes the content that is stored by abstract entities and their relations. This schema appears as a low-level ontology: the provider ontology of the Semantic Information Source. In the middle layer the abstract entities are instantiated in interlinked metadata annotation objects which, in turn, refer to the actual content items on the bottom layer. The content can be of many kinds, e.g. multi-media documents such as pictures, text, 3d models. These content items (instances) are what the user is interested in to retrieve. All the metadata on the schema and annotation level are being used to semantically describe the content and allow for precise and accurate retrieval. The metadata objects in the middle layer appear as instances of classes while the content items appear as references to URL-accessible stores or lower-level database access components.

Note that a Semantic Information Source does not contain a full domain ontology nor does it represent a knowledge base from our point of view. Instead, the Semantic Information Source models the ‘aboutness’ of documents/information contained in the lowest level. The schema only contains what is needed to appropriately describe the kinds of documents/information contained in the lowest level and about which real world concepts they are talking about.

From a functional point of view a Semantic Information Source should be able to process a query posed in a standard format – we are using RQL [3] in the current implementation. In this context processing comprises, mapping it in a syntactic and semantic way.

Furthermore, it shall provide the results in a standard form, preferably enriched by semantic information/context for further semantic processing, e.g. reasoning on the results for filtering and ranking them before showing them to the user. We are using the W3C suggestion for RDF result representation returning not only the results but also structured context information (for details see below) [12].

2.3 User Queries and System Queries

We aim at allowing the user to formulate his query in an as natural as possible way (ease-of-use) but also as precise as possible (quality of results).

To face the users with standard query languages such as RQL, SQL, etc. is certainly not the best approach in terms of usability. Thus we decided to distinguish between User Queries (UQ) and System Queries (SQ). User Queries are input by the user whereas System Queries are derived from User Queries using domain knowledge by the Meta Level.

3 System Architecture

We designed and implemented a distributed system architecture which is divided in to four basic blocks: the User Interface, the Meta Level, the Agency and the Content Level. Each of these is implemented as independent subsystems, the Agency - a multi agent system - is used to “glue” them together, as shown in Figure 2.

3.1 Role of the User Interface

The User Interface (UI) provides a graphic front end to the user and supports the incremental development of user queries in an alphanumeric or graphic way. By easy to use drag and drop operations the user can successively build up his query. This interactive and incremental query development process is supported by the Meta Level and the domain knowledge contained in the Meta Level. Furthermore, the UI presents the returned results and their relationships (semantics) in a graph-based structure. This graph structure can be navigated by the user in order to explore the returned results and their metadata. Based on the returned graph structure and the corresponding metadata the current user query can be refined or a new one can be developed.

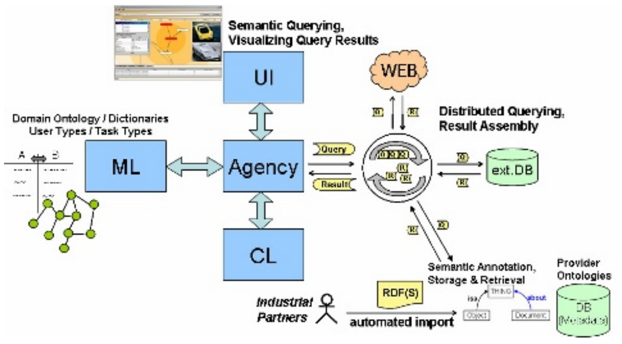
Semantic Querying, Visualizing Query Results
UI
WEB
Distributed Querying, Result Assembly
Domain Ontology / Dictionaries User Types / Task Types
ML
Agency
Query
Result
ext.DB
CL
Semantic Annotation, Storage & Retrieval
Provider Ontologies
RDF(S)
Industrial Partners
automated import
DB (Metadata)

Fig. 2. Distributed system architecture

3.2 Role of the ML

The main purpose of the ML is the semantic processing of user queries into system queries and the semantic processing of the returned results. To do this, the ML uses a domain ontology (for car design), together with a Task Type ontology (knowledge about the different tasks carried out in the domain), User Type ontology (knowledge about the profile of the different users in the domain), and dictionaries of description terms and User Type terms. All these different kinds of knowledge are used to produce the system queries that are then passed to the Agency. The returned results undergo a similar semantic processing as the user queries. They are semantically processed in order to associate them with the appropriate concepts in the domain ontology and finally display them in a meaningful graph structure by the UI.

3.3 Role of the Agency

The Agency subsystem glues the other components together by allowing their interoperation. It further identifies and locates information sources in the Content Level to which the system queries can be sent to produce effective returns, by addressing and managing issues like distribution and heterogeneity in them. The Agency also provides the system's gateway to the Web, which is also considered part of the Content Level. Essentially, Web sites and Web search engines are treated as weakly structured information sources. Besides the planning and execution of queries, the Agency's responsibilities are also to collect and to transform the results of the heterogeneous information sources into a common result format on which the ML is performing semantic processing and reasoning.

3.4 Role of the CL

The Content Level (CL) consists of different information sources (RDF sources [5, 6], ASAM/ODS [4], relational databases and the web) that vary in their semantic richness. Since those information sources might have a different structure than the ML domain ontology, one of the tasks of the CL is to adapt the system queries to the query language understood by the information sources (syntactically, terminologically, and structurally). The main purpose of the CL is to answer precisely the system queries in a quick way.

4 The Process of Information Retrieval

We understand the information retrieval process as a kind of design task by firstly recognizing the difference between user stating needs and forming well specified requirements, and secondly properly supporting the incremental development of a complete and consistent requirements specification (search specification, in this case), and the re-use of the knowledge generated in this (sub)process to effectively support the subsequent steps in the process that concludes in a useful set of search results. According to the system architecture presented in Figure 2, the process of information retrieval is composed of seven steps that are presented in Figure 3.

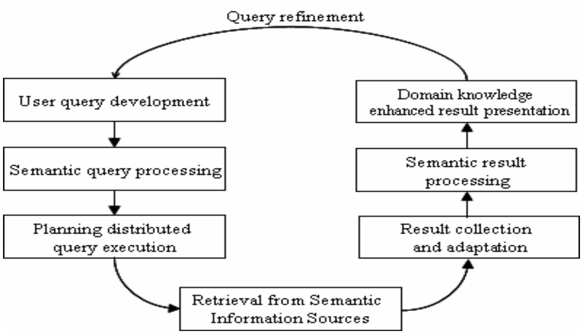
Query refinement
User query development
Semantic query processing
Planning distributed query execution
Retrieval from Semantic Information Sources
Result collection and adaptation
Semantic result processing
Domain knowledge enhanced result presentation

Fig. 3. Seven steps of the information retrieval process

Based on the experienced gathered when developing the system, we are convinced that these are – at least – the steps needed for improved information retrieval on the Semantic Web. According to this, each step of the information retrieval process will be explained in the following sections.

4.1 User Query Development

The UI provides text-based and graphic-based support to specify queries. The graphical version uses domain knowledge (from the ML, suitably selected and presented using the User Type and Task Type specifications) to offer the user a “drag and drop” way of building correct queries. Users can use a combination of both text input and graphical selection to form a query as shown in Figure 4. This is then checked against a BNF grammar [2], for correctness, and passed to the ML. The ML then processes the user query, using its domain ontology, User Type Dictionary, and personal user dictionary, to discover what other concepts it has that are related to the concepts in the UQ.

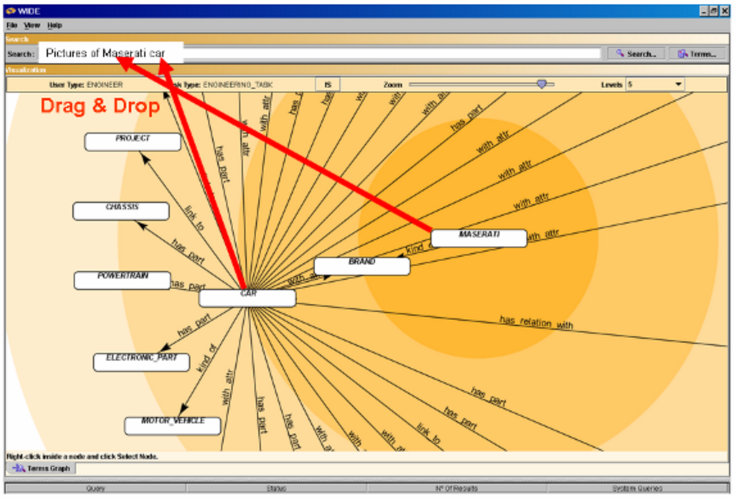

Fig. 4. Graphical interactive user query development

These further domain ontology concepts, and the ways they are related to the user query concepts, are then returned to the UI as an ontology fragment that represents the user query and its immediate conceptual context, where it is graphically presented to the user. This ontology fragment, or query structure, as it is called, supports further navigation of the concepts and properties present, allowing the user to further extend the fragment by including further concepts along selected relations. In this way, a user is able to see how the system understands his or her query, and is supported in further exploring around it, to see how it might be changed, adapted, or extended, to form a more effective query: more precise and/or more complete.

4.2 Semantic Query Processing

When the ML receives the user query from the UI via the Agency, this query is well formed input since it has been successfully parsed by the UI. Thus, this is the starting point of the semantic query processing.

The first task that must be done is to check again the stylized input, but at this occasion from the semantic point of view. The ML uses its knowledge about the domain, but also its knowledge of the User Type and the task context the query is being expressed on, to find out more information about the query.

First of all, the ML uses the dictionaries together with the user profile in order to translate the query into the internal terminology. Next, it tries to find out which of the words that have been classified by the BNF parser as “terms” are known in the domain and which not.

Once the ML has translated the user query into internal terminology, the parser generates an AST (Abstract Syntax Tree) which includes useful information that allows the ML to know the terms within the context of the BNF grammar. For instance, the ML automatically knows that “OF” is a connector and that “OR” is a logical binary operator that links two items. Thus, the ML is able to classify the relevance of each term depending on its BNF grammar nature. Once the ML acquired this knowledge, it can go through this tree and focusing on the known terms in order to try to find out the existing relationships among them. This is possible due to an inference process of the ML domain ontology. For instance, this process allows the ML to relate two concepts basing on subsumption or indirect relationships.

Once this process has been carried out, the ML has a graph view of the user input, which has been enriched with intermediate concepts and/or inter-concept relationships.

After this first query has been built, the ML, basing on the task context the query is related to, tries to create complementary queries by the expansion of some concepts. For instance, if the user is looking for emission standards in Europe, and the task context is wide enough in order to cover also the test cycles, the system will generate another system query that will retrieve test cycles in Europe.

Once the ML has inferred the sub-domain related to the task context and has created several queries, these queries are formatted in RQL, and then passed to the Agency.

For example, a concept designer might start by asking for

user query 1: Photographs of Maserati cars

In response, the UI (with support from the ML) would show that it understands *photograph* to be a kind of picture, where *drawing*, *image*, and *sketch* are other kinds of *picture* concepts. As a result, the user might then change the query to

user query 2: Pictures of Maserati cars

to be more inclusive of other possible kinds of pictures. This is then transformed by the ML in to the following internal form

user query 3: PICTURE ABOUT MASERATI CAR

where *picture* is a known document type, *about* is the term used to connect the document type to the concept, and *Maserati* and *car* are understood as two terms forming an attribute value qualifying phrase. The ML then expands this user query, based upon its knowledge that *Maserati* is the name of an individual of the concept *brand*, and that *brand* is defined as the range of a *has_property*, whose domain is *car*. The resulting expanded system query thus expressed as an RQL query then looks like:

```
SELECT pt, mc
FROM {pt:$pt} @p {mc:$mc},{rc1} @w_a1{c1:$c1},
     {rc2} @w_v1 {v1:Literal}
WHERE @p = “has_info_about”AND
     ($p1 = “PICTURE”) AND
     $mc = “CAR” AND
     mc = rc1 AND
     @w_a1 = “with_attr” AND
     $c1 = “BRAND” AND
     c1 = rc2 AND
     @w_v1 = “with_value” AND
     v1 = “MASERATI”
```

The RQL system query says: select all the pictures and all the cars where picture is a presentation type that has info about the cars and the cars have an attribute named brand, whose value is Maserati.

4.3 Planning Distributed Query Execution

As soon as the Agency subsystem receives the system queries produced by the ML, it sends them to the various information sources, in order to proceed with the search process. The decision about how to distribute the various queries over the available sources is referred to as query execution planning and can be carried out by analyzing the structure of each query. Concerning the structure of a system query expressed in the RQL language, it can easily be seen that the FROM clause can be interpreted as the navigation path inside a proper ontology of concepts bound to one another by means of suitable relationships. As such, it appears evident that one single source might not be able to address the whole system query in its entirety and therefore it is forced to focus on a sub part of it. When this happens, the system query being processed is called a *complex query*, since it can be both logically and physically split up in as many sub-queries as there are sub parts entirely addressable by a single source. We call these latter ones *simple queries*. Using an internal representation in which system queries are made up of conjunctive predicates (reflecting the RDF triples embedded in the RQL queries) the Agency broadcasts the complete sequence of predicates for each query to all the information sources available. The management of the interdependencies between the simple queries is left to the information sources themselves, which may or may not be enabled to do that. If not, they simply consider each complex query as a simple query and try to execute it entirely, possibly returning an empty result in case they cannot understand it or have no data for it. Otherwise, each source decides which simple queries it can address, by looking at the set of mapping rules it has, in order to bridge the semantic and structure gaps between the conceptualization implied in the queries and their own (see below). If no rules exist that have a match with a simple query, then that query is not considered for local execution and the source waits for its results to come from other sources. This is done by means of mobile software agents carrying queries and results back and forth between the core system and the remote sources. Mobile agents are particularly well suited for repetitive tasks like information retrieval, as they realize an asynchronous computing model that adapts well to varying network conditions. In particular, mobile agents improve the fault-tolerance with respect to network communication problems by providing a disconnected operation model: interactions and communication occur at well defined instants, which also generate a more efficient, burst-like, kind of network traffic.

In order to contact the remote sites where to apply the various queries, agents may choose two options: either they move there or they spawn children agents, which are sent on their behalf. Which option to choose depends on factors like the current network traffic, the dimension of the search job in terms of how big is the query and how many information sources must be contacted. Currently, agents choose the first option (move) when there is only one information source to be contacted, otherwise they parallelize the process by choosing the second alternative.

Once all the results are produced, they are assembled and sent back to the system, for final processing and visualization. Result assembling is realized either by pure concatenation of the return fragments obtained, in case the source is not enabled to manage the complex queries. Otherwise results undergo an interdependency-solving process by which join conditions specified in the complex query and binding its constituent simple queries are taken into account.

Query execution at the various information sources requires that all the heterogeneity issues be dealt with, at the syntactic, structural and semantic levels: information pieces, in fact, are likely to be distributed and partially replicated on different repositories, often built using different technologies and modeling techniques. To overcome the syntactic differences a proper algorithm was devised, making use of thesauri on both (on the system and at each source) sides to find matchings and rewrite terms. As for structural differences, they are dealt with by identifying recurrent structural patterns in queries that can be easily rewritten into others. Semantic heterogeneity, on the other hand, has to do with the interpretation given to the stored data along with any relationship binding them. For a machine to be able to communicate and understand what a single piece of information is about, a model describing the semantics of the information has to be provided. Such model is represented by the Provider Ontologies, that are either already available, for semantically rich sources, or can otherwise be created. Semantic mapping, in this respect, has to do with the rewriting of queries, by employing rules associating whole fragments in the modeling of the domain ontology to fragments suitable for the various provider ontologies.

The information sources that we addressed belong to one out of three kinds: an object-relational one served by an ASAM/ODS server, the Google search engine (as an example of an unstructured source), which is accessed by using its Web Services interface and a set of plain relational databases, each one provided with an RDF(S) (provider) ontology that describes its contents and that has a mapping to the (central) domain ontology. To manage each query, execute it and apply the necessary transformations, a *provider agent* was created, whose tasks are to:

- receive and manage the software agents
- transform the incoming RQL query and adapt it to the internal language of the source
- manage the results by formatting them in a suitable and common format to be sent back to the system.

In particular, the latter two points depend from the specific details of the underlying source, even though the interface provided to the outside world should remain unchanged.

4.4 Retrieval from Semantic Information Sources

Our Semantic Information Sources, as described in section 2.2, provide a RQL/RDF(S) interface. The input to such an information source is a RQL query which - from the schema layer point of view - asks for instances of concepts on which there are imposed conditions (in the WHERE clause of RQL queries). The result that is being produced for a system query is a RDF graph fragment which contains RDF instances required answering the RQL query together with their attributes and direct references to other RDF instances. Within our Semantic Information Source the Sesame RQL engine [9] is used to access the RDF(S) store.

Each RQL query undergoes a 2-step query process. In the first step only the RDF resources are returned that form the core answer to the system query. In the second step, the returned RDF resources are semantically enriched by their metadata to be presented and visualized by the UI. Concerning the metadata, we distinguish between properties and context information of a returned RDF instance. The properties are describing the returned RDF instance itself and shall provide the user additional information. For example, an instance of the concept *car* has the properties *brand* and *segment*. It is possible that in the results several concepts (context information) are present that are not strictly related to what originally contained in the system query coming from the ML. The context information is being described by the neighbouring concepts, because we think that they provide meaningful information to reason on the results by the ML. The neighbouring concepts are the direct sub-classes, direct super-classes and further direct related concepts.

Taking into account the neighbouring concepts the ML can easily find out in which context instances of a concept are returned. All this information are encoded in the RDF result fragments in order to provide both the user additional information that he/she can use to refine the current query in a subsequent search process (see section 4.7) and the ML in order to allow reasoning on the returned results. Thus, the RDF result fragment is produced in combination of RQL resource querying and navigation on the RDF(S) store.

The answering of RQL queries takes the RDF model theory [13] into account, which includes some basic inference over inheritance hierarchies as well as domain and range constraints. In addition, we extended the inference rule engine built in Sesame [9] that works as a production system generating inferred RDF triple facts from explicitly stated ones according to rules and axioms expressed in a proprietary XML format. In particular the transitivity of user defined properties is realized in this way.

4.5 Result Collection and Adaptation

Presenting the retrieved results in an efficient way for the users to access them is one of the basis of any good searching activities. The returns produces are assembled into a document with well-known structure, which follows a proposal discussed at the W3C consortium as a standard for query result formatting [12]. This format is made up of very simple yet efficient structure binding variables with their values.

Along with raw data, each result also contains some special meta-information, helpful to better organize, filter and sort the information before presentation to the user, in a process that also encompasses some rewriting steps much like what happens for the queries as explained above. The additional information is the following:

- the source, which each return comes from
- the context of each return (i.e., similar or related concepts)

- an additional ontology fragment, in case the provider ontology is richer than the domain one and a direct mapping cannot be established.
- the relationship with what was originally asked for in the system query (as heterogeneity could cause this reference to be lost)

The formatting process, besides endowing every result with the same structure, also allows for a much simpler filtering procedure, whose purpose is to drop duplicate answers and evident useless information, evaluated on the basis of a syntactic analysis over the results. The final step in the search process is the logical opposite of the query adaptation and negotiation described above. At this stage all the results are expressed by a common structure and format, but the terms used refer to the related provider ontologies. Hence a final mapping of the results to comply with the system ontology must take place. This mapping can be seen as the counterpart of the one already described above. During this activity, each variable of the result is also tagged using a special prefix to emphasize whether or not that term is known to the system ontology, in order for the Meta Level to take that into account for the subsequent phase.

4.6 Semantic Result Processing

Once the Meta Level has sent a set of system queries to the Agency, it assumes that the search process has been launched. At that moment, the main role of the Meta Level is to retrieve the returns of each one of the queries, evaluate and rank them and prepare the graph of the results that will be presented to the user. The Agency sends the results for each of the system queries produced immediately back to the ML, without waiting to have completed them all to it. As it gets them, the ML is responsible for the following actions: grouping the returns, ranking the returns, and constructing the graphical visualization.

The ML, independently from the information sources the returns come from, groups them depending on the system query they belong to. This is an important process since the visualization of the results will be different depending on the semantics of each system query.

The ML evaluates the returns using the information contained in the results, the terms included in the system query the return belongs to, the concepts involved in the task context the query is related to and the concepts appearing in the user query. The idea is to measure the distance not only to the original query but also to the task context where the query is located. If the returned results contain instances of concepts that are

(a) not specified in the system query and

(b) unknown in the ML ontology,

the results are considered as unclassified, and are graphically shown in a different node of the graph.

Using as a start point the graphs that represent each one of the system queries that the returns belong to, the ML builds a result visualization graph where the final results will be attached. This graph shows not only the concepts involved in the results, but also the relationships among them.

4.7 Domain Knowledge Enhanced Result Presentation

After the semantic processing of the results the ML attaches each set of results to one of the concepts of the graph. Beside this, the ML provides each bunch of results of each system query with a semantic path, which shows the set of concepts each result is related to. Thus, although each result appears under a concrete concept, the user is able to see the semantic contextualization of the result. This is done using the graph information of the system query to which the result belongs.

The query refinement is the subsequent browsing of the results presentation that supports further exploration of how the search specification might be useful further developed to better meet his or her information needs. In this way, from the user perspective, the query refinement effectively merges with the query development (see Figure 4) one to form what can be understood as a kind of design process.

5 Technology Used and Problems Encountered

In our approach we have done experiments and investigations with many Semantic Web tools and technologies. In the current implementation we use:

- Protégé [8] for modelling the domain ontology and provider ontology
- RACER [10] for doing reasoning on the domain ontology
- Sesame [9] for querying and navigating the Semantic Information Sources
- RDF(S) [5, 6] for describing content in the Semantic Information Sources
- RQL [3] for the System Queries (SQ)
- OWL [7] for describing the domain ontology
- W3C result format [12] for transforming the results coming from the heterogeneous information sources into a common result format

Based upon the work done so far, the good experience with respect to the use of these can be summarized as: OWL better supports the knowledge representation work involved in building the domain ontology, and other ML ontologies. RQL offers an effective low level query language. Protégé, with the OWL plug-in, provides a good ontology editor and development environment.

The following problems encountered when using these technologies: Sesame, like other general purpose ontology stores, is currently too slow for the query process and to support the kind of ontology-based inference needed by the ML. In order to increase the performance concerning the query process we implemented a logic-based query optimizer for RQL queries. RACER can provide useful support to ontology development, but becomes too slow for ontologies like the ML domain ontology (with approximately 790 concepts and individuals, and 150 relations). None of the published ontology development methods [11], either do not have a validation step, or are strong enough to support effective validation of realistic sizes of ontologies. The rather toy examples typically used to present these methods also don't help much in understanding how to apply them to real ontology developments. Furthermore, we explored that the expressiveness of the ontology description language RDF(S) used in the CL are restricted according to our needs. Currently, it is not possible to relate instances and concepts with user defined RDF(S) properties. If this would be possible, query languages need to be extended in order to allow querying those modelling approaches. The user testing sessions showed that the user interface of the ontology editing tool Protégé is not for everybody. It is still too complex for users who are not familiar with it.

6 Future Needs

To support interoperability between heterogeneous semantic web applications, the need comes up to standardize ontologies. Furthermore, high-performance OWL tools that can handle large size ontologies are still missing.

Acknowledgements

This work was funded in part by the European Commission Grant #IST-2001-34417, Semantic Web-based Information Management and Knowledge Sharing for Innovative Product Design and Engineering (WIDE project).

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Scientific American*, pp 34–43, May 2001.
[2] Backus-Naur form (BNF), WIKIPEDIA, <http://en.wikipedia.org/wiki/Backus-Naur Form>.
[3] The RDF Query Langauge (RQL), FORTH Institute of Computer Science, <http://athena.ics.forth.gr:9090/RDF/RQL/>.
[4] Association for Standarisation of Automation and Measuring Syetems (ASAM)Open Data Service (ODS), <http://www.asam.net/01 asam-ev 01.php>.
[5] Resource Description Framework (RDF), W3C Semantic Web Activity, Technology and Society Domain, <http://www.w3.org/RDF/>.
[6] RDF Vocabulary Description Langauge 1.0: RDF Scheme, W3C Technical Reports and Publications, <http://www.w3.org/TR/rdf-schema/>.
[7] OWL Web Ontology Language Overview, W3C Technical Reports and Publications, <http://www.w3.org/TR/owl-features/>.
[8] The Protégé Ontology Editor, Stanford Medical Informatics, Stanford University School of Medicine, <http://protege.stanford.edu/>.
[9] J. Broekstra and A. Kampman and F. van Harmelen, ”Sesame: A generic architecture for storing and querying RDF and RDF Schema, International Semantic Web Conference (ISWC), pp 54-68, 2002.
[10] RACER: Semantic Middleware for Industrial Projects based on RDF/OWL, <http://www.cs.concordia.ca/˜haarslev/racer/>.

[11] A. Gomez-Perez, M. Fernandez-Lopez and O. Corcho, "Ontological Engineering," London:Springer-Verlag, 2004.
[12] A. Seaborne, Recording Query Results, W3C Discussion document http://www.w3.org/2003/03/rdfqr-tests/recording-query-results.html
[13] RDF semantics W3C Recommendation 10 February 2004 http://www.w3.org/TR/rdf-mt/
[14] S. Staab, M. Erdmann, and A. Maedche. An Extensible Approach for Modeling Ontologies in RDF(S). In First ECDL'2000 SemanticWebWorkshop, Lisbon, Portugal, 2000.
[15] H. Stuckenschmidt et al. Exploring Large Document Repositories with RDF Technology: The DOPE Project Published by the IEEE Computer Society http://www.cs.vu.nl/~frankh/postscript/IEEE-IS04.pdf
[16] N. Shadbolt, N. Gibbins, H. Glaser, S. Harris, and m.c.schraefel, University of Southampton; CS AKTive Space, or How We Learned to Stop Worrying and Love the Semantic Web; Published by the IEEE Computer Science;