DISCOVERING KNOWLEDGE IN WEB ONTOLOGIES: A METHODOLOGY AND PROTOTYPE IMPLEMENTATION*

Dimitrios A. Koutsomitropoulos†, Markos F. Fragkakis‡,
Theodore S. Papatheodorou†

Abstract

One of the most prominent features that the Semantic Web promises to enable is the discovery of new and implied knowledge from existing information that is scattered over the Internet. However, adding reasoning capabilities to the existing Web infrastructure is by no means a trivial task. Current methods and / or implementations do not seem to be declarative and expressive enough to provide the kind of reasoning support that the Semantic Web users will benefit from. In this paper we propose a methodology based on which, the user can construct and pose intelligent queries to Semantic Web documents in an intuitive manner, without prior knowledge of the document's contents or structure. This methodology is implemented through a knowledge discovery prototype interface that relies on an existing inference engine found suitable for this task.

1. Introduction

One of the most important features promised by the Semantic Web is the capability of reasoning and knowledge discovery. In fact, this capability will not be provided in a static environment, as in traditional Artificial Intelligence systems, where the knowledge bases are centralized. On the opposite, knowledge discovery will be dynamic and distributed, utilizing resources that are scattered over the Internet. This is the primary reason why knowledge augmentation on the Semantic Web is monotonic: New statements and facts can only be added, augmenting monotonically the ontologies. These new facts may contradict the previous ones (possibly making the ontology unsatisfiable), but in no case can they retract or cancel them.

Semantic Web has the ability to define and express knowledge through the use of ontologies. The recently finalized Ontology Web Language standardizes the expressiveness levels of the Semantic Web and demonstrates characteristics suitable for its distributed environment. Ontology documents can be seen as knowledge bases containing knowledge about various domains, distributed over the Web and possibly augmenting each other. By the term *knowledge discovery* we mean the ability of discovering information that is not explicitly expressed in the available data of these ontology documents. Knowledge discovery is thus a result of reasoning which, by leveraging existing

---

* This is an extended version of a paper titled "A Methodology for Conducting Knowledge Discovery on the Semantic Web", Applied and Personalized Semantic Web Workshop, HyperText 05, Salzburg, Austria.

† High Performance Information Systems Lab, Computer Engineering and Informatics Dept., University of Patras, Patras, Greece, email: {kotsomit, tsp}@hpclab.ceid.upatras.gr

‡ School of Informatics, University of Edinburgh, Edinburgh, UK, email: M.Fragkakis@sms.ed.ac.uk

information (statements and facts) leads to the extraction of logical conclusions that are not explicitly expressed. Knowledge discovery is therefore different from information retrieval, which aims at finding information that, while literally expressed, is hard to retrieve by a machine. It also differs from data mining, which primarily seeks to discover repeating patterns and implied relations in large databases.

Several solutions have been proposed focused on founding or improving reasoning tasks since the advent of the Semantic Web. Ontology description languages, inference systems and engines, as well as implementations with reasoning capabilities have been developed in order to improve information retrieval and enable knowledge discovery, if possible. Examining existing approaches reveals that no well-established and standardized methodology is followed in general. In addition, the expressiveness achieved varies, not always fulfilling the expressiveness needs of the Semantic Web. Finally, current approaches for composing and performing intelligent queries do not seem to be satisfyingly declarative. Most of the time, the user is burdened with the task of collecting the appropriate information needed to construct the query.

In this paper we propose a methodology for conducting knowledge discovery on the Semantic Web. This methodology consists of three phases: First, the selection of the appropriate logical formalism that will enable the use of existing AI tools and reasoners to perform inferences on Semantic Web documents. Second, the identification of certain key aspects that need to be taken into account for our method to be suitable for a Web environment. Third, the selection of a specific inference engine that meets the criteria posed in the previous phase.

An integral part of our methodology is the *Knowledge Discovery Interface* (KDI). The KDI is a web application that has been developed as an implementation of the decisions and criteria developed during the three phases. As such, its target is threefold:

- To be *expressive* enough, in order to allow for powerful inferences on ontology documents.
- To be *declarative*, being independent of the specific ontology schema or contents.
- To be *intuitive*, by aiding the user to compose his query in a user-friendly manner and allowing transparent query composition.

The rest of this paper is organized as follows: In section 2 we review and discuss some previous work on information retrieval and knowledge discovery. Then, in section 3, we propose our methodology for knowledge discovery on the Semantic Web. The KDI is presented in section 4 alongside with some experimental results that demonstrate its capabilities. In section 5 we propose ways in which the presented work could be continued and improved. Finally, section 6 summarizes the conclusions from our work.

2. Approaches for Reasoning on the Web

Even though the idea of the Semantic Web has only recently begun to standardize, the need for inference extraction and intelligent behaviour on the Internet has long been a research goal. As expected, there have been some efforts in that direction. Such efforts include ontology description languages, inference engines and systems and implementations, based on them.

SHOE (Simple HTML Ontology Extension) [8] was initially developed as an extension to HTML. It enables web page authors to annotate their web documents with machine-readable knowledge. As a result, these documents can be retrieved by knowledge-based search engines in a more efficient way, and then processed by agents. Although SHOE has a number of features, some of which are not present in other languages (e.g. n-ary relations), it lacks the expressiveness needed by the Semantic Web (for example see [4]).

Knowing the constraints of knowledge discovery in a random environment like the Internet, and taking into account the advantages of information retrieval, recent research has tried to combine the two approaches. OWLIR [16], for instance, is a system conducting retrieval of documents that are enriched with markup in RDF, DAML+OIL or OWL. A text editing and extraction system is used to enrich the documents, based on an upper level ontology. This extra information is processed by a rule-based inference system. Search is conducted using classical retrieval methods; in addition, however, the results are refined using the inference system results.

The TAP framework [6] shares the goal of improving the quality of search results by utilizing the semantic relationships of web documents and entities. However, no inference takes place here. Instead, the RDF / OWL documents are treated as structured metadata sets. These sets can be represented as directed graphs, whose edges correspond to relations, and vertices correspond to existing internet resources. This representation is conducted based on the information of a local knowledge base.

The growth and maintenance of a knowledge base is a strenuous procedure, often demanding extensive manual intervention. The Artequakt system [1] tries to overcome this obstacle by following an automated knowledge extraction approach. Artequakt applies natural language processing on Web documents in order to extract information about artists and artistic world to populate its knowledge base. Stored knowledge is then used to automatically produce personalised biographies of artists. The CIDOC-CRM ontology is used as the “conceptual schema” for the information that needs to be extracted from the documents and stored in the knowledge base. Nevertheless, it should be noted that no inference - and thus knowledge discovery - takes place.

The Wine Agent system [14] was developed as a demonstration of the knowledge discovery capabilities of the Semantic Web. This system uses a certain domain ontology written in DAML+OIL / OWL and performs inferences on it. The Wine Agent employs a first order logic theorem prover (JTP).

The Semantic Web standardization procedures and particularly the finalisation of DAML+OIL and OWL, was followed by specifications on query languages. The need for formal querying methods with induction capabilities, has led to DQL as well as OWL-QL [3]. DQL and OWL-QL play an important role that enables intelligent systems on the Semantic Web, and also contribute to their

expansion and interoperability. Nevertheless, they do not provide a direct answer to the knowledge discovery issue. Instead, they serve mainly as communication protocols between agents. Namely:

- They are mainly used as formal client-server protocols for exchanging queries and answers between a querying agent (client) and an answering agent (server).

- They are not declarative enough, in the sense that previous knowledge of the ontology is required in order to compose a query. Furthermore, the composition of a query compatible with these protocols requires decomposition of the original, intuitive query by the user, and its re-composition following some normative specification.

- They do not prescribe a way to conduct inference, since such a task is beyond their scope. As a result, these query languages cannot be used to develop knowledge discovery services, but merely serve as an extra communication layer on top of them.

The approaches and systems discussed so far have been developed in order to cover specific needs for knowledge extraction and for particular knowledge domains. They contribute to the knowledge discovery task either by processing resources to extract useful knowledge (e.g. Artequakt) or by automatically enriching documents with semantic information (e.g. TAP) and using intelligent techniques to process this information (OWLIR and Wine Agent). However, at least as far as knowledge discovery is concerned, they follow fragmented approaches and they do not propose or adhere to any underlying framework that could be generally applied. Even when they do perform knowledge discovery their expressiveness is limited and their approaches are hardly reproducible in the case of intelligent querying of Semantic Web documents.

3. An Inference Methodology for the Semantic Web

Knowledge discovery does not coincide with traditional data retrieval, at least not in the way the latter is standardized by query languages and protocols – e.g. in a relational database. However, it too, assumes the formulation and composition of some kind of intelligent queries, the answers to which will compose the requested knowledge. For example a typical query in SQL or XQuery is not suitable for retrieving answers from an inference engine. Neither the latest proposals on Semantic Web query languages, as shown in the previous section, seem to be adequate for modeling such declarative queries, since they do not provide a concrete inference methodology. Therefore, a major difficulty that needs to be addressed is the development of a concrete and declarative method for design intelligent queries, which, after submitted to an inference engine would lead to the discovery of implied information.

In this section we propose the basic characteristics that such a method should have in order to be appropriate for the Semantic Web environment. Having resolved on an appropriate logical formalism, the resulting methodology relies on an inference engine that should meet certain criteria and be suitable for this purpose. By combing low-level functions of such an engine, our methodology aims at helping the user construct his query in a declarative manner and enables expressive intelligent queries to web ontology documents. A prototype implementation of this methodology, the KDI, is presented in Section 4.

3.1. Choosing the appropriate formalism

Choosing an underlying logical formalism for performing reasoning is crucial, as it will determine to a great extent the expressiveness to be achieved. In this subsection we will consider some available formalisms, as well as a number of existing tools for each of them.

Description Logics (DLs) form a well-defined subset of First Order Logic (FOL). OWL Lite and OWL DL are in fact very expressive description logics, using RDF syntax [10]. Therefore, the semantics of OWL, as well as the decidability and complexity of basic inference problems in it, can be determined by existing research on DLs. OWL Full is even more tightly connected to RDF, but its typical attributes are less comprehensible and the basic inference problems are harder to compute (because OWL Full is undecidable). Inevitably, only the examination of the relation between OWL Lite/DL with DLs may lead to useful conclusions. On the other hand, even the smaller subsets of OWL differ from DLs in certain points, including the use of namespaces and the ability to import other ontologies.

It has been shown [9] that OWL DL can be reduced to *SHOIN*(D) in polynomial time, while there exists an incomplete translation of *SHOIN*(D) to *SHIN*(D). This translation can be used to develop a partial, though powerful reasoning system for OWL DL. A similar procedure is followed for the reduction of OWL Lite to *SHIF*(D), which is completed in polynomial time as well. In that manner, inference engines like FaCT and RACER can be used to provide reasoning services for OWL Lite/DL.

The selection of a DL system to conduct knowledge discovery is not the only option. A fairly used alternative is the use of inference systems that obtain reasoning using applications based in *FOL (theorem provers)*. Such systems are Hoolet, using the Vampire theorem prover, Surnia, using the OTTER theorem prover and JTP used by the Wine Agent. Inference takes place using axioms that reflect the semantics of statements in OWL ontologies. Unfortunately, these axioms often need to be inserted manually. This procedure is particularly difficult not only because the modeling axioms are hard to conceive, but also because of the need for thorough verification. In fact, there are cases where axiom construction depends on the specific contents of the ontology [14].

Another alternative is given by *rule based reasoning systems*. Such systems include DAMLJessKB [15] and OWLLisaKB. The first one uses Jess rule system to conduct inference on DAML ontologies, whereas the second one uses the Lisa rule system to conduct inference on OWL ontologies. Similarly to the case of theorem provers, rule based systems demand manual composition of rules that reflect the semantics of statements in OWL ontologies. This can also be a possible reason for which such systems can presently support inference only up to OWL Lite.

Rule based inference services are also included in the Jena framework [17], developed by Hewlett-Packard. Jena provides a programming environment for web ontologies and supports inference up to OWL Lite level. IBM's corresponding solution is the SNOBASE ontology management system, featuring similar reasoning capabilities.

On the other hand, neither the currently available Description Logic systems nor the algorithms they implement, support the full expressiveness of OWL DL. Even if such algorithms are implemented, their efficiency will be doubtful, since the corresponding problems are optimally solved in non-deterministic exponential time. In [11] a decision procedure is presented for the

SHOIQ Description Logic; this algorithm is claimed to exhibit controllable efficiency and is currently under implementation in two high-end inference engines.

DLs seem to constitute the most appropriate available formalism for ontologies expressed in DAML+OIL or OWL. This fact also derives from the designing process of these languages. In fact, the largest decidable subset of OWL, OWL DL, was explicitly intended to show well studied computational characteristics and feature inference capabilities similar to those of DLs. Furthermore, existing DL inference engines seem to be powerful enough to carry out the inferences we need.

3.2. Suitability for the Web

Research regarding the use of ontologies and Description Logics for semantic matchmaking of Web Services descriptions [5],[19] as well as other types of resources [17] has led to the recognition of a common query method, using subsumption relations between concepts. According to this method, the query is modeled as a new concept, using the Description Logic constructors, and then classified in the hierarchy. The concepts that are subsumed to this new query-concept, as well as their synonyms, are considered to be absolute answers to the query. Concepts that are subsumed by this new query-concept may also be considered to be relevant answers, although they do not fully meet the query criteria. Their distance from the query-concept in the hierarchy can be used in order to determine the degree to which these criteria are met. Furthermore, every instance can be modeled as an atomic concept, so queries demanding use of instances can be conducted the same way [13]. The method just described, could be characterized as *taxonomic* since it is entirely based on the functions provided by the TBox of a knowledge base.

Although 'virtual' class creation seems to be adequate for the matchmaking of Web Service descriptions, querying the Semantic Web demands the added expressiveness provided by the use of instances. The ABox intelligent functions, such as instance checking, are of crucial importance when domain modelling calls for the fine-grained analysis that instances enable. This is especially true in an environment like the Web, where most of the semantic structures are of unspecified and arbitrary detail. In any case, inserting instances in ontology enables inferences and expressions that would have been impossible to accomplish using concept classification solely. Take for example the case where one wants to express that a sword is made of iron. The OWL description for this would be:

```
<owl:Class rdf:ID="sword">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#isMadeOf"/>
         <owl:someValuesFrom rdf:resource="#Iron"/>
      </owl:Restriction>
   <rdfs:subClassOf>
</owl:Class>
```

The expression describing that iron is metal would be:

```
<owl:Class rdf:ID="Iron">
   <rdfs:subClassOf rdf:resource="#Metal" />
</owl:Class>
```

One can now retrieve every metal item:

```
<owl:Restriction>
   <owl:onProperty rdf:resource="#isMadeOf"/>
   <owl:someValuesFrom rdf:resource="#Metal"/>
</owl:Restriction>
```

Queries of that type are easily answered that way, using only the TBox of the ontology. But what if someone wants to find out the material that *sword* is made of? If reasoning in Abox is employed, the answer is straightforward by using a role-filler query.

The KDI presented in the following section utilizes therefore an *instance-based* method for conducting inferences that employs in its core two basic ABox functions: *instance checking*, that finds the concepts an instance belongs to and *role fillers retrieval*, which, given a specific instance and role, infers all related instances through this role. By utilizing these two features, in combination with the TBox functions, we can achieve not only the retrieval of information that has been explicitly expressed in the ontology, but also the discovery of knowledge that is logically deduced by this information. Therefore TBox as well as ABox support is an important criterion for selecting an appropriate reasoning back-end.

3.3. Choosing a reasoning back-end

Having chosen to use the DLs as the underlying formalism for our methodology, and having noted the most important characteristics that would be desirable for a suitable implementation, we will now examine three inference engines based on DLs. Our evaluation is carried out in terms of expressiveness, support for OWL, reasoning about ABox and other main features provided by each of the three systems.

Cerebra, by Cerebra Inc. (formerly Network Inference) is a commercial system, providing reasoning as well as ontology management features. Cerebra differentiates from traditional DL based systems, in that it provides some extra features that may be desirable in a production environment. One of the most interesting ones is the addition of persistency to the knowledge bases it processes, by storing locally ontology information. Nevertheless, its expressive power is by no means exceedingly different.

Cerebra supports nearly all constructors and axioms that would normally classify it to OWL DL expressiveness level. However, our experimentation with the system has shown some deficiencies which, in combination with its inability to reason about the ABox finally ranks Cerebra's expressiveness at *SHIQ* level, at most. On the other hand, the relation model used by Cerebra allows the submission of very powerful queries, based on XQuery syntax. Still, these results are based only on the explicitly expressed information of the ontology, and not on information that may be inferred.

FaCT / FaCT++ [11] is a freely available reasoning software, that is being developed in Manchester. FaCT implements optimised, sound and complete algorithms to compute subsumption in the *SHIQ*(D) DL. FaCT does not support reasoning in ABox, neither concrete domains. It is also syntactically incompatible with OWL, since its knowledge bases are expressed in a Lisp-like or XML syntax.

Many of FaCT's disadvantages are not present in the system's next version, FaCT++[20]. FaCT++ features greater expressiveness, aiming ultimately at OWL DL. Specifically, full support for concrete domains is added, while the underlying logic is *SHIF*(D). OWL syntax is not supported;

however a transformation tool to the Lisp intermediate form is available. Individuals (and thus nominals) survive this transformation, but they are not yet fully supported, as they are approximated as primitive concepts. While FaCT++ documented version could only run in Linux and provides no communication interface with other applications, systems' latest version (ver. 1.1.2) claims to support DIG/1.1 plus unqualified number restrictions and nominals.

RACER [7] is an inference engine for very expressive DLs. It is the first system in its category to support reasoning in ABox as well as TBox, and this is its main asset in comparison to the other inference engines. RACER provides reasoning for *SHIQ*(D), including instances (ABox). There is also full support for the OWL syntax. In fact, RACER expressiveness is superior to OWL DL in regards of qualified number restrictions and concrete domains.

On the other hand, OWL semantics are more expressive than RACER language as far as *nominals* are concerned, because they are not supported by the system. This seems to be the main problem that prevents full compatibility with OWL. RACER deals with nominals by creating a new concept for each of them.

RACER seems to be closer to the expressiveness needed by the Semantic Web mostly because of its enhanced support for OWL and its clear ability to reason about the ABox. Its utilisation in the KDI produced a number of interesting results, some of which are presented in Section 4.3.

4. The Knowledge Discovery Interface

In this section we describe the prototype Knowledge Discovery Interface. The KDI has been developed as an implementation of the decisions and criteria identified in the previous section. First we give a general description of the KDI and the main technologies that were used, along with a brief description of its functionality. Afterwards follows an examination of its programming architecture, as well as the formal relation between the query composition and submission process and the functions provided by RACER. Finally, we present two experimental inferences on OWL documents performed using the KDI, as well as their results.

4.1. General Description and Architecture

The KDI is a web application, providing intelligent query submission services on Web ontology documents. We use the word *Interface* in order to emphasize the fact that the user is offered a simple and intuitive way to compose and submit queries. In addition, the KDI interacts with RACER to conduct inferences.

The interface can load OWL documents that are available either on the local file system, or on the Internet. After connection to RACER has successfully been established, the ontology is loaded and its information is shown on the browser (Figure 1). The user may navigate through the concept hierarchy, which is visualised in a tree form, and select any of the available classes. Upon selection, the page is reloaded, now enabling two drop down menus, the first one listing all the instances of the selected class and the second listing all the roles whose domain is in this class. The user is able to select an instance and a role and then submit his query by pressing the corresponding button. Note that an option is available to invert the selected role, thus resulting in a different query.

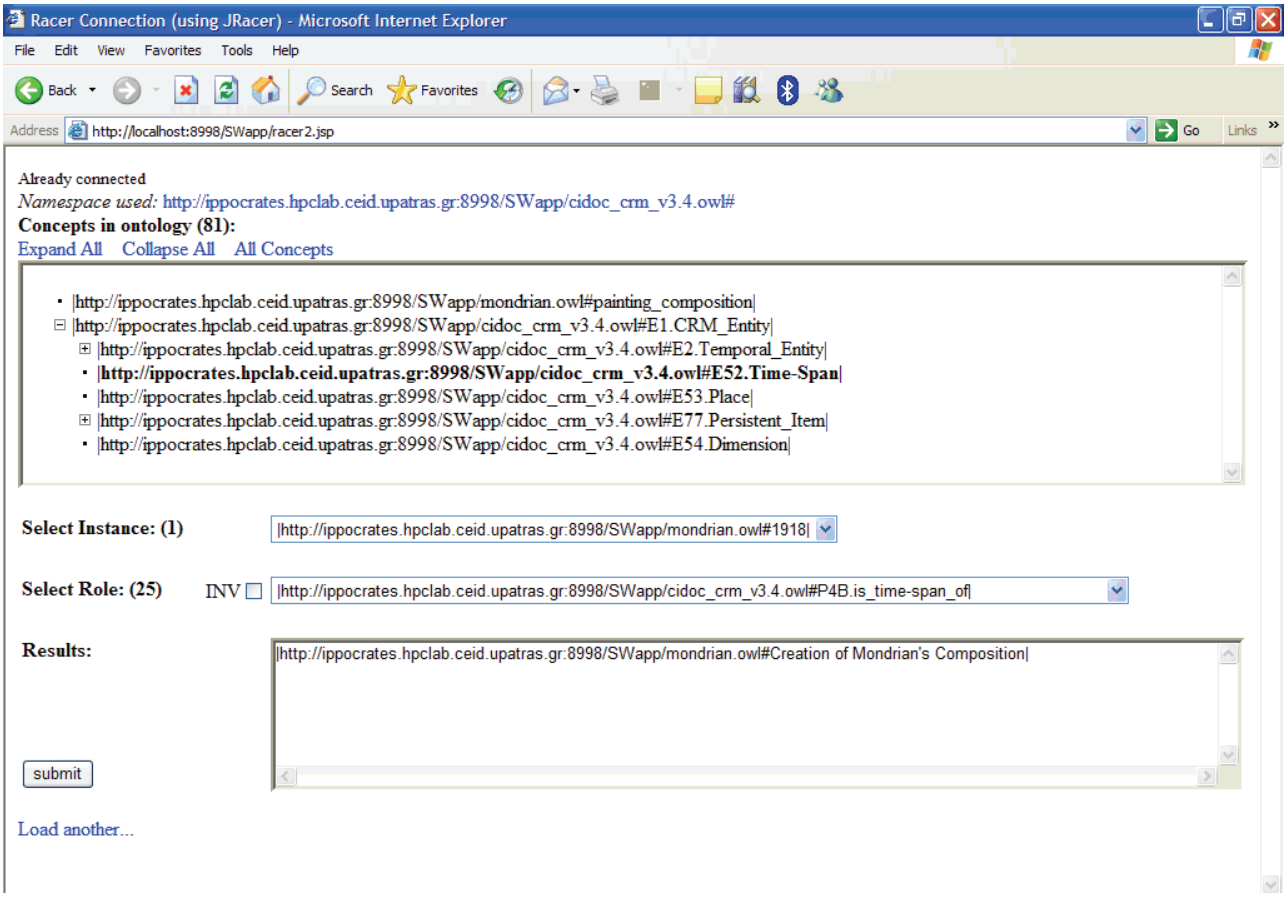

Figure 1: *Ontology classification and intelligent query composition using the KDI. The concept "E52.Time-Span" has been selected, thus only its instances and relevant roles are available. System correctly infers that "1918" is the year of the "Creation of Mondrian's composition".*

The Interface is capable of loading ontologies of any structure and content. Furthermore, the user is capable of browsing the existing classes, their instances and their corresponding roles. In this way, all the needed information to compose a query is made available. We have identified such a *declarative* behaviour to be of crucial importance for the Semantic Web knowledge discovery process; after all, the user should be able to pose queries to any ontology, even ones encountered for the first time.

The Interface helps the user compose a query by selecting a concept, an instance and a role in a user-friendly manner. After the query is composed, it is decomposed into several lower level functions that are then submitted to RACER. This procedure is transparent to the user, withholding the details of the knowledge base actual querying. This kind of intuitive composition of intelligent queries, significantly improves the knowledge discovery process by facilitating the user to pose precise and correct queries.

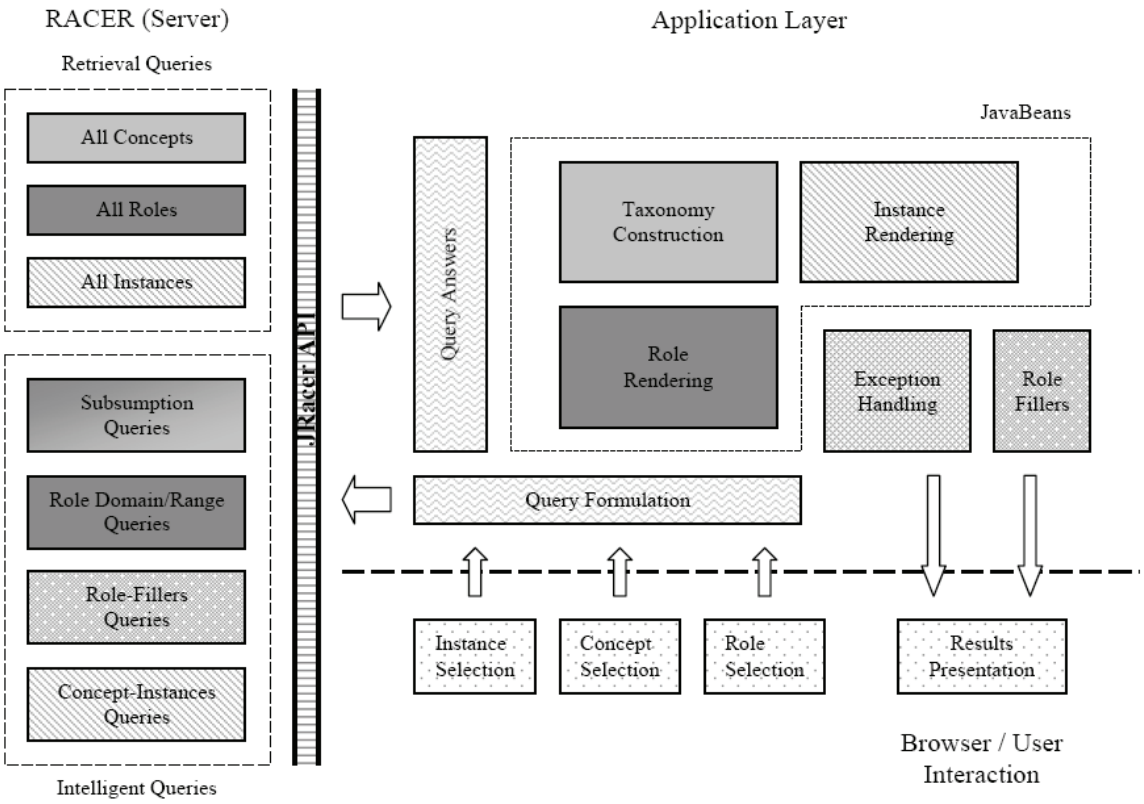

Figure 2: *The Architecture of the Knowledge Discovery Interface.*

4.2. Architecture

At this point, we will examine the programming components that constitute the KDI. Figure 2 shows the various parts of the implemented business-logic (right part) as well as their relation to and dependence on RACER's lower level functions (left part): a functions pattern depicts the components it participates in.

The application layer is responsible for the structure, processing and appearance of results, as well as exception handling. Hierarchy construction and collection of the proper instances and roles are implemented as *set* and *get* methods of a JavaBean. Hierarchy construction and role collection take place only once in a user's session, during the loading of the ontology.

In order to construct the hierarchy, the names of all ontology concepts are initially received ("All Concepts"). Concepts are then classified using subsumption queries to RACER, in combination with a depth-first search algorithm. Instances collection involves the *all-instances* function, as well as the queries that return all the instances of a specific concept ("Concept-Instances Queries"). Similarly, role rendering uses the *all-roles* function, queries returning the domain or range ("Role-Domain / Range Queries") and subsumption queries.

Exception handling and role filler discovery are implemented in JSP scriptlets. Filler discovery takes into account whether the selected role has been inversed, as well as whether the role is actually a datatype property, and combines the results from corresponding queries ("Role-Fillers Queries") to RACER. The results, or potential errors are processed using a combination of JSP and HTML code, and are then shown in the user's browser.

Generally, the construction of queries that are submitted to RACER is conducted either by using JSP code or by the corresponding methods in the JavaBean. This construction is based upon the concept, instance and role (and inverse) chosen by the user. These choices are passed to the KDI through the user submission of HTML forms. For the queries submission to RACER and for the answer reception, we use certain methods of the JRacer API.

4.3. Results

In this section we present the results from two different inference actions performed using the KDI, so as to demonstrate its capabilities as well as its limitations. In order to conduct these inferences we use the CIDOC Conceptual Reference Model [2] as our knowledge base. CIDOC-CRM is domain ontology specializing on the cultural heritage domain. Currently, the machine understandable format closer to the Semantic Web that CIDOC-CRM is available in, is RDFS.

Firstly, we ported version 3.4 of the CRM to OWL format in a file named cidoc_crm_v3.4.owl. Secondly, we semantically enriched and extended CRM with concrete instances and more expressive structures, available only in OWL (like cardinality restrictions, inverse roles, existential and universal quantifications and so on). Although these extensions could have been integrated in the initial document, we chose to include them in a new file. The reason for this is to display in clearer way the capabilities of the Semantic Web technologies for ontology integration and distributed knowledge discovery.

More specifically, we created a document named mondrian.owl that includes CRM concept and role instances, which model facts from the life and work of the Dutch painter Piet Mondrian. In this document we also included axiom and fact declarations that OWL allows to be expressed, as well as new roles and concepts making use of this expressiveness.

The aforementioned documents were made available on the Internet through the Tomcat server. Inclusion of cidoc_crm_v3.4.owl axioms was possible simply by using the <owl:imports> declaration in mondrian.owl. Therefore, loading mondrian.owl through the KDI loads all the axioms from cidoc_crm_v3.4.owl as well, as long as the latter is available on the Internet. In order to resolve potential ambiguities, different namespaces were defined for each document. In order to refer to statements from the imported ontology, the crm prefix is used, whereas for the new statements the default prefix (#) is used.

The following code is a fragment from mondrian.owl stating that a “Painting_Event” is in fact a “Creation_Event” that “has_created” “Painting” objects only:

```
<owl:Class rdf:ID="Painting_Event">
   <rdfs:subClassOf  rdf:resource="&crm;E65.Creation_Event"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="&crm;P94F.has_created"/>
         <owl:allValuesFrom rdf:resource="#Painting"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
<Painting_Event rdf:ID="Creation of Mondrian's composition">
<crm:P94F.has_created rdf:resource="#Mondrian's composition"/></Painting_Event>
```

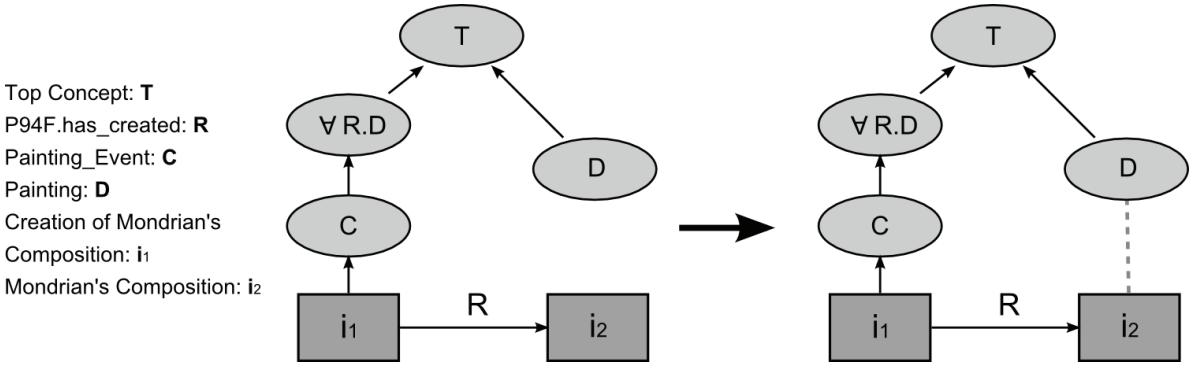

Figure 3: *Inference example using Value Restriction*

The above fragment is graphically depicted in the left part of Figure 3. "Creation of Mondrian's Composition" (***i*₁**) is an explicitly stated "Painting_Event" that "has_created" (***R***) "Mondrian's composition" (***i*₂**). Now, asking the KDI to infer "what is a painting?" it infers that i_2 is indeed a painting (right part of Fig. 3), correctly interpreting the value restriction on role R.

Let's now examine another example that involves the use of nominals. The following fragment from mondrian.owl states that a "Painting" is a "Visual_Item" whose "Type" is "painting_composition".

```
<owl:Class rdf:ID="Painting">
   <owl:subClassOf rdf:resource="&crm;E36.Visual_Item"/>
   <owl:equivalentClass>
      <owl:Restriction>
         <owl:onProperty rdf:resource="&crm;P2F.has_type"/>
         <owl:hasValue rdf:resource="#painting_composition"/>
      </owl:Restriction>
   </owl:equivalentClass>
</owl:Class>
<crm:E55.Type rdf:ID="painting_composition"/>
<Painting rdf:ID="Mondrian's composition" />
```

The above fragment is graphically depicted in the left part of Figure 4.

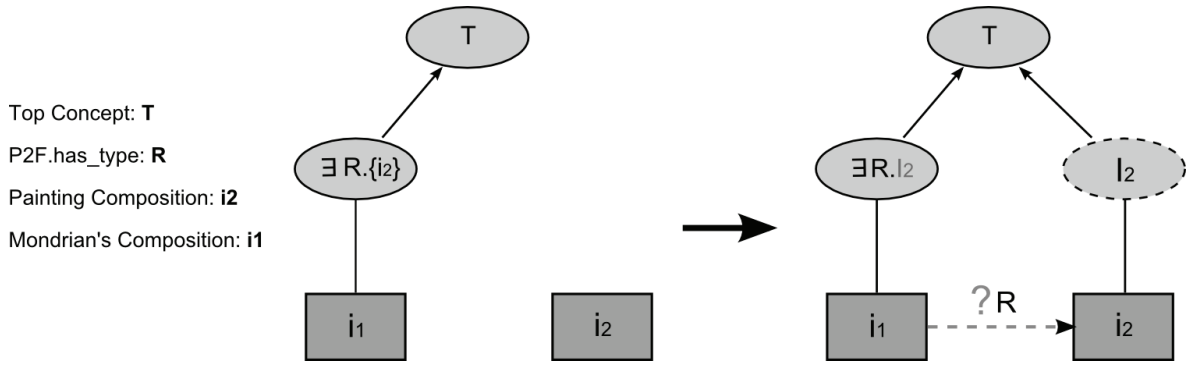

Figure 4: *Inference example using Existential Quantification and nominals.*

"Mondrian's Composition" (***i*₁**) is explicitly declared as a "Painting" instance which in turn is defined as a hasValue restriction on "has_type" (***R***). "Painting_composition" (***i*₂**) is declared as a "Type" object. While the fact that "Mondrian's Composition" "has_type" "Painting" is straightforward, the KDI is unable to infer so and returns *null* when asked "what is the type of Mondrian's composition?"

This example clearly demonstrates the difficulty of RACER as well as every other current DL based system to reason about nominals. Given the $\{i_2\}$ nominal, RACER creates a new synonym concept I_2 and makes i_2 an instance of I_2. It then actually replaces the hasValue restriction with an existential quantifier on *concept* I_2 and thus is unable to infer that $R(i_1,i_2)$ really holds.

5. Future Work

Regarding KDI, an important next step would be to investigate whether the utilization of a database achieves improved performance and to what extent. It is possible that having the instances and their relations (that is, the ABox) stored in a relational database will add persistency and will accelerate the retrieval functions. The knowledge base system can take over when reasoning on the concept and role hierarchy level is explicitly needed. The Interface could also benefit by some optimisations, mainly on the processing of large volume ontologies, where delays were observed during loading. There are indications however that these delays are not caused by the RACER system per se, but by the overhead posed by the JRacer API and the underlying TCP communication protocol.

The inference methodology may also be enriched, allowing the direct composition of taxonomical queries. This way, the user will be able to submit queries that demand the construction of new or artificial concepts (through the use of DL restrictions and constructors), without referring to OWL Full level. Queries involving the use of data types (although to a very limited extent) can also be answered in the same manner.

In addition, the study of existing Description Logic algorithms used by corresponding systems would clarify the possibilities of their expansion or adaption to the characteristics of the Semantic Web and OWL. Furthermore, it could possibly lead to the discovery of more reliable solutions to existing problems and obstacles like the use of nominals (e.g. see [11]). There is also a need for further research on alternatives to the Description Logic inference techniques and engines. This way we will be better able to assess their suitability for the Semantic Web, and develop (if possible) hybrid methods, which will improve inference results in terms of expressiveness and performance.

6. Conclusions

In this paper we have shown how the Semantic Web may fulfill a large part of what it promises, mainly through its knowledge discovery features. These features are grounded on a well-studied background and their adaption to the Web environment is now mature and even standardized to some extent. The KDI and its underlying methodology demonstrate proper evidence of how these features can be practically applied so as to be beneficial for a number of applications.

Our proposed methodology comes to fill the gap between existing approaches by designating a process that can be followed to achieve knowledge discovery on the Semantic Web. Our choices depend on the current state-of-the-art in inference engines and Semantic Web standardization efforts. We trust that in the near future most of the difficulties and incompatibilities identified throughout our work will be overridden by the evolution of systems and the refinement and possibly enrichment of the Ontology Web Language.

The KDI for example is greatly hampered by the limited expressiveness and scalability of the reviewed DL inference engines, regarding the use of nominals and the processing of large ontology documents respectively. Despite that fact, the KDI displays a combination of innovations and distinctive features that are not, to our knowledge, simultaneously met by any other system. Among the most important, is the use an intuitive and declarative way of constructing and submitting queries and the implementation of an original inference methodology that is especially suitable for the Semantic Web environment.

7. References

[1] ALANI, H., Kim, S., Millard, D. E., Weal, M. J., Hall, W., Lewis, P. H. and Shadbolt, N. R., Automated Ontology-Based Knowledge Extraction from Web Documents. IEEE Intelligent Systems, 18(1): 14-21, 2003.

[2] CROFTS, N., Doerr, M. and Gill , T., The CIDOC Conceptual Reference Model: A standard for communicating cultural contents. Cultivate Interactive, issue 9, 2003. http://www.cultivate-int.org/issue9/chios/

[3] FIKES, R., Hayes, P. and Horrocks , I., OWL-QL: A Language for Deductive Query Answering on the Semantic Web. KSL Technical Report 03-14, 2003.

[4] GOMEZ-PEREZ, A. and Corcho, O., Ontology Languages for the Semantic Web. IEEE Intelligent Systems, 17(1):54-60, 2002.

[5] GONZÁLEZ-CASTILLO, J., Trastour, D. and Bartolini, C., Description Logics for Matchmaking of Services, In Proc. of KI-2001 Workshop on Applications of Description Logics, 2001.

[6] GUHA, R., McCool , R., TAP: A Semantic Web Platform. Computer Networks, 42(5):557-577, 2003.

[7] HAARSLEV, V. and Möller, R., Racer: A Core Inference Engine for the Semantic Web. In Proc. of the 2nd Int. Workshop on Evaluation of Ontology-based Tools (EON2003), pp. 27-36, 2003.

[8] HEFLIN, J., Hendler, J. and Luke, S., Reading Between the Lines: Using SHOE to Discover Implicit Knowledge from the Web. In AI and Information Integration: Papers from the 1998 Workshop, pp.51-57. AAAI Press, 1998.

[9] HORROCKS, I. and Patel-Schneider, P. F., Reducing OWL entailment to description logic satisfiability. In Proc. of the 14th Int. Semantic Web Conf. (ISWC 2003), pp. 17-29, 2003.

[10] HORROCKS, I., Patel-Schneider, P. F., and van Harmelen, F., From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics, 1(1):7-26, 2003.

[11] HORROCKS, I. and Sattler, U., A tableaux decision procedure for SHOIQ. In Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), 2005.

[12] HORROCKS, I. and Sattler, U., Optimised reasoning for SHIQ. In Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002), pp. 277-281, 2002.

[13] HORROCKS, I. and Tessaris, S., Querying the Semantic Web: a Formal Approach. In Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002), pp. 177-191, 2002.

[14] HSU, E. and McGuinness, D., Wine Agent: Semantic Web Testbed Application. In Proc. Of Workshop on Description Logics, 2003.

[15] KOPENA, J. and Regli, W. C., DAMLJessKB: A tool for reasoning with the Semantic Web. IEEE Intelligent Systems, 18(3): 74-77, 2003.

[16] MAYFIELD, J. and Finin T., Information retrieval on the Semantic Web: Integrating inference and retrieval. In Proc. of SIGIR Workshop on the Semantic Web, 2003.

[17] MCBRIDGE, B., Jena: A Semantic Web Toolkit. IEEE Internet Computing, 6(6):55-59, 2002.

[18] Network Inference Ltd. Description Logics (white paper), http://www.networkinference.com

[19] PAOLUCCI, M., Kawamura, T., Payne, T. R. and Sycara, K., Semantic Matching of Web Services Capabilities. In Proc. of the 13th Int. Semantic Web Conf. (ISWC), 2002.

[20] TSARKOV, D. and Horrocks, I., Reasoner Prototype: Implementing new reasoner with datatype support. IST-2001-33052 WonderWeb Del. 13, http://wonderweb. semanticweb.org