Establishing the Semantic Web Reasoning Infrastructure on Description Logic Inference Engines

Dimitrios A. Koutsomitropoulos, Dimitrios P. Meidanis, Anastasia N. Kandili, and Theodore S. Papatheodorou

High Performance Information Systems Laboratory, School of Engineering
University of Patras, Buidling B, 26500 Patras-Rio, Greece
{kotsomit,dpm,nkandylh,tsp}@hpclab.ceid.upatras.gr

Abstract. The recent advent of the Semantic Web has given rise to the need for efficient and sound methods that would provide reasoning support over the knowledge scattered on the Internet. Description Logics and DL-based inference engines in particular play a significant role towards this goal, as they seem to have overlapping expressivity with the Semantic Web de facto language, OWL. In this paper we argue that DLs currently constitute one of the most tempting available formalisms to support reasoning with OWL. Further, we present and survey a number of DL based systems that could be used for this task. Around one of them (Racer) we build our Knowledge Discovery Interface, a web application that can be used to pose intelligent queries to Semantic Web documents in an intuitive manner. As a proof of concept, we then apply the KDI on the CIDOC-CRM reference ontology and discuss our results.

1 Introduction

Regarding the success of the Semantic Web, it may be encouraging that relevant applications and systems that utilize its standardized "toolkit" of languages and specifications tend to proliferate day by day. Even these very specifications are subject to ongoing research that attempts to push the limits of the current Semantic Web idea some steps further. Nevertheless, measuring the success of the Semantic Web could also be regarded by the point of view of the goals achieved so far: Web knowledge management; semantic resource description; and distributed knowledge discovery, as one of the most prominent. In order for this promise not to be failed, Semantic Web surely could only benefit from efficient and sound methods that would provide reasoning support for its underlying knowledge.

Description Logics and DL-based inference engines in particular play a significant role towards this goal, as they seem to have overlapping expressivity with the Semantic Web de facto language, OWL. In addition, implemented algorithms and reasoning systems for DLs already exist that could be used to provide knowledge discovery facilities on the Semantic Web. Combined, these two facts make the use of DLs one of the most tempting available formalisms to typically support reasoning with OWL.

Y. Manolopoulos et al. (Eds.): ICEIS 2006, LNBIP 3, pp. 351–362, 2008.

In this paper we first compare DL-based systems with alternatives based on other formalisms, like rule based systems and theorem provers, and argue that DLs are currently the most suitable means to build reasoning services for the Semantic Web. Then, we present and survey five popular systems from the DLs world and evaluate them in terms of their availability, expressivity and ability to reason with individuals (ABox support): Cerebra, FaCT, FaCT++, RACER and Pellet. In order to demonstrate the ability to perform Semantic Web reasoning using DL based systems, we have chosen one of the inference engines above as the core of our Knowledge Discovery Interface (KDI). The KDI is a web-distributed application that can be used to pose intelligent queries to Semantic Web documents in an intuitive manner. In order to answer these queries, the KDI relies on the reasoning services provided by the underlying inference engine. Finally we construct and use some instances of the CIDOC-CRM ontology, which we then feed in to the KDI and discuss the results from a series of intelligent queries posed.

The rest of this paper is organized as follows: In section 2 we compare available reasoning formalisms against DLs and report corresponding systems. Then, in section 3, we survey a number of DL-based systems and explain our evaluation criteria. The KDI is presented in section 4; we describe its functionality and architecture followed by some experimental results on the CIDOC-CRM ontology that demonstrate its capabilities. Finally, section 5 summarizes the conclusions from our work.

2 Inference Systems for the Semantic Web

As OWL does not natively support or suggest a reasoning mechanism, we have to rely on an underlying logical formalism and a corresponding inference engine. In this section we first briefly compare some inference methods for the Semantic Web, alternative to DLs, with DL-based systems. We present the formal relation of DLs with OWL and then discuss a number of systems based either on full First Order Logic (FOL) or rule-based systems.

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. 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 limited versions of OWL differ from DLs, in certain points, including the use of namespaces and the ability to import other ontologies.

It has been shown [6] that OWL DL can be reduced in polynomial time into the description logic *SHOIN*(D), 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 are inference systems that achieve 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 [2] used by the Wine Agent. Inference takes place using axioms reflecting 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 their need for thorough verification. In fact, there are cases where axiom construction depends on the specific contents of the ontology [9].

Another alternative is given by **rule based reasoning systems**. Such systems include DAMLJessKB [10] 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. As in 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 why such systems can presently support inference only up to OWL Lite.

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 [7] a decision procedure is presented for the *SHOIQ* description logic; this algorithm is claimed to exhibit controllable efficiency and is currently implemented in two high-end inference engines (Pellet and FaCT++).

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 DL Systems Evaluation

Having discussed the pros and cons of DLs as the underlying reasoning formalism for the Semantic Web we will now examine five inference engines based on DLs that could be used to provide reasoning services in OWL ontologies: Cerebra, FaCT, FaCT++, RACER and Pellet. Our evaluation, summarized in Table 1, is carried out in terms of their availability, expressiveness, support for OWL, reasoning with instances (ABox) and interconnection capabilities provided.

Table 1. Comparison summary of some DL-based inference engines

	Availability	Connectivity	Reasoning Strength	Native OWL support (syntax)	Reasoning with instances (ABox)
Cerebra	Commercial	RMI, SOAP	*SHIQ*	Yes	No
FaCT	Free	CORBA, DIG/1.0	*SHIQ*	No	No
FaCT++	Free	JNI, DIG/1.1	*SROIQ*(D)	No	Yes
Racer	Free (before 1.8)	TCP, DIG/1.1	*SHIQ*(D)	Yes	Yes
Pellet	Free	DIG/1.1	*SROIQ*(D)	Yes	Yes

3.1 Cerebra

Cerebra, by Cerebra Inc. (formerly Network Inference, now ingested by webMethods and absorbed in its Fabric product) is a commercial system, providing reasoning as well as ontology management features. Cerebra differs from traditional DL based systems, in that it provides some extra features that may be desirable in a production environment. Nevertheless, its expressive power is by no means exceedingly different.

Indeed, one interesting feature of Cerebra is the ability to add persistency to the knowledge bases that is able to process. Cerebra can load OWL documents either from the local file system or directly from the Web, provided the corresponding URL. The ontology information is stored, following an internal data model, in a relational database and can then be reloaded if needed.

Cerebra provides for connecting with client applications written in Java or .NET. Further, any web service may use its functionality through its SOAP interface. Clients written in Java can connect to the system either through RMI or SOAP, by using the classes provided by Cerebra for this purpose. For .NET, Cerebra provides a .dll library which can be used to connect with the SOAP interface. In both cases there is an API that provides for processing, managing and posing queries to ontologies. Query composition, especially when involving instances, follows to some extent the XQuery standard.

To our knowledge, there is no formal documentation for Cerebra's expressive power. It is known however that Cerebra's internal semantic model for conducting inferences is based on DLs. Our experimental evaluation of the system has shown that, for the taxonomic part of the ontology (the TBox), Cerebra supports nearly all constructors and axioms for classes and roles (including set-theoretic operations) that would normally classify it to OWL DL expressiveness level. However, further experimentation with the system has revealed the following:

- Symmetric roles cannot be recognized. This was confirmed as a system's bug.
- Minimum cardinality greater than 1 cannot be expressed (e.g. minCardinality=2), which is especially useful when modelling number restrictions.

- The most important, inference based on instances (ABox) is not supported. One possible exception is the instanceProperty function. However, given a class and a role, instanceProperty returns all the instance pairs that are inferred to be related through the given role, and its left argument comes from the given class.

The above rank Cerebra's expressiveness at *SHIQ* level, at most. On the other hand, the relational model used by Cerebra allows the submission of very powerful instance-retrieval queries, based on XQuery syntax. These queries may involve data types as well, like strings and numbers, as well as operands between them (equation, comparison). Still, the results are based only on the explicitly expressed information of the ontology, and not on information that could be inferred.

3.2 FaCT

FaCT [8] is a freely available reasoning software, that is being developed at Manchester University under Prof. Ian Horrocks. Initially, FaCT supported the *SHF* description logic and then evolved to include *SHIF* and finally *SHIQ*. FaCT's latest versions allow its interconnection with other applications following the client – server model through a CORBA interface. Furthermore, they support the DIG/1.0 standard, which prescribes a simple communication protocol through the exchange of XML requests and responses over HTTP.

FaCT implements optimized complete and sound algorithms to solve the subsumption problem in the description logics mentioned. Even though a pioneering system in its age, whose performance used to outrank other traditional DL systems, FaCT's lack of support for inference in the ABox renders it inappropriate for OWL. Indeed, during our evaluation we attempted to convert a simple OWL ontology to the intermediate form supported by FaCT. This conversion has been achieved using a tool available through the WonderWeb IST project, under which the next version of the system (FaCT++) is also developed. This conversion had the following results:

- Individuals are transformed into primitive concepts.
- Relations between individuals are not preserved.
- The new concepts that were created to represent individuals are now subsumed by the concepts the individuals initially belonged to.

Besides the lack of support for ABox and data types (concrete domains), the system is also syntactically incompatible with OWL. Apart from the intermediate, lisp-like knowledge base format, FaCT also supports ontologies in XML format, following a proprietary schema. Naturally however, the transition to and from OWL would result in significant information loss.

3.3 FaCT++

Many of FaCT's disadvantages are being coped with in the system's next version, FaCT++ [14], which has been developed as a part of the WonderWeb project. FaCT++ differs from FaCT in many aspects. It is a re-implementation of FaCT in C++, featuring however greater expressivity, aiming ulimately to support OWL DL.

FaCT++ currently implements the decision procedure for *SHOIQ* [7] and also the proposed rule-set extensions for OWL 1.1. These, along with its new datatype reasoning architecture, match FaCT++ expressiveness with the *SROIQ*(D) logic.

FaCT++ earlier versions (before 1.0) do not natively support XML/OWL syntax; however a transformation tool to the Lisp intermediate form supported by FaCT++ is provided. Individuals (and thus nominals) survive this transformation, but they are not yet fully supported, as they are all approximated as primitive concepts.

Recent versions appear to mimic OWL abstract syntax through an ASK/TELL manner, but full OWL syntax support is left as an implementation goal [14]. In addition the reasoner binaries do not seem to operate in batch mode or offer command-line loading features. Running the FaCT++ binary initiates the DIG server on a user-specified port.

As a C++ program, in order to support the OWL API (Jena is not yet supported), Fact++ has been extended with a Java Native Interface (JNI) [5]. Still this requires direct in-memory deployment of the reasoner.

3.4 RACER

RACER [3] 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 used to be its main asset in comparison to the other inference engines.

RACER is being developed by profs. Volker Haarslev and Ralf Moeller in Concordia University and Hamburg Technical University respectively. It is freely available for research purposes, while a corporation has been established for the commercial exploitation of the system (Racer v1.8+ aka RacerPro).

RACER's communication with other applications is achieved through the TCP/IP interface provided or through HTTP, since the system supports the DIG/1.1 standard. For TCP communication there are APIs available in C++ as well as in Java. In addition, RACER can been run in "file mode", where the ontology and queries files are given as parameters from the command line.

Apart from the lisp-like knowledge base format, RACER can load and process natively ontologies written in XML, RDF, RDFS, DAML+OIL and finally OWL (since version 1.7.7). The underlying description logic is *SHIQ*(D), including instances (ABox). In fact, RACER expressiveness is superior to OWL DL in regard to qualified number restrictions and concrete domains.

Indeed, RACER implements algorithms for conducting inferences based on min/max relations between integers, linear polynomial equalities and inequalities of reals, non-linear polynomial equations of complex numbers and string comparison. On the other hand, OWL allows only expressing equality between an individual and an instance of the concrete domain.

However, OWL semantics are more expressive than the 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 DL. RACER deals with nominals by creating a new concept for each of them and making the corresponding individual an instance of this new concept.

3.5 Pellet

Pellet [13] is a tableaux-based DL reasoner written in Java. Pellet started as a research project of MindSwap Lab at UMBC and has been the first system that implemented the decision procedure for the *SHOIQ* description logic, thus supporting OWL DL as a whole. Pellet is now open-source and is maintained by Clarck & Parsia LLC.

The reasoner may operate as a command-line standalone application and can directly load and handle OWL documents. As a Java application, it also offers and API that can be used along with other interfaces such as the Jena Toolkit and the OWL API (WonderWeb). As a distributed module, Pellet offers only a DIG/1.1 interface which has limited OWL support. Pellet libraries are also bundled with the SWOOP editor and the alpha-stage Protégé 4.0.

Pellet exhibits a sound and complete reasoning behavior with OWL DL, meaning that it can also handle nominals successfully. Also, after version 1.4, support for the rule extensions proposed with OWL 1.1 has been added. Reasoning with XML Schema datatypes is also offered, therefore *SROIQ*(D) is the underlying logic supported.

3.6 Discussion

FaCT++ and Pellet are currently the only two DL-based engines that appear to fully support the decidable subset of OWL. However they only support DIG 1.1, which is insufficient for full OWL DL support [1], a fact that mostly drives the upcoming 2.0 specification.

DIG 1.1 communication takes place over HTTP and there is no other TCP/IP-like connectivity support; rather, for these reasoners to be utilized by a tool or application, one may use a programmatic API (e.g. Jena or the Manchester API) that interfaces these reasoners as direct in-memory implementations [4]. This approach may have the advantage of reducing the message-passing load of the DIG protocol, but surely is insufficient for developing truly decentralized Web applications and services for the Semantic Web. As DIG 2.0 specification that would solve the aforementioned problems is currently in flux, these reasoners cannot be used in developing a distributed web service for Semantic Web Knowledge Discovery that would fully support OWL DL.

For the purposes of our work, we have opted for RACER as a DL-based reasoning back-end. RACER used to be dominant in terms of expressivity and interface abilities among DL-reasoners, when Pellet was not even existent. Now, RACER, being freely available for non-commercial purposes, is the only free engine, with expressive strength closest to OWL DL that exposes/maintains an independent, full-featured, IP-compatible communication interface. Indeed, its utilization in the KDI produced a number of interesting results, some of which are presented in subsection 4.2.

4 DL-Based Knowledge Discovery

In this section we demonstrate the use of DLs for knowledge discovery on the Semantic Web. First we give a general description of the KDI and the main technologies that were used, along with a brief description of its functionality. Then,

using the KDI, we present two experimental inferences on CIDOC-CRM instances expressed in OWL DL, and their results.

4.1 The Knowledge Discovery Interface

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 design follows the traditional 3-tier model, with an important variation: Where a database server would be typically used, we now use a knowledge base management system (Figure 1). Note that each of the three levels may be physically located on different computer systems.

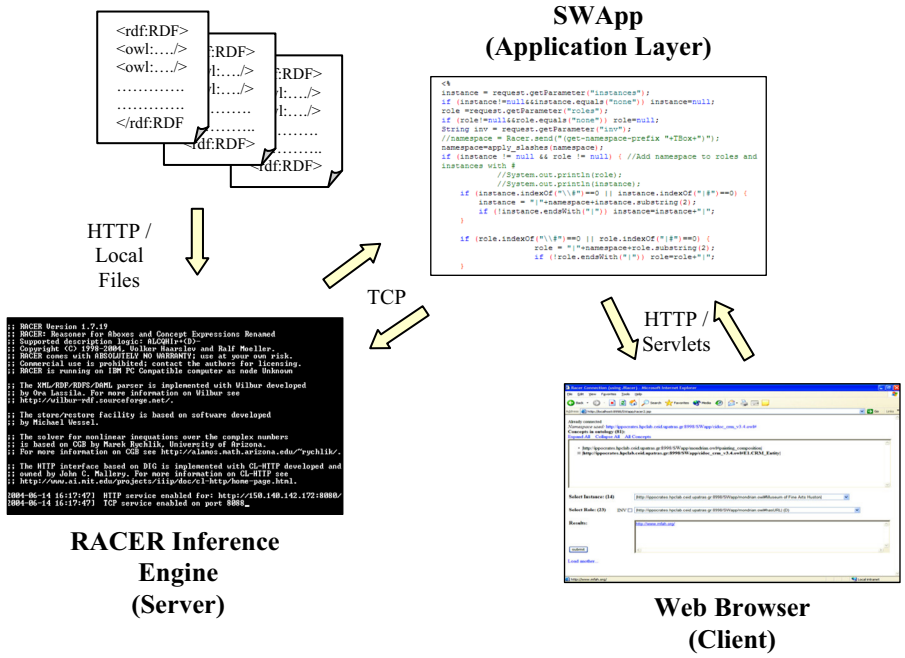

Fig. 1. The three tiers of the Knowledge Discovery Interface

The interface can load OWL documents that are available either on the local file system, or on the Internet. A temporary copy of every document is stored locally on the application server and is then loaded by the **knowledge base server** (RACER). RACER creates and stores in memory an internal model for each ontology that it classifies. Classification takes place once for each ontology, during its initial loading. Furthermore, other documents imported by the ontology may be loaded too.

The Interface business logic was implemented using the Java programming language, as well as JSP, JavaBeans and Java Servlets technologies. Tomcat (version

5.0) was used as an **application server**. Business logic is mostly responsible for document loading, proper rendering of the ontological information to the user, composition and submission of queries and formulation of the results. Ontological data and reasoning results are fetched by interacting with RACER over the TCP/IP protocol. This interaction is greatly facilitated through the JRacer API. The latter has been modified in places, mainly in regard to the processing of web documents links and to the processing of synonym concepts.

The user interacts with the **client front-end**, where the appropriate JSP pages are rendered by the browser. Communication with the application layer is conducted over the HTTP protocol, using forms. At the same time, servlets are used for the administration of multiple user requests and for controlling simultaneous access. Furthermore, when a loaded ontology is not used any more, it is erased from memory, in order to improve the utilization of system resources. For a further description of the KDI, the reader is referred to [11].

4.2 Results

In the following 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 as our knowledge base. This approach is detailed in [12].

Firstly, we ported version 3.4 of the CRM to OWL format. 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). We then 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 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>
```

The above fragment is graphically depicted in the left part of Figure 2. “Creation of Mondrian’s Composition” (i_1) is an explicitly stated “Painting_Event” that

"has_created" (***R***) "Mondrian's composition" (***i₂***). Now, asking the KDI to infer "what is a painting?" it infers that i_2 is indeed a painting (right part of Figure 2), correctly interpreting the value restriction on role R.

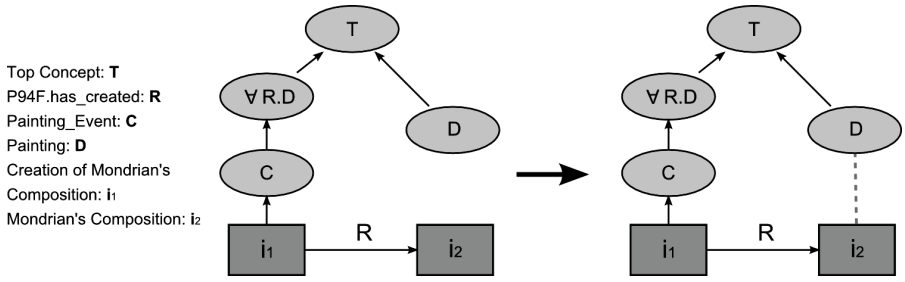

Fig. 2. Inference Example using Value Restriction

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" that its "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 3. "Mondrian's Composition" (***i₁***) is explicitly declared as a "Painting" instance which in turn is defined as a hasValue restriction on "has_type" (***R***). "Painting_composition" (***i₂***) 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?"

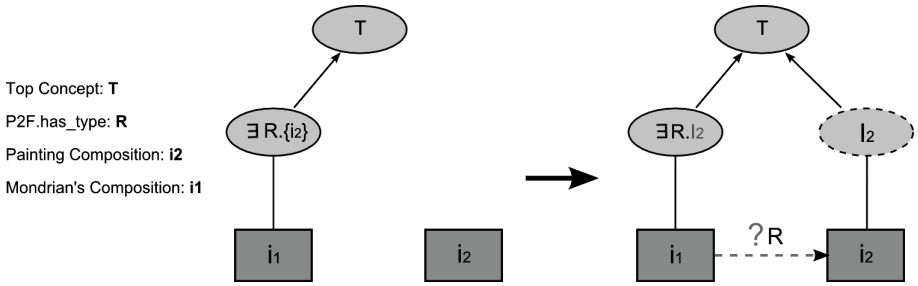

Fig. 3. Inference Example using Existential Quantification and Nominals

This example clearly demonstrates how difficult is for RACER as well as for 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.

It is notable that proceeding with a DIG interface in this case would also not solve the problem, due to its known limitations. Clearly only a direct in-memory implementation, being interfaced by the appropriate OWL API, such as the ones provided by FaCT++ and Pellet would allow a successful answer to this kind of queries.

5 Conclusions

In this paper we have primarily argued about how a well-studied logical formalism, Description Logics, can be utilized in order to enable intelligent querying of Semantic Web documents. In order to achieve this, a key step was the review of available AI formalisms and system families that could be used to ground reasoning services upon. As the scene is currently set, DL-based systems appear to be the most promising choice to achieve streamlined inference results even in the short term. At the same time, DLs show adequate compatibility and corresponding systems tend to exploit the greatest part out of the Semantic Web ontological formalism expressiveness, as it is now standardized in OWL.

We believe that our hands-on experimentation with a number of state-of-the-art DL inference engines has produced at least two lessons learned: First, the need for instance-based reasoning, which we have shown to be of crucial importance for the Semantic Web environment [11], is now becoming *sine qua non* for the majority of the systems reviewed; second, we confirmed that there are still issues, even with the most advanced DL-systems, when trying to fully support OWL's decidable expressivity.

The potential as well as the limits of the DL-based approach are clearly demonstrated through our "wrapper prototype", the KDI: On the one hand, we have succeeded in demonstrating tangible and meaningful knowledge discovery results on Semantic Web documents, with a web-distributed architecture. On the other hand, we found that the KDI is greatly hampered by the difficulty of current DL inference engines to deal with nominals or, equivalently, by the absence of suitable communication standards and implementations. We trust though that at the near future most of the difficulties and incompatibilities identified throughout our work would be overridden by the evolution of systems and the refinement of the Ontology Web Language.

Acknowledgements

Dimitrios A. Koutsomitropoulos is partially supported by a grant from the "Alexander S. Onassis" Public Benefit Foundation.

References

1. Dickinson, I.: Implementation experience with the DIG 1.1 specification. Tech. Report HPL-2004-85, Hewlett Packard, Digital Media Sys. Labs, Bristol (2004)
2. Fikes, R., Jenkins, J., Gleb, F.: JTP: A System Architecture and Component Library for Hybrid Reasoning. In: Proc. of the 7th World Multiconference on Systemics, Cybernetics, and Informatics (2003)
3. Haarslev, V., Möller, R.: Racer: A Core Inference Engine for the Semantic Web. In: EON 2003. Proc. of the 2nd International Workshop on Evaluation of Ontology-based Tools, pp. 27–36 (2003)
4. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 Touch Paper: The OWL API. In: OWLED 2007. Proc. of the OWL Experiences and Directions Workshop (2007)
5. Horridge, M., Tsarkov, D., Redmond, T.: Supporting Early Adoption of OWL 1.1 with Protégé-OWL and FaCT++. In: OWLED 2006. Proc. of the OWL Experiences and Directions Workshop (2006)
6. Horrocks, I., Patel-Schneider, P.F.: Reducing OWL entailment to description logic satisfiability. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 17–29. Springer, Heidelberg (2003)
7. Horrocks, I., Sattler, U.: A tableaux decision procedure for SHOIQ. In: IJCAI 2005. Proc. Of the 19th Int. Joint Conf. on Art. Intelligence, Morgan Kaufm., Seattle (2005)
8. Horrocks, I., Sattler, U.: Optimised reasoning for SHIQ. In: ECAI 2002. Proc. of the 15th Eur. Conf. on Artificial Intelligence, pp. 277–281 (2002)
9. Hsu, E., McGuinness, D.: Wine Agent: Semantic Web Testbed Application. In: Proc. Of Workshop on Description Logics (2003)
10. Kopena, J., Regli, W.C.: DAMLJessKB: A tool for reasoning with the Semantic Web. IEEE Intelligent Systems 18(3), 74–77 (2003)
11. Koutsomitropoulos, D.A., Fragakis, M.F., Papatheodorou, T.S.: Discovering Knowledge in Web Ontologies: A Methodology and Prototype Implementation. In: Proc. of SEMANTICS 2006, OCG, pp. 151–164 (2006)
12. Koutsomitropoulos, D.A., Papatheodorou, T.S.: Expressive Reasoning about Cultural Heritage Knowledge Using Web Ontologies. In: WEBIST 2007. Proc. of 3d Int.Conf. on Web Information Systems and Technologies, WIA track, pp. 276–281 (2007)
13. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5(2) (2007)
14. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, Springer, Heidelberg (2006)