Developing rule-based applications for the Web: Methodologies and Tools

Vassilis Papataxiarhis, Vassileios Tsetsos, Isambo Karali, Panagiotis Stamatopoulos, Stathes Hadjiefthymiades

Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, GR-15784, Athens, Greece,
{vpap, b.tsetsos, izambo, takis, shadj}@di.uoa.gr

ABSTRACT

Embedding rules into Web applications, and distributed applications in general, seems to constitute a significant task in order to accommodate desired expressivity features in such environments. Various methodologies and reasoning modules have been proposed to manage rules and knowledge on the Web. The main objective of the chapter is to survey related work in this area and discuss relevant theories, methodologies and tools that can be used to develop rule-based applications for the Web. The chapter deals with both ways that have been formally defined for modeling a domain of interest: the first based on standard logics while the second one stemmed from the logic programming perspective. Furthermore, a comparative study that evaluates the reasoning engines and the various knowledge representation methodologies, focusing on rules, is presented.

1 INTRODUCTION AND MOTIVATION

Nowadays, with the evolution of traditional web of documents to a more complex web of services, an increasing demand for embedding intelligence to Web applications arises. In this context, the efficient management of knowledge seems to play a key role in order to achieve smart behavior of Web applications and to overcome several issues of such environment (e.g., information integration). Ontologies, mainly written with Semantic Web technologies, constitute a well-established paradigm for representing knowledge on the Web. Though, current efforts are focused on extending ontologies with more expressive forms of knowledge like rules. In fact, given the state-of-the-art in the realization of the Semantic Web vision, rules constitute the next prominent challenge. Since the ontology layer of the Semantic Web architecture stack has reached a sufficient degree of maturity through Web Ontology Language (OWL) (Dean et al., 2004), the next step of progress involves the integration of rules with ontologies, most of them based on subsets of First Order Logic (FOL).

Rules are capable of extending the expressiveness provided by ontology languages through the definition of more complex relationships between individuals. Additionally, as a modular form of knowledge, they fit well in domains like personalization, policies and business-to-business (B2B) interaction. However, it has been shown that extending ontologies even with simple forms of rules can lead to undecidability of key inference problems.

On the other hand, many business-logic applications have extensively taken advantage of existing rule management systems or solvers (Jess, 2008; ILOG, 2008; Drools, 2008), aiming at facilitating the knowledge management process. As a result,

the success of rules in non-Web applications moved Web researchers to use traditional rule engines on the Web.
However, the aforementioned stable rule systems have not been originally created for open and heterogeneous environments like the Web. Such platforms have adopted different knowledge representation formalisms, mainly based on principles of logic programming, instead of classical logic. As a consequence, they differ from recent Semantic Web technologies in many aspects, including representational features and reasoning functionality, as well. Hence, building a rule-based application for the Web with existing rule technologies is not a straightforward task.
In the rest of this chapter, we provide foundational knowledge on this topic together with implementation issues, techniques and design patterns. Section 2 briefly describes how the things have gone so far in the area of Web knowledge formalisms. In Section 3, various knowledge representation methodologies and tools are discussed. Specifically, Section 3.1 demonstrates the different languages and formalisms, derived from both classical logic and logic programming view, while Section 3.2 focuses on various engines able to reason over such knowledge bases. Section 4.1 gives the main requirements for rule-based web applications. The evaluation presented includes both a qualitative comparison (Section 4.2) of the existing approaches and a performance analysis (Section 4.3) of current ontology reasoners and rule engines. Finally, several future trends and open issues are identified in Section 5. Hence, this chapter aims at becoming a helpful guide for applying rules to Web applications.

2 THE STORY SO FAR

The knowledge representation languages proposed (see Section 3.1) for representing knowledge on the Web are based either on the Classical Logic (CL) perspective or on Logic Programming[1] (LP). As a result, a debate was started between the Database community and AI researchers, respectively, in order to determine the more suitable of the two approaches in the formalization of Web knowledge. Additionally, different languages of the same perspective, providing various degrees of expressivity, have been proposed. Hence, the integration of knowledge with Web applications was more complicated. Recently, with the evolution of Semantic Web technologies, these modeling paradigms have been extensively discussed by researchers (Motik, Horrocks, Rosati, & Sattler, 2006; Patel-Schneider & Horrocks, 2006; Eiter, Ianni, Polleres, Schindlauer, & Tompits, 2006a; Donini, Lenzerini, Nardi, & Schaerf, 1998; Antoniou, et al., 2005; Boley, Kifer, Patranjan, & Polleres, 2007).
Clearly, this confrontation also affected rules. Although, significant efforts have been devoted in order to develop appropriate rule languages for the Web (see Section 3.1), a debate between proponents of the different perspectives has appeared. The main argument involves the degree that rules would be combined with ontologies. Initially, as formulated by Tim Berners-Lee, it had been commonly accepted that Semantic Web should be structured in a hierarchy of layers that seamlessly interoperate (Antoniou & van Harmelen, 2004). However, some proposals (mainly stemmed from the database community) preferred to keep the rules and the ontology layer separated, in order to preserve expressiveness and decidability of reasoning process, as mentioned in (Eiter et al., 2006a). Specifically, in (Kifer, de Bruijn, Boley, & Fensel, 2005) the authors argue that the rules layer of Semantic Web architecture should be placed next to the ontology layer of the Semantic Web stack in order to take

advantage of important rule languages based on logic programming. However, in (Horrocks, Parsia, Patel-Schneider, & Hendler, 2005) the authors criticize the aforementioned approach and denote that such a distinction would lead to two Semantic Webs based on different semantics. These discussions along with the Semantic Web Activity workgroup have led to several modifications of the stack. Figure 1 presents the initial and the current form of the Semantic Web stack diagram specified by the W3C Semantic Web Activity Group (W3C Semantic Web Activity, 2008).

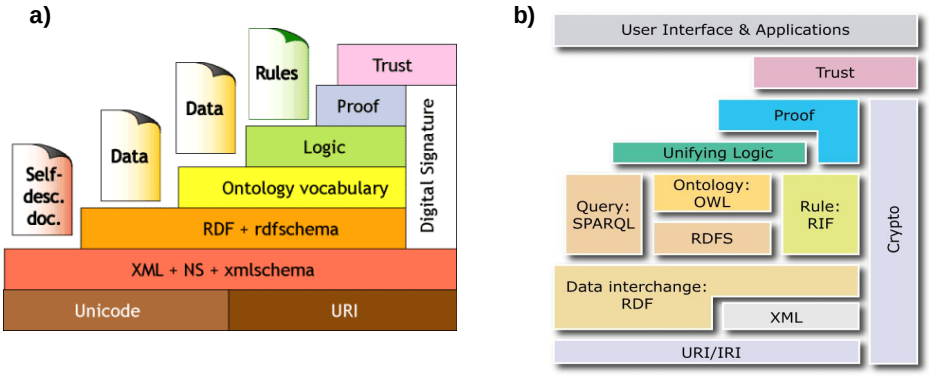

Figure 1. a) Tim Berners-Lee Semantic Web layer "cake" and b) the latest form of Semantic Web stack diagram (W3C Semantic Web Activity, 2008)

The aforementioned facts gave rise to confusion inside the Web community. If the experts (i.e., the logicians stemming from both AI and databases) could not decide and recommend a unified framework for formalizing knowledge, how Web developers and users should choose the best language according to their needs? The current solutions are either to study the capabilities provided by all these languages (i.e., available expressiveness, support by existing reasoning engines, tractability etc.) or to exclude knowledge from their applications.

3 KNOWLEDGE REPRESENTATION METHODOLOGIES AND TOOLS

This section surveys different knowledge representation methodologies along with reasoning modules that have been proposed for managing knowledge on the Web. After an overview of Description Logics that comprise the main formalism for representing ontological knowledge in Web applications, we focus on rules.

3.1 Knowledge Representation Formalisms

Description Logics

Description Logics (DLs) (Baader, Calvanese, McGuiness, Nardi, & Patel-Schneider, 2003) are subsets of first-order-logic, originating from semantic networks and frame-based systems. They constitute a family of knowledge representation languages that aim at providing well-understood mechanisms in order to formalize knowledge that describes a domain. This way, DLs are equipped with formal, logic-based semantics, emphasizing on reasoning process. Typical reasoning tasks include consistency checking of the knowledge base, concept satisfiability, instance checking etc.

A DL-based knowledge base is composed of two components: TBox and ABox. TBox contains the vocabulary of the application domain, called *terminology*, as well as axioms based on that vocabulary. Practically, such vocabulary consists of *concepts* and *roles*. Concepts are generic descriptions of sets of individuals, while roles constitute binary predicates for defining properties of the individuals. On the other hand, ABox includes assertions of individuals that may refer to either concepts or roles. For example, a statement declaring that a specific individual is instance of a concept resides in ABox, while a statement denoting that "every human is mortal" belongs to TBox. Figure 2 shows the generic architecture of a DL knowledge representation system.

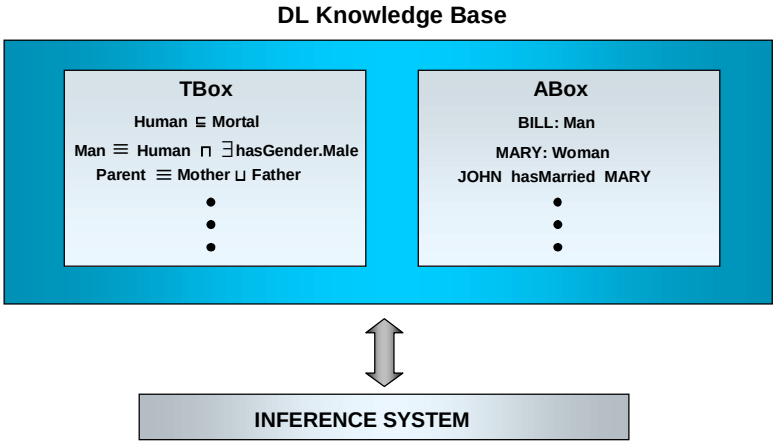

Figure 2: Architecture of a Description Logics knowledge-based system

Each description logic language is determined by a set of constructs, enabling the use of atomic concepts and atomic roles in order to define complex ones. These constructs directly affect the expressive power of the language and, thus, the complexity of inference tasks. As a result, the selection of the appropriate description logic language in order to describe a specific domain includes the examination of the imposed requirements for representation expressiveness.

DLs set up the base for the definition of Resource Description Framework (RDF) (Klyne & Caroll, 2004), RDF Schema (Brickley & Guha, 2004) and Web Ontology Language (OWL) (Dean et al., 2004) that constitute W3C standards for representing knowledge on the Web. Specifically, RDF is a simple data model for representing information on the Web. RDF statements are expressed in the form of (*subject, predicate, object*) triples. A set of such statements can be viewed as a graph where the subjects and the objects of these statements constitute the graph nodes while the predicates correspond to the graph edges. RDFS is an extension of RDF for expressing simple taxonomies through the definition of class/property hierarchies and domain/range of properties. OWL is totally based on DLs and it comes in three species: OWL-Lite, OWL-DL and OWL-Full. OWL-Full is the most expressive OWL species, since it takes advantage of all the OWL language primitives. OWL-DL limits OWL-Full to a subset of OWL primitives so as to achieve efficiency of reasoning. Finally, OWL-Lite is a sublanguage of OWL-DL by restricting more its expressiveness.

Description Logic Programs

The expressiveness of Description Logic Programs (DLP) approach (Grosof et al., 2003) corresponds to a fragment of OWL defined by the expressive intersection of

Description Logics and logic programming. This approach intends to define a mapping from DL to logic programming (specifically, Horn programs in which no function symbols, negation and disjunction are allowed) and vice versa. An instance of such a mapping in the case of conjunction follows:

$$C_1 \sqcap C_2 \sqsubseteq D \quad \equiv \quad D(x) \leftarrow C_1(x) \wedge C_2(x) \qquad (1)$$

However, for the sake of decidability, DLP offers limited expressiveness, since the aforementioned mapping covers only a few DL-constructs (in particular, conjunction, disjunction and quantification restrictions). For example, Description Logic Programs do not cover negation in class descriptions nor fully support cardinality restrictions.

Semantic Web Rule Language

Semantic Web Rule Language (SWRL) (Horrocks et al., 2004b) is probably the most popular formalism in Web community for expressing knowledge in the form of rules. Specifically, SWRL is based on a combination of Web Ontology Language (OWL) (Dean, 2004) and Rule Markup Language (RuleML) (RuleML, 2008) and has been proposed as a W3C candidate standard for formalizing the expression of rules in Web context. Contrary to DLP, SWRL extends OWL-DL with a specific form of Horn-like rules.

The proposed rules are in the form of an implication between the body and the head of the rule. A typical SWRL rule can be of the following form:

$$a_1 \wedge a_2 \wedge \ldots \wedge a_n \rightarrow b_1 \wedge b_2 \wedge \ldots \wedge b_m \qquad (2)$$

where a_i and b_i are OWL atoms of the following forms:

- Concepts, e.g., C(x), where C is an OWL description, in general, and x is either a variable, an OWL individual or a data value.
- Object properties, e.g., P(x,y), where P is an OWL property and x, y are either variables, individuals or data values.
- Datatype properties, e.g., P(x,y), where P is an OWL property, x is variable or individual, while y is a data value.
- B(x1,x2,…), where B is a built-in relation and x1, x2, … are either variables, individuals or data values.
- sameAs(x,y) or differentFrom(x,y) where x, y are either variables, individuals or data values.

The main advantage of SWRL is the simplicity it offers, while extending the expressiveness of OWL. Another benefit of SWRL is its compatibility with OWL syntax and semantics, since they are both combined in the same logical language. On the other hand, extending OWL-DL with SWRL rules leads to undecidability of simple inference problems. A possible solution of this problem is presented in (Motik, Sattler, & Studer, 2005) which introduces the notion of *DL-safe* rules. Specifically, that approach restricts the application of SWRL rules only to individuals of the ABox part of the DL knowledge base. Moreover, SWRL does not support negation (neither classical nor negation as failure - NAF) and disjunctions. Finally, there is no efficient support of first-order provers to execute reasoning over SWRL. Usually, the SWRL rules are translated to existing rule systems (e.g., Jess (O'Connor, Knublauch, Samson, & Musen, 2005)) that handle the reasoning tasks partially, since they are not aimed to manage knowledge expressed in terms of first-order logic or subsets.

In the case of a Web service composition paradigm, some SWRL example rules could be the following:

$$\text{profile:hasOutput(?S1,?out)} \wedge \text{profile:hasInput(?S2,?in)} \wedge \text{process:parameterType(?in, ?cin)} \wedge \text{process:parameterType(?out, ?cout)} \wedge (\text{rdfs:subClassOf(?cout, ?cin)} \vee \text{rdfs:subClassOf(?cin, ?cout)}) \rightarrow \text{composableWith(?S1, ?S2)} \quad (3)$$

The abovementioned rule captures the knowledge that a web service S1 is composable with a service S2, if an input of S1 is either subclass or superclass of at least one output of S2. This rule uses specific namespaces of OWL-S ontology like "profile" and "process".

Answer Set Programming

Answer Set Programming (ASP) (Gelfond & Lifschitz, 1991) is a paradigm for knowledge representation and declarative programming. It has several advantages compared to other logic programming paradigms (e.g., Prolog) such as:

- *Full declarativity*: the order of rules in a program is not important.
- *ASP programs are in general decidable*
- *Non-monotonic inference*: both negation as failure (NAF) and strong negation are supported, thus enabling default reasoning and reasoning under the Closed World Assumption (CWA).
- *Availability of efficient solvers*: there are several ASP solvers that are scalable enough to deal with large knowledge bases.

A general ASP rule is of the following form:

$$a_1 \vee a_2 \vee \ldots \vee a_n \leftarrow b_1 \wedge \ldots \wedge b_k \wedge \text{not } b_{k+1} \wedge \ldots \wedge \text{not } b_m \quad (4)$$

where a_i and b_j are literals (atoms or strong negations of atoms) and *not* denotes NAF.

A set of such rules is an ASP program. What is interesting is the fact that these rules can have disjunctions in their head. This is a very important feature of ASP, since it introduces non-determinism in the inference process (i.e., an ASP program may have several models which are called *answer sets*).

In order to be able to use ASP on the Web, the ASP rules should be combined with Web knowledge. Since ontologies is the most common way to represent knowledge on the Web, an interaction between ASP programs and ontologies is deemed necessary. A solution to this integration problem is description-logic programs (or dl-programs). These consist of ASP rules that may contain queries to DL knowledge bases. For example, the following rule "brings" into the ASP program all instances of the class MovieTitle in the DL knowledge base:

$$\text{movie(X)} \leftarrow \text{DL["Movie"](X).} \quad (5)$$

Several extensions to dl-programs were proposed (and implemented) so that they become more "suitable" for open environments like the Web, where information may be expressed in many diverse ways (e.g., multiple different ontologies). The most well known extension is HEX-programs (Eiter, Ianni, Schindlauer, & Tompits, 2005), which enable handling knowledge expressed in various formalisms, even with potentially different semantics (e.g., RDF(S) and OWL). HEX-programs contain several features (e.g., higher-order logic features) that enable more flexible integration

with external knowledge bases. These extensions result in new syntax elements. For example, the atom *&rdf[u](s,p,o)* evaluates to true if <s p o> is an RDF triple asserted at URI u. Finally, another advantage of HEX-programs is that they allow using external data processing services that logic programming cannot handle (e.g., string processing). An interesting engine for HEX-programs is dlvhex[2] which is described in Section 3.2.

There have been proposed ASP-based several applications in the context of (Semantic) Web. One of the most promising is Web service composition (Rainer, 2005). The authors apply ASP techniques to "build" service compositions from available services that match a certain service request.

Web Service Modeling Language (WSML)

WSML (de Bruijn et al., 2005) is a language of representation languages for the Semantic Web. These languages are based on several different formalisms such as Description Logics, Logic Programming and First-Order Logic. Some of the basic variants of WSML are the following:

- WSML-Core: A subset of a Description Logic which falls inside the Horn logic fragment of FOL. It supports subsumption reasoning and query answering.
- WSML-Flight: An extension of WSML-Core which also supports full Datalog rules, default negation and integrity constraints. It can provide query answering in the context of Logic Programming.
- WSML-Rule: An extension of WSML-Flight with support for function symbols and unsafe rules.

Reasoning for these WSML variants can be implemented by several Logic Programming engines. For some features of these languages, DL reasoners can also be used or First Order theorem provers. These languages have been extensively used in several European projects, mainly in the application domain of semantic web services. Hence, several APIs, tools and other facilities are available for building WSML-enabled applications.

Defeasible Rules

Defeasible logic (Nute, 1994) is a rule-based, non-monotonic approach able to deal with incomplete knowledge and inconsistencies. These features have been widely remarked in the context of realizing Semantic Web vision, mostly in information integration areas (e.g., ontology merging). As a result, some efforts in research community (Antoniou, Billington, Governatori, & Maher, 2001) were devoted to carry the advantages of defeasible logic in the area of Semantic Web technologies.

The main idea behind defeasible logic reasoning systems is the ability to handle a number of additional features with regard to classical rules like priorities of rules, default inheritance, exceptions, etc. There are three different types of rules in a defeasible logic reasoning system: a) classical rules (called *strict rules*), b) *defeasible rules* that can be contradicted by other rules and c) *defeaters* used to specify exceptions of defeasible rules. This way, an important aspect achieved by such reasoning modules is their capability of resolving the possible conflicts that arise among defeasible rules.

Classification of Approaches Integrating Ontologies and Rules

Although several approaches have been discussed for combining rules with Semantic Web ontologies (Horrocks et al., 2004b; Grosof, Horrocks, Volz, & Decker, 2003; Eiter, Lukasiewicz, Schindlauer, & Tompits, 2004; Bassiliades, Antoniou, & Vlahavas, 2006; Rosati, 2006a; Rosati, 2006b), there is no totally accepted solution in the field. The main topic of argumentation is the degree of integration between the ontology layer and the rules layer. In this section we intend to provide a brief classification of the proposed approaches.
Two main categories of integration approaches have been distinguished in this context:
a) Homogeneous approaches. These approaches suppose a tight semantic integration of the two layers. Specifically, both ontologies and rules are embedded in a common logical language, permitting predicate sharing in a coherent way. In such approaches, ontology concepts and properties may be defined through rules. The most typical homogeneous paradigm is the combination of SWRL rules with OWL ontologies. This is also the most In addition, Description Logic Programs (DLP) (Grosof et al., 2003) constitutes another similar approach.
b) Hybrid approaches. These approaches correspond to a strict semantic separation between ontologies and rules. In particular, this strict separation concerns the rule predicates and the ontology elements. Hence, the vocabulary (concepts and properties) offered by the ontologies is used as a conceptualization of the domain and rules cannot directly define ontology classes or properties. Many integration approaches adhere to this category, including Answer Set Programming (Gelfond & Lifschitz, 1991), *dl-programs* (Eiter et al., 2004) and *DL+log* (Rosati, 2006c).
The user may find more details on this topic in (Eiter et al., 2006a).

Table 1 summarizes the aforementioned knowledge representation languages and their basic features.

Table 1: Basic features of various knowledge representation languages

Feature / Language	Logical Foundation	Decidability*	Serialization formats
OWL	Classical Logic (FOL subset)	OWL-Lite: decidable OWL-DL: decidable OWL-Full: undecidable	XML/ N-triples (textual)
OWL + SWRL	Classical Logic (FOL subset)	undecidable	XML
DLP	DL and LP intersection	decidable	textual serialization (in terms of rules)
ASP	Extension of LP (disjunction in	decidable	textual serialization (in terms of rules)

	rule heads, DL queries)		
DL+log	DLs + Datalog rules (disjunctive, non-monotonic)	decidable	textual serialization (in terms of rules)

*regarding key inference problems (e.g., consistency of the knowledge base)

3.2 Reasoning Engines

In this section we intend to provide the reader with a comparative feature analysis of existing reasoning modules, including description logic reasoners and rule engines, as well.

Jena2 (McBribe, 2002) is the second generation of Jena Semantic Web programming toolkit, which is a Java framework for developing applications based on Semantic Web technologies. Specifically, Jena provides an Application Programming Interface (API) for creating, storing, managing and querying RDF graphs as well as RDFS, OWL ontologies in various formats (RDF/XML, N3 and N-triples). The RDFS reasoner included in Jena framework does not support datatypes and blank node entailments. The built-in OWL reasoner is very limited, since it is a rule-based implementation of OWL-Lite. However, Jena is supplied with an interface which facilitates the connection and interoperation of the framework with any external reasoner that supports the DIG (DL-Implementation Group) standard (Bechhofer, Moller, & Crowther, 2003). Hence, the API provided by Jena could be integrated with most of the existing description logics reasoners. Furthermore, Jena provides a query engine in order to execute SPARQL (Prud'hommeaux & Seaborne, 2005) queries over RDF graphs.

RacerPro system (RacerPro, 2008) is the commercial[3] extension of Racer (Haarslev & Moller, 2001), which is probably the most popular reasoning engine for OWL ontologies to practitioners of Semantic Web technologies. RacerPro can be seen as a knowledge-based repository that can handle and it is a system for managing OWL ontologies, in particular. It implements the DIG interface and it offers an optimized tableau calculus for the description logic SHIQ(D). Additionally, it supports qualified cardinality restrictions as well as some extensions of OWL (e.g., OWL-E (Pan & Horrocks, 2004) except user defined XML datatype expressions). Moreover, the latest version of RacerPro includes a first implementation of an SWRL rule engine.

Pellet (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007) is a Java-based, open source reasoner capable of handling expressive OWL ontologies. It implements an optimized tableau algorithm, augmented with a number of additional features (e.g., support for Unique Name Assumption - UNA, closed world reasoning, SPARQL query answering). Pellet also provides an explanation service in order to facilitate the debugging of the ontology engineering (Parsia, Sirin, & Kalyanpur, 2005). Although typical reasoners detect the inconsistencies between concepts of the knowledge base (KB), Pellet can explain *why* a concept description led to unsatisfiability. This way,

the reasoner provides user with additional knowledge (e.g., relevant axioms or restrictions) sufficient to understand the problem and reform the KB properly. Finally, Pellet allows ontologies to use XML-Schema built-in and user-defined datatypes that extend numeric and date/time types.

Bossam (Jang & Sohn, 2004) is a RETE-based, forward-chaining reasoning module for reasoning and querying over RDF(S) and OWL documents, while it also supports rules execution (SWRL rules are included). It is based on Logic Programming (LP), augmented with some expressiveness features stemming from First-Order-Logic (FOL). For example, a number of additional to LP features are provided by Bossam, including support for both classical negation and NAF and disjunctions in the body of rules. Hence, it does not support complete reasoning over OWL Ontologies. Furthermore, it facilitates the integration of rules with Java by supporting a procedural attachments mechanism for SWRL rules. Finally, Bossam provides an API for managing the engine, loading ontologies and rules, querying RDF(S)/OWL documents and giving explanations about derived facts. Currently, Bossam does not support SPARQL query answering, while the serialization of the knowledge base to a persistent store (e.g., file system) is another missing feature.

FACT++ system (Tsarkov & Horrocks, 2006) is the descendant of FACT (Horrocks, 1998). Contrary to the lisp-based FACT system, FACT++ is an open source reasoner for SHOIQ(D) implemented in C++. It is based on tableaux algorithms in order to provide both TBox and ABox reasoning tasks and it can be accessed through the DIG interface. It also supports a number of additional features like handling enumerated classes (a.k.a. nominals) and it is a very efficient TBox reasoning engine. However, the main disadvantage of FACT++ is its inefficiency to support complete ABox reasoning. Hence, FACT++ is unsuitable to applications that call for instance classification and retrieval.

KAON2 (KAON2, 2008) is the successor of KAON Project (KAON, 2008) and unlike pure description logic reasoners, KAON2 does not implement a tableaux algorithm. In fact, KAON2 is a hybrid reasoning module able to handle both ontologies expressed in description logics terms and Disjunctive Datalog programs. It implements algorithms that reduce description logic SHIQ(D) to Disjunctive Datalog (Hustadt, Motik, & Sattler, 2004), taking advantage and applying well-known practices stemming from deductive databases (e.g., magic sets) to DL reasoning. It can also handle SWRL, F-Logic ontologies and SPARQL query answering, as well. Moreover, it provides a Java API in order to accommodate the management and the integration of different knowledge formalisms (e.g., OWL ontologies with SWRL rules). However, KAON2 does not support reasoning about nominals and cannot handle a large number of cardinality restrictions. Hence, it cannot deal with the full OWL-DL expressiveness.

dlvhex (Eiter, Ianni, Schindlauer, & Tompits, 2006b) is the name of a prototype application for computing the models of so-called HEX-programs, which are an extension of Answer Set Programs towards integration of external computation sources. dlvhex can communicate with OWL and RDF knowledge bases and can also return the results in RuleML syntax. A strong point of dlvhex is that it enables developers to write and embed plug-ins to the core engine. In fact, support for RDF and OWL is implemented through plug-ins too. Recently, another plug-in for

querying HEX models through the SPARQL language (Polleres & Schindlauer, 2007). Through this plug-in dlvhex can be used as a query engine by providing a rewriter from SPARQL to rules. The source code and binaries of dlvhex are publicly available.

DR-DEVICE (Bassiliades, et al., 2006) is a defeasible logic reasoner for the Web based on the CLIPS expert system shell (CLIPS, 2008) that intends to integrate Semantic Web standards (RDF metadata, XML-syntax of rules) with non-monotonicity (e.g., strong negation). Specifically, it provides reasoning services over RDF metadata by taking advantage of rules defined by defeasible logic (strict rules, defeasible rules and defeaters). It also claims for reasoning efficiency compared to other systems based on logic programming.

Jess (Jess, 2008) is a Java framework for editing and applying rules, since it contains a scripting environment and a rule engine, as well. It supports a CLIPS-like language suitable for developing applications based on declarative rules (a.k.a. expert systems). Jess also uses an optimized version of Rete algorithm (Forgy, 1982) tailored for Java, comprising a very efficient rule engine. Recently, the evolution of rule technologies on the Web has led Jess to rebound its practical value in the community of Web developers. Moreover, the fact that Jess is a Java-based system facilitates its integration with a number of Web programming paradigms like Java servlets or applets. Finally, it supports backward-chaining and some additional features like procedural attachments.

Table 2 presents the types of inference support provided by the aforementioned reasoning modules. In (Cardoso, 2007), an analysis of the current trends and the adoption of available reasoning engines by the Semantic Web community are presented in detail.

Table 2: Types of inference support by various reasoning engines

Inference Support / **Modules**	**TBox Reasoning**	**Abox Reasoning**	**Rules Reasoning**
Jena2	Limited (incomplete RDFS/OWL reasoning)	Limited (incomplete RDFS/OWL reasoning)	JenaRules (forward/tabled backward chaining)
RacerPro	√	√	nRQL rules, first implementation of SWRL (forward chaining)
Pellet	√	√	DL-safe rules (SWRL subset)

Bossam	sound, incomplete	sound, incomplete	SWRL, Buchingae (forward chaining)
FACT++	√	sound, incomplete	-
KAON2	√ (except nominals)	√ (except nominals)	DL-safe rules (SWRL subset)
dlvhex	Limited, through interface with external reasoner	Limited, through interface with external reasoner	DL-Rules (non-monotonic logic program rules with queries to DL KB)
DR-DEVICE	-	-	Defeasible rules
Jess	Limited*	Limited*	SWRL*, Jess rules (forward/backward chaining)

* through appropriate transformations (Mei, Bontas & Lin, 2005; O'Connor et al., 2005)

4 EVALUATION OF EXISTING APPROACHES

In this section, a comparative study across the aforementioned methodologies for introducing rules in Web applications is presented. Firstly, we give some fundamental requirements that have to be satisfied by such formalisms. Afterwards, we compare various aspects from the perspectives of classical logic and logic programming, giving indicative examples. Finally, some experimental results that examine the efficiency of current efforts in the combination of ontologies with rules are presented.

4.1 Rule-based Web Applications are Still "Web Applications"

Before evaluating the existing approaches to introducing rules in Web applications, we should identify the main requirements for such approaches. These requirements mainly stem from the nature of the Web itself and its current status. Firstly, WWW is a ubiquitous and massive multi-user distributed environment. In fact, this massive characteristic is its strong point, and is a direct consequence of its architectural and technological simplicity. Web technologies such as HTTP, HTML and XML are very simple to learn and use, even for plain users who are not IT experts. Such simplicity should be taken as granted for any Web-related technology, and this applies to rule

systems as well. In this context, rules for the Web should be written and used even by users/developers not familiar with advanced knowledge engineering concepts. For example, negation is a rather advanced topic in logic-based systems (with Horn rules). Hence, the semantics of rules should be such that can be easily understood by naïve users. However, this would affect their expressiveness and would limit the inference power of rule-based systems that may be required by more demanding applications (i.e., Web service discovery engines). It is worthy to mention that most of the existing rule-based applications for the Web have adopted SWRL approach (see Section 3.1) in order to express rules. SWRL is neither a highly expressive language (e.g., no negation is available) nor a decidable one, but it remains simple.

Another requirement is that the rule-based systems for the Web are compatible with existing (Semantic) Web standards, such as XML, RDF, OWL and SAWSDL (Farrell & Lausen, 2007). This means that direct support for URIs or XML syntax should be available. On the other hand, writing rules in XML (e.g., like the SWRL and RuleML approaches propose) is a cumbersome task. This is indeed a difficult decision that has to be taken by the Web architects.

Furthermore, the addition of rules in Web applications requires their compatibility with existing forms of Web knowledge that have already attained a certain maturity level. Most of the knowledge bases developed in the context of Web are expressed in the form of ontologies (mainly written in OWL or RDFS). As a result, rules have to be integrated with ontologies properly.

4.2 Qualitative Comparison of Classical Logic and Logic Programming approaches

The aforementioned knowledge representation methodologies (see Section 3.1) are either based on classical logic (e.g., DLs and SWRL) or logic programming (e.g., answer-set programming and defeasible rules). Across the literature, several differences between classical logic and logic programming paradigms have already been identified. In this section, we intend to survey and discuss these efforts by presenting the main incompatibilities that impede the reconciliation of the two approaches.

Monotonicity vs. Non-monotonicity

Classical logic is based on standard model theoretic semantics and adheres to monotonicity of entailment. Informally, monotonicity means that the addition of new information to the knowledge base cannot invalidate conclusions inferred by current knowledge. As a result, classical logic is able to deal with incomplete information by nature.

Instead of classical logic, logic programming has non-monotonic features. It assumes complete knowledge and there is a unique model describing the state of the world. This way, addition of knowledge may reduce the inferences.

Unique Name Assumption

Another difference between the two paradigms is that logic programming approaches typically deploy the *Unique Name Assumption* (UNA). This assumption supposes that different names represent different objects of the world. This fact imposes severe limitations on the Web context, since several distinct URIs[4] may refer to the same content or data. On the contrary, in classical logic there is no one-to-one correspondence between the names and the objects of the domain. Hence, equality

between individuals represented by different names can be derived. Although it imposes a huge computational cost, most of the current description logic reasoners support reasoning with equality.

Negation

Classical logic and logic programming face the aspect of negation from different perspectives. With respect to its monotonic nature, negation in classical logic allows inferring new information only if the truth or the falsehood of a statement is explicitly declared. This fact is related with the *Open World Assumption* (OWA) of classical logic theory where incompleteness of the knowledge base is considered.

On the other hand, *Negation-As-Failure* (NAF) adheres to the *Closed World Assumption* (CWA). Specifically, if a description is not known to be true, then the truth of the negated description is drawn. In that sense, absence of knowledge draws to negated knowledge and, thus, NAF has a non-monotonic behavior. Modeling the world according to CWA seems to be somewhat inappropriate for the Semantic Web. Since knowledge on the Web is not always available (e.g., web servers breakdowns), such an assumption could lead to incorrect inferences. However, the usefulness of different types of negation in rule-based Web applications is demonstrated in (Wagner, 2003; Alferes, Damasio, & Pereira, 2003).

Constraints and Restrictions

Another aspect where the classical paradigm differs from logic programming is on the treatment of *constraints* and *restrictions*. Regardless of the apparent similarity between constraints and restrictions, there are important differences between them.

Restrictions are used in the classical logic paradigm and they constitute part of the logical theory. By adding restrictions to a knowledge base, the knowledge statements are also increased and, thus, the inference of additional knowledge is permitted.

Restrictions may be further classified to the following main categories:

- *Value restrictions* or *range restrictions*. They are used to restrict the values of a property and possibly infer new information according to their type (a.k.a. range).
- *Cardinality restrictions*. They are used for restricting the number of values that an individual may have for a specific property. They can possibly infer the existence of new instances or equality between known individuals.

In terms of Description Logics, an example of cardinality restriction in the web service paradigm could be the following:

$$\text{Profile} \sqsubseteq \geq 1 \text{ hasInput} \tag{6}$$

Such rule denotes that every service profile should have at least one input.

On the other hand, constraints (also called *integrity constraints*) are mainly used in deductive databases and logic programming in order to check if the knowledge base is consistent with a number of specified conditions. As a result, constraints cannot draw inference of new information, but they may lead to inconsistencies in the case that some conditions specified by them are violated inside the knowledge base. Usually, constraints are rules without head (i.e., they have a “false” value in their head), denoting that the conditions stated in the body should not be satisfied concurrently by the knowledge base.

Similarly to restrictions, constraints may be divided to:

- *Value constraints* or *range constraints*. Their importance consists in checking the type of a property value.
- *Cardinality constraints.* They are used for checking the number of values that an individual may have for a specific property.

Similarly to (6), a cardinality constraint could be written in logic programming as:

$$\leftarrow \text{Profile}(s) \wedge \neg\text{hasInput}(s,i) \qquad (7)$$

More details about integrity constraints and restrictions may be found in (de Bruijn, Polleres, Lara, & Fensel, 2005), where an alternative ontology language based on the logic programming subset of OWL is presented.

Other differences

A number of additional differences between the described approaches have also been discussed in the literature. (Eiter et al., 2006a) raises the point of non-ground entailment in the logic programming approach, declaring that, since the semantics of logic programming is defined in terms of sets of ground facts, the inference of non-factual knowledge is not allowed. Contrary to the logic programming, classical logic permits the entailment of non-factual knowledge. Furthermore, decidability issues are explored in the combination of the two perspectives. In addition, (Pattel-Schneider, 2006) describes the treatment of datatypes by classical logic approaches (e.g., DLs) and logic programming. Moreover, it examines the role of tools that support each methodology and how they can facilitate the modeling of knowledge.

The aforementioned modeling paradigms have been proposed for formalizing rules on the Web. Concerning the advantages and the disadvantages of each approach, we believe that both perspectives are useful in the context of Semantic Web.

The open nature of the classical logic seems to be more suitable for formalizing such a distributed environment. By this mean, Semantic Web technologies based on classical logic accomplish the integration of information stemming from different sources. Knowledge becomes shareable in the sense that it can be accessed without any obligation to adopt a specific model or schema. Hence, data are available to everyone through WWW so as to be retrieved and used in applications appropriately.

On the other hand, logic programming makes several assumptions in order to simplify tasks like reasoning or modeling of the knowledge base. There are several useful features of logic programming like high expressiveness, negation-as-failure and decidability of reasoning. Moreover, many of the reasoning engines targeting at managing knowledge expressed in classical logic formalisms have been recently extended to support such features (e.g., Pellet, RacerPro, Bossam). Additionally, both users and developers are very familiar with the world of databases and logic programming languages like Prolog. It is worthy to mention that even SPARQL engines return “no” in ASK query patterns with no solutions. However, “don’t know” would fit better in an open environment where multiple solutions may exist outside the current knowledge base. This way, SPARQL takes advantage of the users’ familiarity with common databases, deciding not to complicate the query answering process.

To summarize, last decade has shown to the developers of Web applications that different types of Semantic Web enabled applications requires different styles of modeling. As a result, the development of a framework that satisfy most of these needs seems to become unavoidable.

4.3 Quantitative Evaluation of Reasoning Modules

This section intends to quantify the total time required by popular reasoning modules in order to perform common reasoning tasks. Instead of (Pan, 2005; Gardiner, Horrocks, & Tsarkov, 2006), where the evaluation of description logic reasoners focuses on TBox reasoning tasks, here we examine both TBox and ABox reasoning as well as rules application. In particular, Figures 3 to 6 show the results of some indicative performance tests. These tests aim at demonstrating the efficiency of description logic reasoners and mainstream rule engines.

This quantitative evaluation involved the last stable versions of RacerPro (v1.9.0), Pellet (v1.5.1) and FACT++ (v1.1.11) description logic reasoners that were available for ontology reasoning as well as the Jess rule engine for SWRL rules application. Bossam was also tested as both ontology reasoner and rule engine in this evaluation. However, Bossam's results are not displayed in the figures, since such a comparison would be improper. This stems from the fact that although Bossam constitutes a sound reasoning module it is not a complete reasoner for OWL. However, we should mention that Bossam performed well (i.e., it accomplished smaller execution times than the other OWL reasoners) handling small-size ontologies, while it threw out-of-memory errors when provided with large-scale ontologies (i.e., containing 500 and 1000 instances). Moreover, all reasoners were managed through Protégé OWL API in Java except Bossam which provides its own API. Finally, Jess was accessed via SWRL-Jess bridge (O'Connor et al., 2005).

The knowledge base used in these tests stems from the Web service description paradigm and it consists of a domain ontology and OWL-S service descriptions. The domain ontology, which represents the domain of football (i.e., player/team details and statistics), includes concepts described through necessary and sufficient conditions. The elements of this ontology are used for the annotation of some Semantic Web services that provide search functionality to some underlying football statistics databases. Specifically, the service inputs and outputs are annotated through concepts of this domain ontology. The extract of the domain ontology used contains 37 concepts, 8 object properties and 3 datatype properties. Each service description involves 19 concept instances and 11 property instances.

Figure 3 depicts the performance of three description logic reasoners while checking the consistency of all classes over the aforementioned knowledge base. The execution times presented in this figure do not involve the time needed to load the models. The figure shows that these times increase as the number of instances contained in the knowledge base also increases. FACT++ and Pellet attained the best performance by achieving similar execution times. On the other hand, RacerPro performed worse than the others.

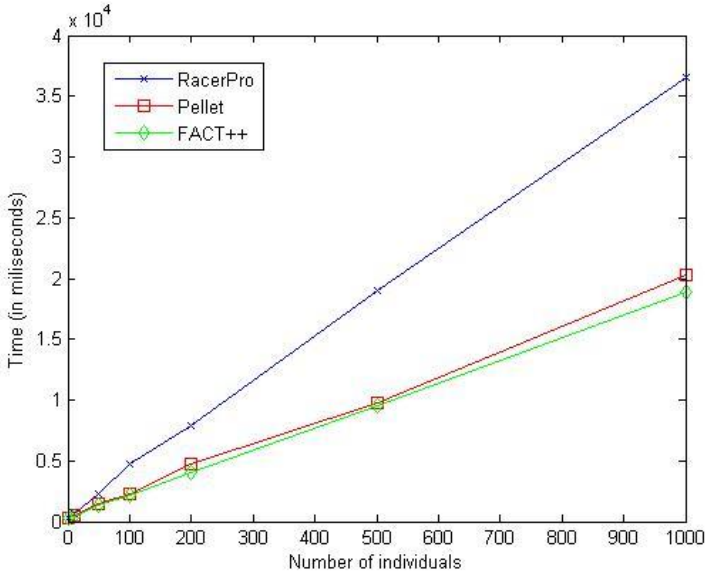

Figure 3: Evaluation of consistency check (TBox)

In Figure 4, the times required for classifying the hierarchy of the ontology are presented. Again, the times shown in the figure do not include the loading time of each model. The resulting performances are very similar to those of the consistency checking process. FACT++ has slightly better performance than Pellet, while RacerPro has the worst performance again.

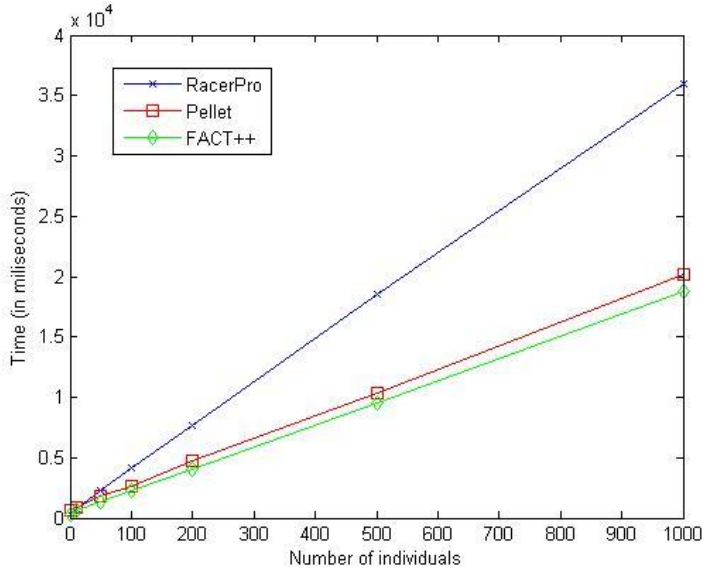

Figure 4: Evaluation of classification (TBox)

Figure 5 shows the efficiency of RacerPro and Pellet reasoners in realizing reasoning over ABoxes. In this figure we do not display FACT++, since it provides incomplete ABox reasoning. In particular, the experiment concerns the identification of the types (i.e., classes) that each individual belongs to. The figure shows that even the reasoning over an ontology containing 100 instances using Pellet (which seems to be the most efficient reasoner in this reasoning task) requires about 20 seconds. Moreover, both reasoners failed to compute inferred individual types for ontologies containing 500 and 1000 individuals.

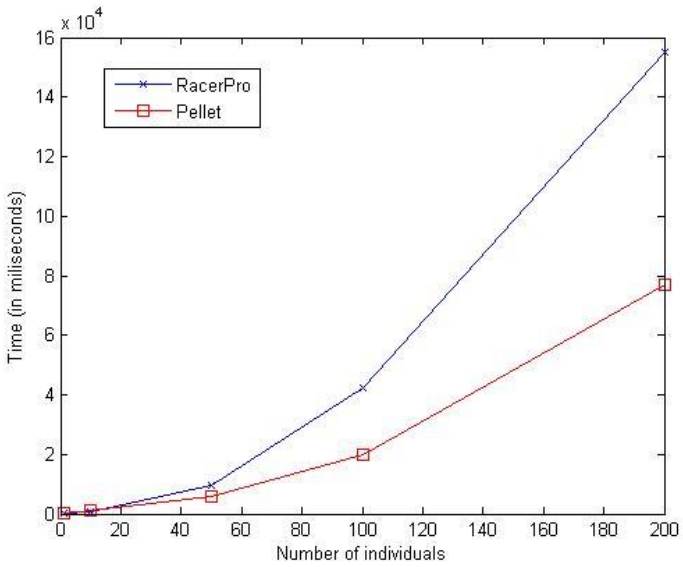

Figure 5: ABox reasoning evaluation (computation of inferred individual types)

It is worthy to mention that version 1.9.2 Beta of RacerPro was also available during the experiments. This version seemed to remain slower than FACT++ and Pellet in TBox reasoning tasks (consistency check and classification of hierarchy) while it performed better in ABox reasoning. However, the results are not displayed in the figures above, since this is not a stable version of the reasoner.

Figure 6 shows the time required by Jess rule engine to apply SWRL rules over ontologies containing various numbers of instances. These rules represent service composition constraints that should apply to the inputs and outputs of the services. We applied two different sets of rules to the previously described knowledge bases: the first set consisted of 10 rules while the second of 50 rules. In both cases, the execution time increases proportionally to the number of instances contained in the ontology. As expected, the delay imposed by the execution of rules was affected by the number of rules. However, the number of instances seems to affect execution times more than the number of rules. Hence, the inference process is complicated when real-time demands come up in distributed environments where applications have to support multiple individual instances.

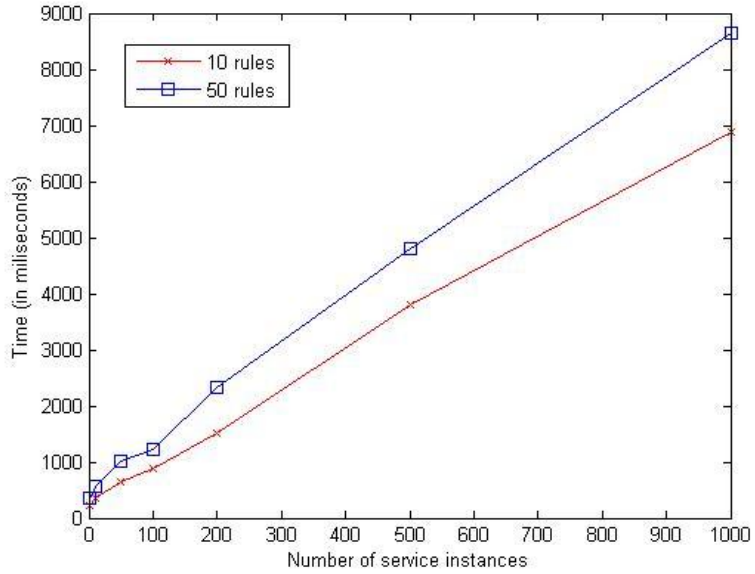

Figure 6: Jess performance executing SWRL rules

To summarize, the time required to handle a large KB and execute reasoning in terms of Semantic Web technologies like DLs constitutes a limiting factor, even for the most efficient reasoners. This is a fundamental problem on the Web, since applications have to deal with a large number of instances representing data. Especially, when considering real time applications that call for small response times, the aforementioned results become unmanageable. This is the main reason why many researchers have recently focused on developing efficient and scalable reasoning applications over large individual sets. Instance Store (Horrocks, Li, Turi, & Bechhofer, 2004a) is such an approach that permits working in conjunction with any ontology reasoner that implements the DIG interface.

5 FUTURE TRENDS AND OPEN ISSUES

Several open issues that deserve further research efforts in the future have been identified throughout the chapter. Firstly, a Web standard for editing and embedding rules into Web applications should be specified. In our opinion, such a language should focus on its simplicity, instead of providing major expressiveness capabilities that will lead to a clumsy formalism. Rule Interchange Format (RIF) Working Group (RIF, 2008) works in this direction in order to produce W3C recommendations for enabling interchange of rules. Specifically, RIF does not intend to provide explicit mappings between various rule languages, but specify a mechanism capable of defining the meaning of the formulas of a rule language. This way, rules could be automatically translated across different formalisms.

Additionally, more tractable algorithms for Web-rules, TBox and ABox inference problems should be developed. These algorithms should be able to handle complex and heavy knowledge bases that combine rules and ontologies with a large number of concepts and relationships. Such knowledge bases are substantial, especially in the context of information integration where multiple domain vocabularies interact with upper-level ontologies. Furthermore, the Web is a large scale environment, where a huge set of resources are added every day. As a result, future knowledge bases will

have to capture and describe more and more individuals. These facts should be taken into account by the developers that will target at designing new reasoning algorithms. Finally, a more practical issue that should be investigated is the development of a unified reasoning framework capable of managing both ontologies and rules. Today, there is no efficient and easy-to-use integrated reasoning module that can reason over both formalisms. As a consequence, the developer should use at least two different reasoning modules to handle such an integrated knowledge base. This can result to unexpected situations. For example, the restrictions defined by the ontology can be violated by the application of rules, since the rule engines do not take into account these restrictions (e.g., disjointness of concepts). Similarly, the application of rules could produce knowledge that would be useful for further description logic inferences.

6 CONCLUSIONS

In this chapter we have discussed the application of rules to Web applications in order to achieve intelligent application behavior and efficient management of knowledge. We have described the main methodologies and technologies for integrating rules with ontologies, since the latter constitutes a mature knowledge technology on the Web. The chapter has also examined aspects of the different approaches. We also showed through several experiments that current reasoning modules are not efficient enough to manage knowledge stemming from large-scale environments like the Web, especially in the context of real-time applications which impose severe constraints in response times. Finally, more improvements should be made in the standardization of rules formalism on the Web and in the development of reasoning modules that can handle and reason over both ontologies and rules as integral knowledge.

7 REFERENCES

Alferes, J., Damasio, C., & Pereira, L. (2003). Semantic Web Logic Programming Tools. *LNCS 2901*, (pp. 16-32). Springer- Verlag.

Antoniou, G., Billington, D., Governatori G., & Maher, M., J. (2001). Representation Results for Defeasible Logic. ACM Transactions on Computational Logic 2(2), 255–287.

Antoniou, G., Damasio, C., V., Grosof, B., Horrocks, I., Kifer, M., Maluszynski, J., & Patel Schneider, P., F. (2005). Combining Rules and Ontologies : A Survey. Deliverables I3-D3, REWERSE, http://rewerse.net/deliverables/m12/i3-d3.pdf.

Antoniou, G., & van Harmelen, F. (2004). A Semantic Web Primer. MIT Press.

Baader, F., Calvanese, D., McGuiness, D., Nardi, D., & Patel-Schneider, P. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge: Cambridge University Press.

Bassiliades, N., Antoniou, G., Vlahavas, I. (2006). A defeasible logic reasoner for the semantic web, *International Journal on Semantic Web and Information Systems 2 (1), 1–4.*

Bechhofer, S., Moller, R., & Crowther, P. (2003). The DIG Description Logic Interface. In D. Calvanese, G. de Giacomo, and F. Franconi, editors, Proc. of the 2003 Int. Workshop on Description Logics (DL 2003), volume 81 of CEUR Workshop Proceedings, Rome, Italy, September 5–7 2003.

Boley, H., Kifer, M., Patranjan, P., L., & Polleres, A. (2007). Rule Interchange on the Web. In Reasoning Web 2007, number 4636, (pp. 269-309). Springer, September 2007.

Brickley, D., & Guha, R.V. (2004). RDF vocabulary description language 1.0: RDF schema. Recommendation 10 February 2004, W3C. Retrieved April 22, 2008, from http://www.w3.org/TR/rdf-schema/.

de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., & Fensel, D. (2005). The Web Service Modeling Language WSML. Technical report, WSML. WSML Final Draft D16.1v0.21. http://www.wsmo.org/TR/d16/d16.1/v0.21/.

de Bruijn, J., Polleres, A., Lara, R., & Fensel, D. (2005). OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning on the Semantic Web. In Proc. WWW2005, (pp. 623–632), Chiba, Japan, May 10–14 2005.

Cardoso, J. (2007). The semantic web vision: Where are we? Intelligent Systems, 22(5):84–88.

CLIPS: A tool for building expert systems. (2008). Retrieved April 22, 2008, from http://clipsrules.sourceforge.net/.

Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., L., Patel-Schneider, P., F., & Stein, L., A. (2004). OWL Web Ontology Language Reference. W3C Recommendation 10 February 2004. Retrieved April 22, 2008, from http://www.w3.org/TR/owl-ref/.

Donini, M. F., Lenzerini, M., Nardi, D., & Schaerf, A. (1998). AL-log: Integrating Datalog and Description Logics. *Journal of Intelligent Information Systems, 10(3),* 1998, pp. 227-252.

Drools (2008). Retrieved April 22, 2008, from http://www.jboss.org/drools/.

Eiter, T., Ianni, G., B., Polleres, A., Schindlauer, R., & Tompits, H. (2006a). Reasoning with rules and ontologies. In Pedro Barahona, FraŕEcois Bry, Enrico Franconi, Ulrike Sattler, and Nicola Henze, editors, Reasoning Web, Second International Summer School 2005, Tutorial Lectures, Lecture Notes in Computer Science. (pp. 93–127). Springer.

Eiter, T., Ianni, G., B., Schindlauer, R., & Tompits, H. (2006b). dlvhex: A System for Integrating Multiple Semantics in an Answer-Set Programming Framework. In M. Fink, H. Tompits, and S. Woltran, editors, Proceedings 20th Workshop on Logic Programming and Constraint Systems (WLP 06), (pp. 206-210).

Eiter, T., Ianni, G., B., Schindlauer, R., & Tompits, H. (2005). A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In Proc. IJCAI 2005, Morgan Kaufmann.

Eiter, T., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2004). Combining answer set programming with description logics for the semantic web. In Proc. of the International Conference of Knowledge Representation and Reasoning (KR04).

Farrell, J., & Lausen, H. (2007). Semantic Annotations for WSDL and XML Schema. W3C Recommendation, W3C, August 2007. Retrieved April 22, 2008, from http://www.w3.org/TR/sawsdl/.

Forgy, C. (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence, 19, 17-37.

Gardiner, T., Horrocks, I., & Tsarkov, D. (2006). Automated benchmarking of description logic reasoners. In Proc. of DL.

Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3_4), 365-386.

Grosof, B. N., Horrocks, I., Volz, R., & Decker, S. (2003). Description logic programs: Combining logic programs with description logic. In Proc. of the Twelfth International World Wide Web Conference (WWW 2003), (pp. 48-57). ACM.

Haarslev, V., & Moller, R. (2001). RACER system description. In R. Gor´e, A. Leitsch, and T. Nipkow, editors, International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy, (pp. 701–705). Springer-Verlag.

Horrocks, I. (1998), Using an expressive Description Logic: FaCT or fiction? in: Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2–5, 1998, (pp. 636–647).

Horrocks, I., Li, L., Turi, D. & Bechhofer, S. (2004a). The instance store: DL reasoning with large numbers of individuals. In Proc. of the 2004 Description Logic Workshop (DL~2004), (pp. 31-40).

Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004b). SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, 21 May 2004. Retrieved April 22, 2008, from http://www.w3.org/Submission/SWRL/.

Horrocks, I., Parsia, B., Patel-Schneider, P., F., & Hendler, J. (2005). Semantic web architecture: Stack or two towers? In Proc. PPSWR 2005, (pp. 37–41), Dagstuhl Castle, Germany, September 11–16 2005.

Hustadt, U., Motik, B., & Sattler, U. (2004). Reducing SHIQ$^-$ Descrption Logic to Disjunctive Datalog Programs. Proc. of the 9th International Conference on Knowledge Representation and Reasoning (KR2004), June 2004, (pp. 152-162).

ILOG, Business Rule Management Systems, Optimization Tools and Engines, Visualization Software Components, Supply Chain Applications (2008). Retrieved April 22, 2008, from http://www.ilog.com/.

Jang, M., & Sohn, J-C. (2004). Bossam: An Extended Rule Engine for OWL Inferencing. Hiroshima, Japan: In Workshop on Rules and Rule Markup Languages for the Semantic Web at the 3rd International Semantic Web Conference (LNCS 3323), (pp. 128-138). Hirosima, Japan: Springer-Verlag.

Jess, The Rule Engine For the Java Platform (2008). Retrieved April 22, 2008, from http://www.jessrules.com/jess/index.shtml

KAON – The KArlsruhe ONtology and Semantic Web tool suite (2008). Retrieved April 22, 2008, from http://kaon.semanticweb.org.

KAON2 – Ontology Management for the Semantic Web (2008). Retrieved April 22, 2008, from http://kaon2.semanticweb.org.

Kifer, M., de Bruijn, J., Boley, H., & Fensel, D. (2005). A Realistic Architecture for the Semantic Web. In Proc. RuleML 2005, (pp. 17–29), Galway, Ireland.

Klyne, G., & Carroll, J.J. (2004). Resource description framework (RDF): Concepts and abstract syntax. Recommendation 10 February 2004, W3C. Retrieved April 22, 2008, from http://www.w3.org/TR/rdf-concepts/.

McBride, B. (2002). Jena: A Semantic Web Toolkit, *IEEE Internet Computing, 6(6)*:55-59.

Mei, J, Bontas, E.P., & Lin, Z. (2005). OWL2Jess: A Transformational Implementation of the OWL Semantics. In *Proceedings of International Workshops on ISPA*, LNCS 3759, pages 599–608, 2005.

Motik, B., Sattler, U., & Studer, R. (2005). Query answering for owl-dl with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 3, 41–60.

Motik, B., Horrocks, I., Rosati, R., & Sattler, U. (2006). Can OWL and logic programming live together happily ever after? In Proceedings ISWC-2006. LNCS, vol. 4273, (pp. 501–514). Springer.

Nute, D. (1994). Defeasible logic, in: Handbook of Logic in Artificial Intelligence and Logic Programming, Nonmonotonic Reasoning and Uncertain Reasoning, vol. 3, Oxford University Press.

O'Connor, M., Knublauch, H., Samson, T., & Musen, M. (2005). Writing Rules for the Semantic Web Using SWRL and Jess. Protégé With Rules WS, Madrid.

Pan, J. Z., & Horrocks, I. (2004). OWL-E: Extending owl with expressive datatype expressions. Technical report, IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester.

Pan, J. (2005). Benchmarking DL reasoners using realistic ontologies. In Proc. of the OWL: Experiences and Directions Workshop.

Parsia, B., Sirin, E., & Kalyanpur, A. (2005) Debugging OWL Ontologies, in: Proceedings of the 14th International World Wide Web Conference (WWW2005), Chiba, Japan, May 2005, URL: http://www.mindswap.org/papers/debuggingOWL.pdf.

Patel-Schneider, P. F., & Horrocks, I. (2006). Position paper: a comparison of two modelling paradigms in the Semantic Web. In Proceedings of the 15th International Conference on

World Wide Web (Edinburgh, Scotland, May 23 - 26, 2006). WWW '06, New York, NY, (pp. 3-12). ACM Press.
Polleres, A., & Schindlauer, R. (2007). DLVHEX-SPARQL: A SPARQL Compliant Query Engine Based on DLVHEX. Proceedings of 2nd International Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services (ALPSWS2007), Porto, Portugal (13th September 2007).
Prud'hommeaux, E., & Seaborne, A. (2005). SPARQL Query Language for RDF. http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/, 2005.
RacerPro (2008). Retrieved April 22, 2008, from http://www.racer-systems.com/products/racerpro/index.phtml.
Rainer. A. (2005). Web Service Composition under Answer Set Programming. KI-Workshop "Planen, Scheduling und Konfigurieren, Entwerfenl" (PuK).
RIF - Rule Interchange Format Working Group (2008). Retrieved April 22, 2008, from http://www.w3.org/2005/rules/wiki/RIF_Working_Group
Rosati, R. (2006a). DL+log: Tight Integration of Description Logics and Disjunctive Datalog. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), Lake District, UK (pp. 68-78). AAAI Press.
Rosati, R. (2006b). Integrating Ontologies and Rules: Semantic and Computational Issues. In Pedro Barahona, François Bry, Enrico Franconi, Nicola Henze, and Ulrike Sattler, editors, Reasoning Web, Second International Summer School 2006, Lissabon, Portugal, September 4_8, 2006, Tutorial Lectures, volume 4126 of Lecture Notes in Computer Science (LNCS), (pp. 128-151). Springer.
Rosati, R. (2006c). DL+log: Tight Integration of Description Logics and Disjunctive Datalog. In Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), pages 68–78. AAAI Press, 2006.
RuleML, Rule Markup Initiative (2008). Retrieved April 22, 2008, from http://www.ruleml.org/.
Sirin, E., Parsia, B., Grau, B., C., Kalyanpur A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner, *Journal of Web Semantics*, 5(2).
Tsarkov, D., & Horrocks, I. (2006). FaCT++ Description Logic Reasoner: System Description. In Proc. of the International Joint Conference on Automated Reasoning (IJCAR 2006).
W3C Semantic Web Activity (2008). Retrieved April 22, 2008, from http://www.w3.org/2001/sw/.
Wagner, G. (2003). Web Rules Need Two Kinds of Negation. LNCS, Vol. 2901., Berlin Heidelberg New York (pp. 33-50). Springer- Verlag.

Endnotes

[1] also called as Datalog view in the literature
[2] http://con.fusion.at/dlvhex/
[3] it offers free trials while educational and research licenses are available
[4] Universal Resource Identifiers