A Web Service Reasoner for the Semantic Web

Efstathios Simeonidis[1], Georgios Meditskos[2], Efstratios Kontopoulos[2], Nick Bassiliades[2]

[1]*University of Macedonia, Thessaloniki, Greece, mis0518@uom.gr*

[2]*Department of Computer Science, Aristotle University of Thessaloniki, Greece, {gmeditsk, nbassili, skontopo}@csd.auth.gr*

Web Services offer a standard interface for describing the available services on the Web. Common applications of Web Services are B2B communications and e-commerce, mainly because they are platform and network-independent, easily deployable and offer great reusability. Moreover, the Semantic Web initiative proposes technologies and languages for annotating information on the Web, so that it can be understood, interpreted and exploited by software agents. For the realization of such architecture, agents should be able to reason over the annotated information, in order to make decisions and to successfully cooperate with each other. To this end, logic and rules play an important role, enabling the description of assertions that can be used to derive new knowledge and the implementation of agent behaviour. In this paper we describe the Web Service implementation of a rule-based RDF reasoner, called DR-DEVICE. The deployment of the reasoner as a Web Service enables other applications to use the system over the Internet, by exploiting the well-defined interface that Web Service technology offers. Agents can use the service by interchanging messages, based on standards (SOAP) and already existing Internet protocols (HTTP), in order to enrich their functionality with reasoning capabilities. Furthermore, the system can serve as a software component of a more complex and distributed framework that would compose a variety of Web Services in order to achieve a new functionality. DR-DEVICE supports both deductive and defeasible rules and can be extended to the proof layer of the Semantic Web architecture, for validating the derivations stemming from the inferential logic activity. The service is available to end users through a Web interface.

Keywords

RDF, Rule-based Reasoning, Semantic Web, Web Services.

1. Introduction

The advent of *Web Services* is a proof that nowadays the need for communication among loosely coupled distributed systems is bigger than ever. Web Services offer a well-defined interface, through which other programs may interact, by sending messages based on Internet protocols and Web standards. They may also be combined, in order to achieve a complex service, whose functionality cannot be achieved by a single one, a procedure that is called *service composition*. The description of a service interface is based on the Web Service Description Language (WSDL [14]) that describes the syntax of the input and output messages using XML, as well as other details needed for the invocation of the service. The communication is based on the Simple Object Access Protocol (SOAP [12]), an XML-based framework that provides a message construct that can be exchanged over a variety of underlying protocols.

The vision of the *Semantic Web* is to provide the necessary standards and infrastructure for transforming the Web into a more automatic environment, where software agents would have the ability to search for requested information automatically. This is feasible by describing appropriately the already available data on the Web in a machine-understandable way. Ontologies can be considered as a key towards this goal since they provide a controlled vocabulary of concepts, each with an explicitly defined and machine processable semantics.

The development of the Semantic Web proceeds in layers, where each layer is built on top of the others [5]. Currently, the *ontology* layer has reached a sufficient level of maturity, having OWL [13] as the basic ontology definition form, built on top of RDF [10]. The next step is to move to the higher levels of *logic* and *proof*, which are built on top of the *ontology* layer, where rules are considered the key, since (a) they can serve as extensions of, or alternatives to, description logic based ontology languages and (b) they can be used to develop declarative systems on top of (using) ontologies.

A lot of effort is undertaken to define a rule language for the Semantic Web on top of ontologies, in order to combine already existing information and deduce new knowledge. Currently, RuleML [6] is the main standardization effort for rules on the Web to specify queries and inference in Web ontologies, mappings between ontologies and dynamic Web behaviors of workflows, services and agents. Furthermore, the Rule Interchange Format Working Group [11], which very recently has been formed in W3C, aims at producing an extensible core rule language, which will allow rules to be translated between rule languages and thus transferred between rule systems.

One approach to implement a rule system on top of the Semantic Web ontology layer is to start from scratch and build inference engines that draw conclusions directly on the OWL or RDF data model. However, such an approach tends to throw away decades of research and development on efficient and robust rule engines.

In this paper we present the Web Service implementation of the *DR-DEVICE* system [3], an RDF rule reasoner built on top of the CLIPS production rule system. More specifically, DR-DEVICE is a reasoning system that processes RDF data, performs defeasible inference, produces the results and exports them as RDF data. By deploying

the DR-DEVICE system as a Web Service, we enable intelligent agents to directly use the system and draw conclusions about semantically described information on the Web.

The rest of the paper is organized as follows: in section 2 we give a short introduction to Web Services and the Semantic Web, underlying the basic features of these Web technologies. In section 3 we describe an imaginary use case scenario, involving agents that use the DR-DEVICE Web Service, mentioned above, while the next section presents the implementation of the DR-DEVICE system as a Web Service, giving also screenshots of a client Web application we have developed that consumes the Web Service. Finally, section 5 concludes the paper, also posing directions for future work.

2. Web Services and the Semantic Web

2.1 Web Services

The promise of Web Services and all service-oriented architectures is mainly to create a distributed environment, in which all applications and their components can collaborate in a transparent fashion regardless of their platform [8]. A Web Service is a piece of business logic located somewhere in the Internet and made available through its basic protocols, such as HTTP or SMTP. Using one Web Service can be as easy as logging into a web site or as complicated as a negotiation between organizations.

Web Services and their technologies are based on XML [1]. That is because XML as a language allows a platform-independent exchange of data at a basic level. XML and its great adaptation, combined with the rest of the Web Service technologies, allow for global co-operation between organizations.

Web Services use XML for data representation at all stages so that their components can co-operate seamlessly [7]. XML can eliminate any connection an application may have with a particular network type or operating system.

Anyone that consumes a Web Service is not tightly connected to it – the service may change during time and the client will still be able to interoperate with the Web Service. This is called *loose coupling*. In the exact opposite case, the slightest change in the service side would require change in the client side and vice versa.

Web Services also support Remote Procedure Calls (RPC). Clients consuming Web Services can call procedures and functions of remote objects, using an XML-based protocol, SOAP. Web Services allow clients to transparently exchange complicated documents, using XML, in order to facilitate cross-organizational integration.

The Web Services architecture states that every Web Service needs to be found, described, accessed and return the result to its client. UDDI (Universal Description, Discovery and Integration) is one of the core Web Services standards and acts as a registry of Web Services helping in their discovery by clients. Each UDDI registry provides clients with a WSDL (Web Services Description Language) document that

describes how a client can communicate with a Web Service and what functions are available. Once a client finds a suitable Web Service, it can revoke a function from a Web Service and get its result as a message, using the SOAP (Simple Object Access Protocol) protocol. These messages/calls are nothing but XML messages transported over HTTP or SMTP providing access to the Web Service's functions.

2.2 Semantic Web

Today's Web is suitable for human consumption and is organized around content presentation and not information meaning. Getting in touch with other people, seeking information about hotels and tickets, searching for lower prices and ordering products on-line are the majority of activities that take place on the WWW. However, these activities utilize only a small amount of information available on the World Wide Web.

Search engines do a relatively good job, but only in simple queries. Complex queries may yield results not related to the ones the user expected to see or, even worse, the relevant pages may be amongst thousands of others mildly relevant or totally irrelevant to the query. However, even when the search is successful, the information needed may be spanned across various documents and the user must manually put the information pieces together. That would often require more queries using synonyms of the original keyword or using other sources to find more possible relevant pages.

The Semantic Web vision emerged after Information Retrieval (IR) received great attention in the late 90s and the term *metadata* was coined. Metadata is often described as "data about data" and is used to facilitate the understanding, use and management of other data. The aim of the Semantic Web is to allow the creation of advanced knowledge management systems that will organize the knowledge and support easy maintenance and access to it. Tasks like these require the metadata to be machine-readable and machine-interchangeable [9].

The Semantic Web relies on already existing technologies to facilitate efficient information retrieval, by organizing knowledge in conceptual hierarchies, called *ontologies*. In other words, the Semantic Web provides the tools to model knowledge on the Web, just as UML provides the tools to model business workflow.

But Semantic Web and Web Services both depend crucially on moving beyond syntax and upgrading the role of machines (computers). Additionally, the Semantic Web needs already existing and established technologies that facilitate platform-independent machine-to-machine interoperation. The Web Services framework offers these technologies and fitting Web Services into the Semantic Web will provide the connections and functions needed to coordinate such an experiment.

Additionally, machines should be able to reason over the represented data, drawing conclusions that seem obvious to humans but not to machines. Reasoning and logic have been under extensive study in A.I. and their ultimate goal is to make implicit knowledge explicit. Using automated reasoning in the Semantic Web can help uncover implicit knowledge hidden into the ontologies.

Logic and rules, in particular, can provide explanations for the conclusions drawn. All logic systems have a "retrace" function, where anyone can inspect the steps taken to draw a final conclusion. This procedure is often described as "proof" and is used to increase user confidence towards a logic system.

However, in real life, Semantic Web agents will do all the "hard work". Agents are described as pieces of software that receive a task and some preferences from the user and give answers to him about the initial task (or even accomplish it after the user's confirmation). Agents will not replace real users in the Semantic Web but in most cases will work interactively with them. Semantic Web agents will make extensive use of all the Semantic Web technologies [2]:

- Metadata, in order to identify and extract information from Web sources.
- Ontologies, to increase the accuracy of Web searches, to allow them to interpret the retrieved information and to be able to communicate with other agents.
- Logic, in order to process the retrieved information and to draw conclusions about the preferable courses of action.

3. Use Case

The vision of the Semantic Web, concerning the future interaction with the Web through personal intelligent agents, can be better illustrated through an example. Assume that a woman, let's call her Alice, is looking for an apartment to rent. More specifically, she is looking for an apartment of at least 45 m^2 with at least 2 bedrooms. If it is on the 3rd floor or higher, the house must have an elevator. Also, pet animals must be allowed. Alice is willing to pay \$300 for a centrally located 45 m^2 apartment and \$250 for a similar flat in the suburbs. In addition, she is willing to pay an extra \$5 per m^2 for a larger apartment, and \$2 per m^2 for a garden. She is, however, unable to pay more than \$400 in total. If given the choice, she would go for the cheapest option. Her 2nd priority is the presence of a garden, while additional apartment space (m^2) is her lowest priority.

Alice assigns her personal Semantic Web agent to search for an apartment that fulfils her demands. While searching the Web, the agent comes across another agent that represents a broker and asks for an apartment, which will meet Alice's needs. The latter agent accepts Alice's requirements and scans through the available apartments in the database of its brokering agency, looking for apartments that meet the demands. Eventually, the broker's agent comes up with three suitable apartments, also passing to Alice's agent the corresponding details (size, location, etc).

Alice's agent, however, is programmed to demand explanations of other agents' allegations, which it subsequently requests from the broker's agent. The latter responds to this new request with the URI of the brokering agency's Web Service reasoner it used, in order to extract the suitable results. Alice's agent invokes the Web Service and compares the newly produced results with the previous ones. It infers that the two result

sets are consistent and decides to trust the specific brokering agency. Ultimately, Alice's agent returns to its user the apartments that suit her needs, accompanied by the corresponding brokering agency's contact details, plus a confirmation that the specific results are trustworthy.

The use case described above may seem like science fiction, but its implementation is indeed within the capabilities of today's technology! Its realization is not a matter of new scientific discoveries, but rather a technological issue that requires adoption by the users of the Semantic Web technologies, described in the paper.

4. DR-DEVICE

4.1 System Description

DR-DEVICE is a system capable of reasoning about RDF metadata over multiple Web sources, using defeasible logic rules. Defeasible reasoning is a rule-based approach for efficient reasoning with incomplete and inconsistent information. Such reasoning is, among others, useful for ontology integration, where conflicting information arises naturally; and for the modelling of business rules and policies, where rules with exceptions are often used. The system is implemented on top of the CLIPS production rule system and builds upon R-DEVICE, a deductive rule system over RDF metadata that also supports derived attribute and aggregate attribute rules. Rules can be expressed either in a native CLIPS-like language, or in an extension of the OO-RuleML syntax. The operational semantics of defeasible logic are implemented through compilation into the generic rule language of R-DEVICE [3].

The most important features of DR-DEVICE are the following:

- Support for multiple rule types of defeasible logic, such as strict rules, defeasible rules, and defeaters.
- Support for both classical (strong) negation and negation-as-failure.
- Support for conflicting literals, i.e. derived objects that exclude each other.
- Direct import from the Web of RDF ontologies and data as input facts to the defeasible logic program.
- Direct import from the Web of defeasible logic programs in an XML compliant rule syntax (RuleML).
- Direct export to the Web of the results (conclusions) of the logic program as an RDF document.

- Currently, we are working on providing direct export to the Web of explanations about the results in the form of a proof tree for each derived object in an extension of the RuleML syntax [4].

4.2 DR-DEVICE as a Web Service Reasoner

Turning DR-DEVICE into a Web Service is an excellent case of enabling a legacy application to function over the Web and thus help towards the Semantic Web aim. There are several steps required to create and deploy a Web Service. First of all, we need to select the appropriate platform that will host the service. Our Web Service will use the Java framework for Web Services, hosted in an Apache Tomcat application server, using the Apache SOAP libraries.

The Java, Apache Tomcat and Apache SOAP solution is an excellent open-source tool that assists in creating and implementing Web Services. Few lines of code are required to create a service and few clicks are required to deploy it. The programmer has a low-level handle over what is going on (message exchanged, procedure calls, etc.) and all the Java packages available to assist him.

Our next job was to install and deploy the components needed. The system architecture is depicted in Figure 1. The concept was to create a Web Service that will (a) accept SOAP messages through the messaging subsystem of Apache SOAP, (b) create a DR-DEVICE project which results would be published on the Apache Tomcat server, (c) create and return a SOAP response revealing the location of the RDF result file so the client/agent can retrieve it.

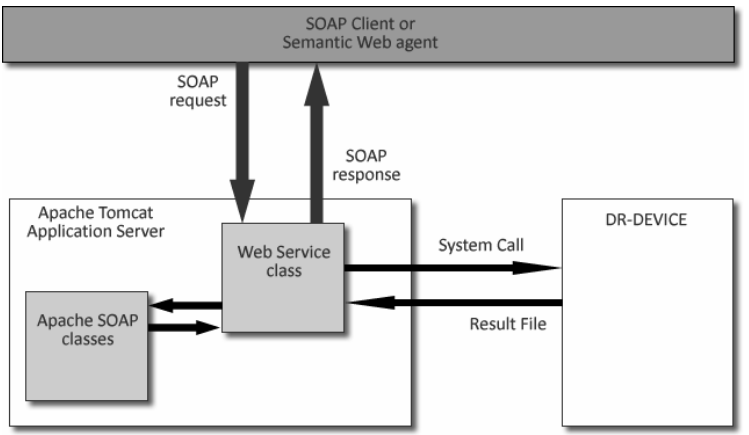

Figure 1 The system architecture.

The SOAP request is a properly designed XML document that is sent to the Web Service, in order to initiate the process. Since this request is a like a function call, we must include in its contents all the parameters needed for this process to run. At this point, we must define the format of this message:

```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessRules xmlns="webrdevice">
      <ruleFile>[URL of rule file(ruleML)]</ruleFile>
      <xsltFile>[DR-DEVICE XSL file (optional)]</xsltFile>
      <projectName>[project name (optional)]</projectName>
      <timeToLive>[minutes to keep the results]</timeToLive>
</ProcessRules>
```

Figure 2 The SOAP request format.

This request format also defines how the service must be deployed. The root element of this XML file is called "`ProcessRules`", which means that this will be the name of our Web Service class. Additionally, the namespace "`webrdevice`" of this document is the name of the Web Service as it must be deployed on our server. The first parameter is the location of the rule file that will act as input to the DR-DEVICE system. DR-DEVICE will transform the RuleML into its native format, using the next parameter's XSL file. If it is not specified, it will use the one already located on the server. The third parameter is the desired project name that will also act as the output folder. If it is left blank, the Web Service will generate a project name based on the current date/time. To facilitate clean-up, the results will be purged after "`timeToLive`" minutes have passed.

Next, we have to describe the Web Service class and as we will not focus on code, we will outline the workflow of the Java class acting as the Web Service as follows:

- Import the libraries needed (SOAP, XML, Java.net)
- Include some basic functions such as copying a file, replacing a string in a file, etc
- Handle the received message and pass it to objects
- Format the project name in case it is not explicitly defined
- Use DOM elements to parse the incoming SOAP-XML file
- Create the project folder into DR-DEVICE's work folder and create the required batch files for the project to run
- Run the project and wait for its completion
- Publish the result file on the server
- Return a SOAP response revealing the location (URL) of the result file.

In order for the Web Service to function properly, we must deploy it on a Tomcat server. The Apache SOAP implementation provides us with a web interface that deploys Web Services onto the server (Figure 3). But first we must compile our class and copy the .class file to the `[Tomcat folder]\webapps\soap\WEB-`

`INF\classes` folder. Next we must visit the soap administration interface URL and select the Deploy option. Table 1 shows the required parameters and their values.

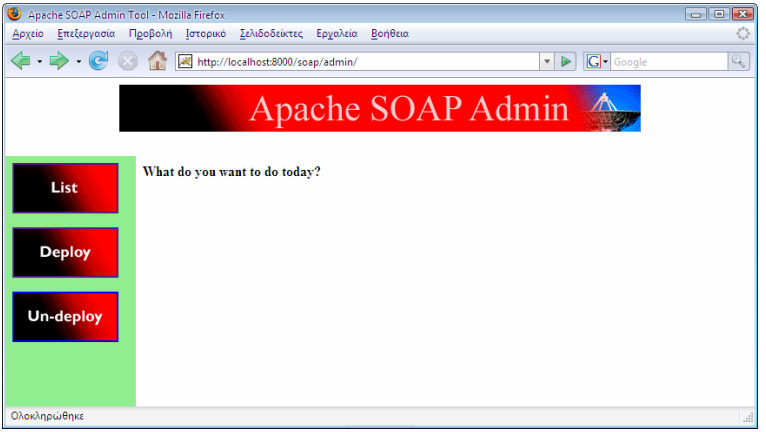

Figure 3 Apache SOAP administration tool

Service Deployment Descriptor

Property	Value
ID	webrdevice
Scope	Request
Provider Type	Java
Provider Class	WebRDevice
Use Static Class	False
Methods	ProcessRules
Type Mappings	
Default Mapping Registry Class	

Table 1 Web Service deployment settings.

In order to test and demonstrate our Web Service, we must create a web user interface so that the Web Service can become available to the end user. The web interface is depicted in figure 4. It is developed in JSP mainly because of the ability to reuse Java code for use over the web. The primary job of the interface is to create a SOAP request for the user using the parameters entered in a web form and then send it to the Web Service. Finally, it retrieves the SOAP response and locates the result file to finally present it to the user (figure 5).

The interface is designed in a way that it gives the user an understanding about what the service does. In *System Parameters* the user gives the deployment details of the service, first its location (the SOAP messages handler) and then its unique name (ID) on the server. In the *Run Parameters* section the user can either point the URL where the rule file is located, or upload it directly to the server. Optionally, the user can specify the transformation file for the according RuleML version of the rule file (if it is other than 0.85), the project name, and the time (in seconds) to keep the results in the server.

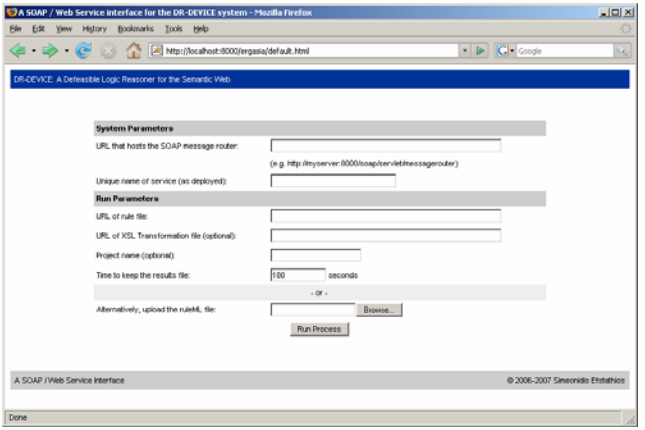

Figure 4 End-user Web interface (1)

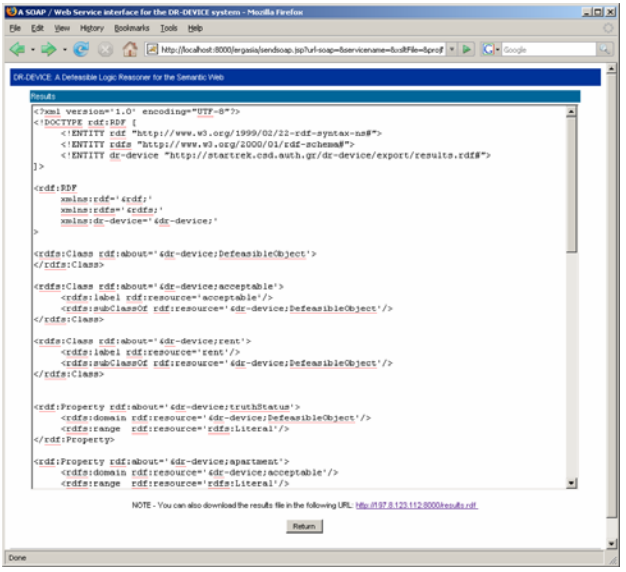

Figure 5 End-user Web interface (2)

5. Conclusions and Future Work

This paper presented Web Services and the Semantic Web, two rapidly evolving technologies that will play a key role in the future development of the Web. Both technologies aim at turning the WWW into a more automatic environment, where loosely coupled distributed systems will inter-communicate efficiently and software agents will have the ability to search for requested information automatically. The

paper also presented an imaginary scenario, which will allegedly take place in the (not so distant) future, involving intelligent software agents that utilize the above mentioned technologies.

The main contribution of the paper, however, is the introduction of a system that can potentially move the aforementioned scenario one step closer to realization. The software is a Web Service implementation of an existing system, called DR-DEVICE, which is an RDF rule reasoner, capable of performing defeasible inference and exporting the results as RDF data. A client Web application that consumes the Web Service was also developed and is presented in this work.

Nevertheless, our work just starts here: our future goals involve a variety of tasks, with the priority given to turning the imaginary scenario of section 3 into a reality. Thus, our imminent future plans include the implementation of software agents that "talk" to each other and utilize the Web Service reasoner implementation we developed.

References

1. Antoniou, G., van Harmelen, F. "A Semantic Web Primer". MIT Press, 2004.
2. Bassiliades, N., "Semantic Web: Vision and Technologies", *2nd Int. Scientific Conf. on Computer Science*, Plamenka Borovska, Sofoklis Christofordis (Ed.), IEEE Computer Society, Bulgarian Section, 30th Sep-2nd Oct 2005, Halkidiki, Greece, 2005.
3. Bassiliades, N., Antoniou G., Vlahavas I., "A Defeasible Logic Reasoner for the Semantic Web", *Int. Journal on Semantic Web and Information Systems*, 2(1), pp. 1-41, 2006.
4. Bassiliades, N., Antoniou, G., Governatori, G., "Proof Explanation in the DR-DEVICE System", Proc. 1st International Conference on Web Reasoning and Rule Systems (RR 2007), Innsbruck, Austria, 7-8 June 2007, LNCS 4524, pp. 249–258, Springer-Verlag, 2007.
5. Berners-Lee, T., Hendler, J., and Lassila, O., "The Semantic Web", *Scientific American*, 284(5), pp. 34-43, 2001.
6. Boley, H., Tabet, S., and Wagner, G., "Design Rationale of RuleML: A Markup Language for Semantic Web Rules", *Proc. Int. Semantic Web Working Symp*., pp. 381-402, 2001.
7. Chappell, D., Jewell, T. "Java Web Services" First Edition. O'Reilly, 2002.
8. Daconta, M., Obrst, L., Smith, K. "The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management". Wiley Publishing Inc, 2003.
9. Passin, T. "Explorer's Guide to the Semantic Web". Manning Publications Co, 2004.
10. RDF: Recourse Description Framework, `http://www.w3.org/RDF/`
11. Rule Interchange Format Working Group, W3C, `http://www.w3.org/2005/rules/wg`
12. SOAP Version 1.2 Part 1: Messaging Framework, `http://www.w3.org/TR/soap12-part1/`
13. Web Ontology Language (OWL), `http://www.w3.org/2004/OWL/`
14. Web Services Description Language (WSDL) 1.1, `http://www.w3.org/TR/wsdl`