VDR-DEVICE: A Visual Editor for a Defeasible Logic RuleML-compatible Rule Language

Nick Bassiliades[1], Efstratios Kontopoulos[1], Grigoris Antoniou[2]
[1]Department of Informatics, Aristotle University of Thessaloniki
GR-54124 Thessaloniki, Greece
{nbassili, skontopo}@csd.auth.gr
[2]Institute of Computer Science, FO.R.T.H.
P.O. Box 1385, GR-71110, Heraklion, Greece
antoniou@ics.forth.gr

Abstract

RuleML is a promising standardization effort for rule languages for the Semantic Web. However, the RuleML syntax may appear too complex for many users. Furthermore, the interplay between various Semantic Web technologies and languages impose a demand for using multiple, diverse tools for building rule-based applications for the Semantic Web. In this demonstration we present VDR-Device, a visual RuleML-compliant rule editor and an integrated development environment for developing and using defeasible logic rule bases on top of RDF ontologies. The visual rule editor constrains the allowed vocabulary through analysis of the input RDF ontologies. The development environment is supported by an RDF-aware defeasible reasoning system. Defeasible reasoning is a rule-based approach for efficient reasoning with incomplete and inconsistent information. Such reasoning is useful for many applications in the Semantic Web, such as policies and business rules, agent brokering and negotiation, ontology and knowledge merging, etc., mainly due to interesting features, such as conflicting rules and rule priorities.

1 Introduction

Defeasible reasoning [Nute, 1987], a member of the non-monotonic reasoning family, constitutes a simple rule-based approach to reasoning with incomplete and conflicting information. Defeasible reasoning can represent facts, rules as well as priorities and conflicts among rules. Such conflicts arise, among others, from rules with exceptions, which are a natural representation for policies and business rules [Antoniou *et al.*, 1999]. And priority information is often implicitly or explicitly available to resolve conflicts among rules. Potential applications include security policies, business rules, personalization, brokering, bargaining and agent negotiations.

Although defeasible logic is certainly a very promising reasoning technology for the Semantic Web, the *development* of rule-based applications for the Semantic Web can be greatly compromised by two factors. First, defeasible logic is certainly not an end-user language but rather a developer's one, because its syntax may appear too complex. Furthermore, the interplay between various technologies and languages involved in such applications, namely defeasible reasoning, RuleML, and RDF, impose a demand for using multiple, diverse tools, which is a high burden even for the developer.

In this demonstration we present VDR-Device, a visual RuleML-compliant rule editor and an integrated development environment for developing and using defeasible logic rule bases on top of RDF ontologies. VDR-Device is supported by a defeasible reasoning system that processes RDF data and RDF Schema ontologies [Bassiliades *et al.*, 2004]. The rule editor helps users to develop a defeasible rule base by constraining the allowed vocabulary after analyzing the input RDF ontologies. Therefore, it removes from the user the burden of typing-in class and property names and prevents potential semantical and syntactical errors. The visualization of rules follows the tree model of RuleML.

2 The VDR-Device System

VDR-Device consists of two primary components:

1. DR-Device, which acts as the reasoning system, performing the processing of RDF schema and data, the inferencing and producing the results.
2. DRREd (Defeasible Reasoning Rule Editor) that serves both as a visual rule authoring tool and as a graphical integrated development environment, wrapped around the core reasoning system.

2.1 The DR-Device Reasoning System

The core reasoning system of VDR-Device is DR-Device [Bassiliades *et al.*, 2004] and consists of two primary components (Fig. 1): The *RDF loader/translator* and the *rule loader/translator*. The user can either develop a rule base with the help of the rule editor described in the following section, or he/she can load an already existing one. The rule base contains: (a) a set of rules, (b) the URL(s) of the input RDF document(s), which is forwarded to the RDF loader, (c) the names of the derived classes to be exported as results and (d) the name of the output RDF document.

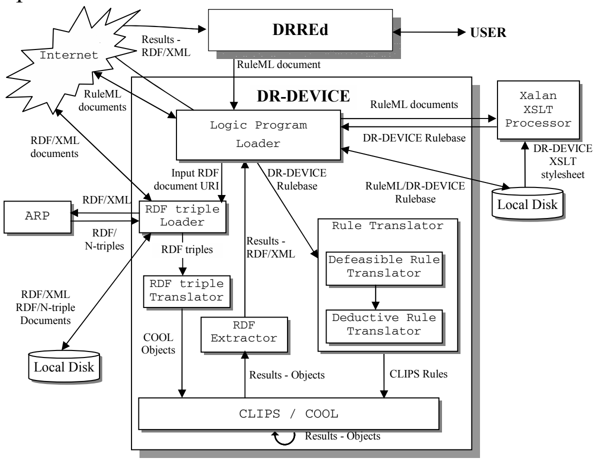

Fig. 1. The architecture of the core reasoning system.

The rule base is submitted to the *rule loader*, which transforms it into the native CLIPS-like syntax through an XSLT stylesheet and the resulting program is then forwarded to the *rule translator*, where the defeasible logic rules are compiled into a set of CLIPS production rules.

Meanwhile, the *RDF loader* downloads the input RDF documents, including their schemas, and translates RDF descriptions into CLIPS objects, according to the RDF-to-object translation scheme described in [Bassiliades and Vlahavas, 2004].

Finally, the result-objects are exported to the user as an RDF/XML document through the RDF extractor. The RDF document includes the instances of the exported derived classes, which have been proven.

2.2 The DRREd Rule Editor

Writing rules in RuleML can often be a highly cumbersome task; thus, the need for authoring tools that assist end-users in writing and expressing rules is imperative. VDR-Device is equipped with DRREd, a Java-built visual rule editor that aims at enhancing user-friendliness and efficiency during the development of VDR-Device RuleML documents. Its implementation is oriented towards simplicity of use and familiarity of interface.

The development of a rule base with DRREd follows the XML-tree format, the most intuitive means of displaying RuleML-like syntax. The user can navigate through the tree and add to or remove elements from it. However, since a rule base is backed by a DTD document, potential addition or removal of tree elements has to obey the DTD limitations. Therefore, the rule editor allows a limited number of operations performed on each element, according to its meaning within the rule tree.

The same principle is encountered in attribute processing. The values that the user can insert for each attribute are limited by the chosen attribute and node as well as the DTD specifications each time.

DRREd also permits namespace handling; namespace declarations are treated as addresses of input RDF Schema ontologies that contain the vocabulary for the input RDF documents, over which the rules of the rule base will be run. The namespaces entered by the user, as well as those contained in the input RDF documents are then analyzed in order to extract all the allowed class and property names for the rule base being developed. These names are then used throughout the authoring phase of the RuleML rule base, constraining the corresponding allowed names that can be applied and narrowing the possibility for errors on behalf of the user.

3 Conclusions and Future Work

In this demonstration we argued that although defeasible reasoning is useful for many applications in the Semantic Web, the development of defeasible rule bases on top of Semantic Web ontologies may appear complex to most users. To this end, we have implemented VDR-Device, a visual RuleML-compliant rule editor and a visual integrated development environment that facilitates the development of rule-based applications for the Semantic Web.

The system is freely available at the address: `iskp.csd.auth.gr/systems/dr-device.html`.

In the future, we plan to delve into the proof layer of the Semantic Web architecture by enhancing further the graphical environment with rule execution tracing, explanation, proof exchange in an XML or RDF format, proof visualization and validation, etc. These facilities would be useful for increasing the trust of users for the Semantic Web agents and for automating proof exchange and trust among agents in the Semantic Web.

Acknowledgments

This work is partially supported by the PYTHAGORAS II program which is jointly funded by the Greek Ministry of Education (EPEAEK) and the European Union.

References

[Antoniou *et al.*, 1999] Antoniou G., Billington D. and Maher M.J., "On the analysis of regulations using defeasible rules", *Proc. 32nd Hawaii International Conference on Systems Science*, 1999.

[Antoniou and van Harmelen, 2004] Antoniou G., Harmelen F. van, *A Semantic Web Primer*, MIT Press, 2004.

[Bassiliades *et al.*, 2004] Bassiliades N., Antoniou, G., Vlahavas I., "A Defeasible Logic Reasoner for the Semantic Web", *RuleML 2004*, Springer-Verlag, LNCS 3323, pp. 49-64, Hiroshima, 2004.

[Bassiliades and Vlahavas, 2004] Bassiliades N., Vlahavas I., "R-DEVICE: A Deductive RDF Rule Language", *RuleML 2004*, Springer-Verlag, LNCS 3323, pp. 65-80, Hiroshima, 2004.

[Nute, 1987] Nute D., "Defeasible Reasoning", *Proc. 20th Int. Conference on Systems Science*, IEEE Press, 1987, pp. 470-477.

Demo Explanation

This section presents a full example of using VDR-Device in a brokered trade application that takes place via an independent third party, the broker. The broker matches the buyer's requirements and the sellers' capabilities, and proposes a transaction when both parties can be satisfied by the trade. In our case, the concrete application (adopted from [Antoniou and van Harmelen, 2004]) is apartment renting and the landlord takes the role of the abstract seller.

Available apartments reside in an RDF document (Fig. 2). Each apartment has the following properties:

- `size` of the apartment (in m^2)
- number of `bedrooms` of the apartment
- `price` of the apartment
- `floor` of the apartment
- size of the garden (`gardenSize`) of the apartment
- existence of a `lift` in the house of the apartment
- pet allowance (`pets`) in the house of the apartment
- apartment location (`central` or not)

```
<carlo:apartment rdf:about="&carlo_ex;a1">
  <carlo:bedrooms rdf:datatype="&xsd;integer">1
  </carlo:bedrooms>
  <carlo:central>yes</carlo:central>
  <carlo:floor rdf:datatype="&xsd;integer">1
  </carlo:floor>
  <carlo:gardenSize rdf:datatype="&xsd;integer">0
  </carlo:gardenSize>
  <carlo:lift>no</carlo:lift>
  <carlo:name>a1</carlo:name>
  <carlo:pets>yes</carlo:pets>
  <carlo:price rdf:datatype="&xsd;integer">300
  </carlo:price>
  <carlo:size rdf:datatype="&xsd;integer">50
  </carlo:size>
</carlo:apartment>
```

Fig. 2. RDF document excerpt for available apartments.

The requirements of a potential renter, called e.g. Carlos, are the following:

- Carlos is looking for an apartment of at least 45m^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.
- Carlos is willing to pay \$300 for a centrally located 45m^2 apartment, and \$250 for a similar flat in the suburbs. In addition, he is willing to pay an extra \$5 per m^2 for a larger apartment, and \$2 per m^2 for a garden.
- He is unable to pay more than \$400 in total. If given the choice, he would go for the cheapest option. His 2nd priority is the presence of a garden; lowest priority is additional space.

These requirements are expressed in defeasible logic, in the RuleML-like syntax of VDR-Device. For example, Fig. 3 displays the RuleML-like format of rule r_4:`¬pets(X)⇒¬acceptable(X)`, which reads as "*If pets are not allowed, then the apartment is not acceptable*". Fig. 4 shows rule r_4 in the graphical rule editor.

```
<imp>
  <_rlab ruleID="r4" ruletype="defeasiblerule"
        superior="r1"/>
  <_head><neg><atom>
            <_opr><rel>acceptable</rel></_opr>
            <_slot name="apartment"><var>x</var>
            </_slot></atom></neg></_head>
  <_body><atom>
       <_opr><rel href="carlo:apartment"/></_opr>
       <_slot name="carlo:name"><var>x</var>
       </_slot>
       <_slot name="carlo:pets"><ind>"no"</ind>
       </_slot>
     </atom></_body>
</imp>
```

Fig. 3. A sample rule.

The set of Carlo's requirements forms a rule base document, which is loaded into VDR-DEVICE. It is initially transformed into the native DR-DEVICE syntax [Bassiliades et al., 2004]. DR-DEVICE rules are further translated into R-DEVICE rules [Bassiliades and Vlahavas, 2004], which in turn are translated into CLIPS production rules. Then the RDF document is loaded and transformed into CLIPS (COOL) objects. Finally, the reasoning takes place and ends up with 3 acceptable apartments and one suggested apartment for renting, according to Carlo's requirements and the available apartments.

The results (i.e. objects of derived classes) are exported in an RDF file according to the specifications posed in the RuleML document. Both the positively and negatively proven (defeasibly or definitely) objects are exported, while objects that cannot be at least defeasibly proven, either negatively or positively, are not exported, although they exist inside DR-DEVICE. Furthermore, the RDF schema of the derived classes is also exported.

Users can examine all the exported results and the execution trace of compilation and running.

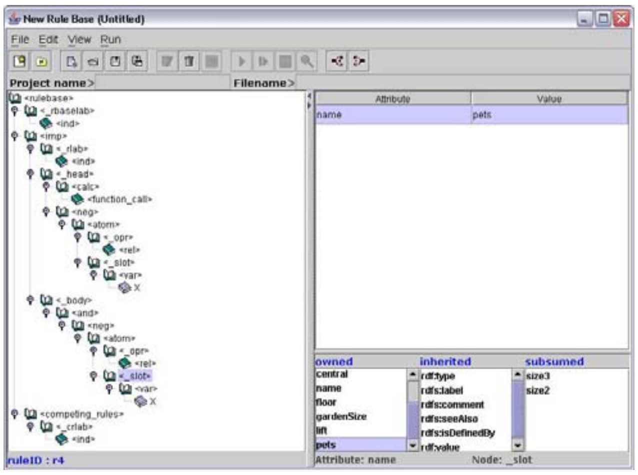

Fig. 4. The graphical rule editor.