Proof Explanation in the DR-DEVICE System

Nick Bassiliades[1], Grigoris Antoniou[2] and Guido Governatori[3]

[1] Aristotle University of Thessaloniki, Greece
nbassili@csd.auth.gr
[2] FORTH-ICS, Greece and University of Crete, Greece
antoniou@ics.forth.gr
[3]University of Queensland, Australia
guido@itee.uq.edu.au

Abstract. Trust is a vital feature for the Semantic Web: If users (humans and agents) are to use and integrate system answers, they must trust them. Thus, systems should be able to explain their actions, sources, and beliefs, and this issue is the topic of the proof layer in the design of the Semantic Web. This paper presents the design of a system for proof explanation on the Semantic Web, based on defeasible reasoning. The basis of this work is the DR-DEVICE system that is extended to handle proofs. A critical aspect is the representation of proofs in an XML language, which is achieved by a RuleML language extension.

1 Introduction

The development of the Semantic Web proceeds in steps, each step building a layer on top of another. At present, the highest layer that has reached sufficient maturity is the ontology layer in the form of the description logic-based language OWL [8]. The next step in the development of the Semantic Web will be the logic and proof layers. The implementation of these two layers will allow the user to state any logical principles, and permit the computer to infer new knowledge by applying these principles on the existing data. Rule systems appear to lie in the mainstream of such activities.

Many recent studies have focused on the integration of rules and ontologies, and various solutions have been proposed. The Description Logic Programs is the approach followed in [13]; DLPs derive from the intersection of Description Logics and Horn Logic, and enable reasoning with available efficient LP inferencing algorithms over large-scale DL ontologies. We also distinguish the approaches presented in [16] and [20], which study the integration of Description Logics and Datalog rules. Two representative examples of rule languages for the Semantic Web are TRIPLE [22] and SWRL [14]. They both provide a model for rules on the Semantic Web. TRIPLE is based on F-Logic and provides support for RDFS and a subset of OWL Lite, while SWRL extends OWL DL with Horn-style rules.

Different, but equally interesting research efforts, deal with the standardization of rules for the Semantic Web. Works in this direction include (a) the RuleML Markup Initiative [9], whose ultimate goal is to develop a canonical Web language for rules using XML markup, formal semantics, and efficient implementations; and (b) the research

M. Marchiori, J.F. Pan and C. de Sainte Marie (eds)
Web Reasoning and Rule Systems. LNCS 4524, pp. 249-258

conducted by the Rule Interchange Format (RIF) Working Group, which was recently launched by W3C.

Apart from classical rules that lead to monotonic logical systems, recently researchers started to study systems capable of handling conflicts among rules and reasoning with partial information. Recently developed nonmonotonic rule systems for the Semantic Web include DR-Prolog [1], SweetJess [12], dlvhex [10] and DR-DEVICE [5], a defeasible reasoning system for the Semantic Web, implemented in CLIPS, which integrates well with RuleML and RDF.

The upper levels of the Semantic Web have not been researched enough and contain critical issues, like accessibility, trust and credibility. The next step in the architecture of the Semantic Web is the proof layer and little has been written and done for this layer. The main difference between a query posed to a traditional database system and a semantic web system is that the answer in the first case is returned from a given collection of data, while for the semantic web system the answer is the result of a reasoning process. While in some cases the answer speaks for itself, in other cases the user will not be confident in the answer unless he/she can trust the reasons why the answer has been produced. In addition it is envisioned that the semantic web is a distributed system with disparate sources of information. Thus a semantic web answering system, to gain the trust of a user must be able, if required, to provide an explanation or justification for an answer. Since the answer is the result of a reasoning process, the justification can be given as a derivation of the conclusion with the sources of information for the various steps.

In this work we describe the design of an extension of the nonmonotonic rules system DR-DEVICE, to extract and present explanations of answers. This work can be viewed as a contribution to the realization of a proof layer for a nonmonotonic rule language on the semantic web.

2 Defeasible Logics

The root of defeasible logics lies on research in knowledge representation, and in particular on inheritance networks. Defeasible logics can be seen as inheritance networks expressed in a logical rules language. In fact, they are the first nonmonotonic reasoning approach designed from its beginning to be implementable.

Being nonmonotonic, defeasible logics deal with potential conflicts (inconsistencies) among knowledge items. Thus they contain classical negation, contrary to usual logic programming systems. They can also deal with negation as failure (NAF), the other type of negation typical of nonmonotonic logic programming systems; in fact, [24] argues that the Semantic Web requires both types of negation. In defeasible logics, often it is assumed that NAF is not included in the object language. However, as [3] argues, it can be easily simulated when necessary. Thus, we may use NAF in the object language and transform the original knowledge to logical rules without NAF exhibiting the same behavior.

Conflicts among rules are indicated by a conflict between their conclusions. These conflicts are of local nature. The simpler case is that one conclusion is the negation of the other. The more complex case arises when the conclusions have been declared to be mutually exclusive, a very useful representation feature in practical applications.

Defeasible logics are skeptical in the sense that conflicting rules do not fire. Thus consistency of drawn conclusions is preserved.

Priorities on rules may be used to resolve some conflicts among rules. Priority information is often found in practice, and constitutes another representational feature of defeasible logics.

The logics take a pragmatic view and have low computational complexity. This is, among others, achieved through the absence of disjunction and the local nature of priorities: only priorities between conflicting rules are used, as opposed to systems of formal argumentation where often more complex kinds of priorities (e.g. comparing the strength of reasoning chains) are incorporated.

Generally speaking, defeasible logics are closely related to Courteous Logic Programs [11], as discussed in, e.g., [5].

The Language

A *defeasible theory D* is a couple $(R,>)$ where R a finite set of rules, and $>$ a superiority relation on R. Rules containing free variables are interpreted as the set of their variable-free instances.

There are three kinds of rules: *Strict rules* are denoted by $A \rightarrow p$, and are interpreted in the classical sense: whenever the premises are indisputable then so is the conclusion. An example of a strict rule is "Professors are faculty members". Written formally: `professor(X)` → `faculty(X)`. Inference from strict rules only is called *definite inference*. Strict rules are intended to define relationships that are definitional in nature, for example ontological knowledge.

Defeasible rules are denoted by $A \Rightarrow p$, and can be defeated by contrary evidence. An example of such a rule is `faculty(X)` ⇒ `tenured(X)` which reads as follows: "Professors are typically tenured".

Defeaters are denoted as $A \sim> p$ and are used only to prevent some conclusions, not to actively support conclusions. An example of such a defeater is `assistantProf(X)` `~>` ¬`tenured(X)` which reads as follows: "Assistant professors may be not tenured".

A *superiority relation* on R is an acyclic relation $>$ on R (that is, the transitive closure of $>$ is irreflexive). When $r_1 > r_2$, then r_1 is called *superior* to r_2, and r_2 *inferior* to r_1. This expresses that r_1 may override r_2. For example, given the defeasible rules

```
r:  professor(X) =>  tenured(X)
r': visiting(X)  => ¬tenured(X)
```

which contradict one another, no conclusive decision can be made about whether a visiting professor is tenured. But if we introduce a superiority relation $>$ with r' > r, then we can indeed conclude that a visiting professor is not tenured.

The system works roughly in the following way: to prove a conclusion A defeasibly, there must be a firing rule with A as its head (that is, all literals in the rule body have

already been proved); in addition, we must rebut all attacking rules with head the (strong) negation of *A*. For each such attacking rule we must establish either (a) that this rule cannot fire because we have already established that one of the literals in its body cannot be proved defeasibly (finite failure), or (b) that there is a firing rule with head *A* superior to the attacking rule.

A formal definition of the proof theory is found in [3]. A model theoretic semantics is found in [17].

3 System Functionality

In this section we mainly concentrate on the functionality of the proof explanations facility of the DR-DEVICE system (Fig. 1). More details on the architecture and the implementation of the system can be found in [5].

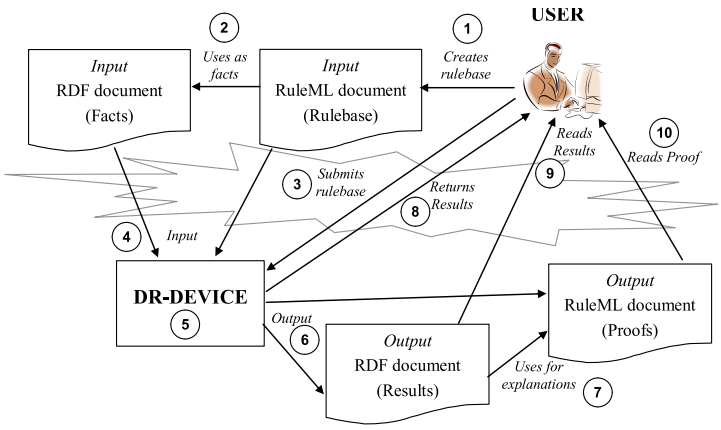

Fig. 1. Functionality of the DR-DEVICE system.

The DR-DEVICE system accepts as input a defeasibe logic rulebase (step 4) in a RuleML-like syntax [9]. The rulebase has been created by a user (step 1) and its address is submitted to the DR-DEVICE system (step 3) through the stand-alone user interface of the system [6], or through a web-based interface hat we are currently developing. The rulebase contains only rules; the facts for the rule program are (input) RDF documents, whose addresses are declared in the rulebase header (step 2). The rule conclusions are materialized inside DR-DEVICE as objects (step 5) and when the inference procedure terminates, the instances of designated derived classes are exported as an RDF document (step 6). The RDF document includes:

- The RDF Schema definitions for the exported derived classes.
- Those instances of the exported derived classes, which have been proven, either positively or negatively, either defeasibly or definitely.

Furthermore, the system exports the grounds for all derived objects in a separate RuleML document (steps 6, 7). To this end we have extended RuleML with an XML schema for proofs of both classically (definitely) derived objects and defeasibly derived

objects, which is discussed in the next section. DR-DEVICE returns to the user (step 8) the address of the RDF document with the results and the address of the RuleML document with the proof traces. Finally, the user can access the results (step 9) and the proofs (step 10) through a web browser or through a specialized software that can customize the visualization. Notice, that DR-DEVICE can also provide explanations about non-proved objects.

4 Proof Schema

The XML Schema for proof traces[1] explaining DR-DEVICE's results is an extension of the RuleML's 0.91 schema[2]. Actually, the rule language of DR-DEVICE is also an extension of RuleML. Extensions (for the rule language) deal with two aspects of DR-DEVICE, namely defeasible logic and its CLIPS implementation. Defeasible logic extensions include rule types, superiority relation among rules and conflicting literals, while CLIPS-related extensions deal with constraints on predicate arguments and functions. More details about the rule language can be found in [5].

The top-level element of the proof schema is the optional `Grounds` element, which is a top-level element of a RuleML document, although it should actually be an alternative to an `Assert` element. The latter could not be achieved using the redefinition mechanism of XML Schema, since element extensions deal only with sequences and not choices. Grounds consist of multiple proved or even not proved rule conclusions. Proofs can be either definite, i.e. using classical strict rules, or defeasible, which can use all three rule types of defeasible logic.

Definitely proved literals consist of the literal itself and the definite proof tree. The literal can be a positive atom or its negation, or even a reference to an RDF resource. Notice that DR-DEVICE uses RDF resources as facts and its conclusions are also materialized as RDF resources. A literal is definitely proved if there is a strict clause, either a strict rule or a fact, whose body literals are also definitely proven. Rules can either be in-lined in the proof tree or an external reference can exist to rules in another RuleML document. Similarly, the proofs for body literals can either be encapsulated in the proof tree of the rule head or can be referenced from another place of the proof document.

On the other hand, defeasible proofs are more complicated since they require either a defeasible or a strict rule (collectively called supportive rules), whose body literals are defeasibly proven. Notice that a definite proof subsumes a defeasible proof, that is why the `Definite_Proof` element is an alternative to the `Defeasible_Proof` element. Furthermore, the defeasibe conclusion must not be strongly attacked, i.e. the negation of the conclusion must not be definitely proved. Finally, the rules that defeasibly attack the current one must all be blocked, so the defeasible conclusion of this rules prevails.

A rule can be blocked in three ways. A defeasible rule (or a defeater) is blocked either when its body literals are not defeasibly proven or when it is attacked by another supe-

---

[1] `http://lpis.csd.auth.gr/systems/dr-device/dr-device-0.91.xsd`

[2] `http://www.ruleml.org/0.91/xsd/nafnegdatalog.xsd`

rior defeasible rule, whose body literals are defeasibly proven. A strict rule is blocked if its body literals are not definitely proven. Finally, inferior defeasible rules are considered as blocked.

Not proved conclusions follow a similar structure, i.e. the supportive rule that could not prove something must be included along with the reason why this happened. In the case of a defeasible non-proof, reasons include either the non-proof of some of the body literals or a definitely proved negated literal or an undefeated defeasible attacker. A defeasible attacker can be a defeasible rule or a defeater, whose body literals are proven and whose possible attackers have been blocked. Notice that in order for a conclusion to not be defeasibly provable it must also be not definitely provable. The latter is similar to the blocked strict rule case above.

5 Proof Example

In this section we include a full example of the functionality of DR-DEVICE concerning both the inferencing and proof provision procedures. Assume that the user wants to submit the following rulebase (shown in simple logical notation) and wants to find out why the conclusion `rich(antonis)` is defeasibly derived.

`wins_lotto(antonis)` `owns(antonis)`

r_1: `wins_lotto(X)` ⇒ `rich(X)` r_2: `paid_well(X)` ⇒ `rich(X)`

r_3: `owns(X)` ⇒ ¬`rich(X)` $r_1 > r_3$

r_4: `gamble(X)` ⇒ ¬`rich(X)`

The rulebase is submitted to DR-DEVICE as a RuleML document (Fig. 2). Notice that facts are not directly included in the RuleML document but in a separate input RDF document (Fig. 3), as indicated by the `rdf_input` attribute of the top-level `RuleML` element in Fig. 2. The `rdf_export_classes` attribute indicates which are the exported conclusions, the `rdf_export` attribute designates the output RDF document (Fig. 4) and the `proof` attribute designates the output RuleML document (Fig. 5) that contains the proofs for the exported conclusions.

DR-DEVICE atoms follow an object-oriented structure; the operator is a class name and the arguments are named slots. The DR-DEVICE system employs an object-oriented RDF data model ([5], [7]), where properties as normal encapsulated attributes of resource objects. The operator of an atom corresponds to the type of an RDF resource, the `oid` element to the URI of the resource and the slot arguments to the resource's properties.

The exported results in Fig. 4 contain the materialization of the derived object as an RDF resource, which also contains some system-defined properties, such as `truthStatus` that indicates if the conclusion was definitely or defeasibly proven, and `proof` that references the proof ID of the corresponding proof tree in the output proof document (Fig. 5). The latter indicates that the corresponding RDF resource was defeasibly proved using defeasible rule r_1, whose body literal was also defeasibly proved via a definitive proof due to the existence of a fact (RDF resource of the input RDF document). Fur-

thermore, the negated conclusion was not definitely proven, because there are no appropriate strict rules, which is indicated by the fact that the `not_strongly_attacked` element is empty. Finally, defeasible rules r_3 and r_4 which attack r_1 are both blocked; r_3 is blocked because it is attacked by the superior rule r_1 and r_4 is blocked because its body literal cannot be proved.

```
<RuleML rdf_import="http://.../ex1.rdf" rdf_export_classes="rich"
       rdf_export="export-ex1.rdf" proof="http://.../proof-ex1.ruleml"
       xsi:schemaLocation="http://www.ruleml.org/0.91/xsd
                           http://.../dr-device/dr-device-0.91.xsd">
  <Assert>
    <Implies ruletype="defeasiblerule">
      <oid><Ind uri="&ex_rb;r1">r1</Ind></oid>
      <head>  <Atom>  <op><Rel>rich</Rel></op>
                      <slot><Ind>person</Ind> <Var>x</Var></slot> </Atom> </head>
      <body>  <Atom>  <op><Rel uri="ex:person"/></op>
                <slot><Ind>ex:name</Ind><Var>x</Var></slot>
                <slot>  <Ind>ex:wins_lotto</Ind>
                        <Data xsi:type="xs:string">true</Data>  </slot> </Atom> </body>
      <superior>  <Ind uri="&ex_rb;r3"/>  </superior>
    </Implies>
...
    <Implies ruletype="defeasiblerule">
      <oid><Ind uri="&ex_rb;r3">r3</Ind></oid>
      <head>  <Neg> <Atom>  <op><Rel>rich</Rel></op>
                      <slot><Ind>person</Ind><Var>x</Var></slot>  </Atom> </Neg>  </head>
      <body>  <Atom>  <op><Rel uri="ex:person"/></op>
                <slot><Ind>ex:name</Ind><Var>x</Var></slot>
                <slot>  <Ind>ex:owns</Ind>
                        <Data xsi:type="xs:string">true</Data>  </slot> </Atom> </body>
    </Implies>
...
  </Assert>
</RuleML>
```

Fig. 2. Rulebase example parts.

```
<rdf:RDF ... >
   <ex:person   rdf:ID="Inst_6"
      ex:name="antonis"       ex:owns="false"
      ex:paid_well="true"     ex:wins_lotto="true"/>
</rdf:RDF>
```

Fig. 3. Input RDF document example.

```
<rdf:RDF  xmlns:defeasible="http://.../defeasible.rdfs#"
          xmlns:dr-device="http://.../export-ex1.rdf#" ... >
...
  <dr-device:rich rdf:about="http://.../export-ex1.rdf#rich1">
    <dr-device:person>antonis</dr-device:person>
    <defeasible:truthStatus>defeasibly-proven</defeasible:truthStatus>
    <defeasible:proof
        rdf:datatype="&xsd;anyURI">'http://.../proof-ex1.ruleml#proof1'</defeasible:proof>
  </dr-device:rich>
</rdf:RDF>
```

Fig. 4. Output RDF document example.

6 Related Work

Besides teaching logic [4], not much work has been centered around explanation in reasoning systems so far. Rule-based expert systems have been very successful in applications of AI, and from the beginning, their designers and users have noted the need for

explanations in their recommendations. In expert systems like [21] and Explainable Expert System [23], a simple trace of the program execution rule firing appears to provide a sufficient basis on which to build an explanation facility and they generate explanations in a language understandable to its users.

```
<RuleML rdf_import="http://.../ex1.rdf"         rdf_export="http://.../export-ex1.rdf"
        rulebase="http://.../dr-device/proof/ex/ex1.ruleml"
        xsi:schemaLocation="http://www.ruleml.org/0.91/xsd http://.../dr-device-0.91.xsd">
  <Grounds>
    <Proved>
      <Defeasibly_Proved> <oid><Ind uri="&pr_ex;proof1">proof1</Ind></oid>
        <Literal> <RDF_resource uri="http://.../export-ex1.rdf#rich1"/>
        <Defeasible_Proof>
          <supportive_rule> <rule_ref rule="&ex_rb;r1"/>  </supportive_rule>
          <defeasible_body_grounds>
            <Defeasibly_Proved>
              <Literal> <Atom>  <op><Rel uri="ex:person"/></op>
                          <slot>  <Ind>ex:name</Ind>
                                  <Data xsi:type="xs:string">Antonis</Data></slot>
                          <slot>  <Ind>ex:wins_lotto</Ind>
                                  <Data xsi:type="xs:string">true</Data>  </slot> </Atom>
              </Literal>
              <Definite_Proof>
                <strict_clause>
                  <Fact>  <RDF_resource uri="http://...ex1.rdf#Inst_6"/> </Fact>
                </strict_clause>
              </Definite_Proof>
            </Defeasibly_Proved>
          </defeasible_body_grounds>
          <not_strongly_attacked/>
          <defeasible_attackers_blocked>
            <Blocked>
              <Blocked_Defeasible_rule>
                <rule_ref rule="&ex_rb;r3"/>
                <Attacked_by_Superior>  <rule_ref rule="&ex_rb;r1"/>
                </Attacked_by_Superior> </Blocked_Defeasible_rule>  </Blocked>
            <Blocked>
              <Blocked_Defeasible_rule>
                <rule_ref rule="&ex_rb;r4"/>
                <not_defeasible_body_grounds>
                  <Not_Defeasibly_Proved>
                    <Literal> <Atom>  <op><Rel uri="ex:person"/></op>
                                <slot>  <Ind>ex:name</Ind>
                                        <Data xsi:type="xs:string">Antonis</Data></slot>
                                <slot>  <Ind>ex:gambles</Ind>
                                        <Data xsi:type="xs:string">true</Data>  </slot>
                    </Atom>   </Literal>
                    <Not_Defeasible_Proof/>
                    <Not_Definite_Proof/>
...
</RuleML>
```

Fig. 5. Proof example.

Work has also been done in explaining the reasoning in description logics [18]. This research presents a logical infrastructure for separating pieces of logical proofs and automatically generating follow-up queries based on the logical format.

The most prominent work on proofs in the Semantic Web context is *Inference Web* [19]. The Inference Web (IW) is a Semantic Web based knowledge provenance infrastructure that supports interoperable explanations of sources, assumptions, learned information, and answers as an enabler for trust. It supports provenance, by providing proof metadata about sources, and explanation, by providing manipulation trace information. It also supports trust, by rating the sources about their trustworthiness.

IW simply requires inference rule registration and PML format. It does not limit itself to only extracting deductive engines. It provides a proof theoretic foundation on which

to build and present its explanations, but any question answering system may be registered in the Inference Web and thus explained. So, in order to use the Inference Web infrastructure, a question answering system must register in the IWBase its inference engine along with its supported inference rules, using the PML specification format. The IW supports proof generation service that facilitates the creation of PML proofs by inference engines.

Closest to this paper is the work [2] that also focuses on explanation extraction and presentation for defeasible reasoning on the semantic web, but relies on an XSB-based reasoning engine and is embedded in a multi-agent environment, while it provides few details regarding the extensions of RuleML.

7 Conclusion and Future Work

This work presented a new system that aims to increase the trust of the users for Semantic Web applications. The system automatically generates an explanation for every answer to user's queries, in a formal and useful representation. It can be used by individual users who want to get a more detailed explanation from a reasoning system in the Semantic Web, in a more human readable way. Also, an explanation could be fed into a proof checker to verify the validity of a conclusion; this is important in a multi-agent setting. Our reasoning system is based on defeasible logic (a nonmonotonic rule system) and we used the related reasoning engine DR-DEVICE. One contribution of our work is a RuleML extension for a formal representation of an explanation using defeasible logic.

In future work, we intend to improve the explanation facility to make it more intuitive and human-friendly, to suit users unfamiliar with logic. This effort includes proof visualization and visual rule execution tracing through integrating the work described in this paper with a tool for rule visualization [15] we have developed. Also, integration with the Inference Web infrastructure will be explored. Finally, we will investigate the use of the system in semantic web applications in which explanation and trust are essential elements.

Acknowledgments

This work was partially supported by the REWERSE Network of Excellence, and a GSRT Greek-Australian Project "Defeasible Reasoning for Semantic Web e-Commerce Applications".

References

[1] Antoniou G., Bikakis A., "DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web", *IEEE Tran. on Knowledge and Data Engineering*, 19(2), pp. 233-245, 2007.

[2] Antoniou G. et al. "Proof Explanation for the Semantic Web Using Defeasible Logic", *submitted*.

[3] Antoniou G., Billington D., Governatori G. and Maher M.J., "Representation results for defeasible logic", *ACM Trans. on Computational Logic*, 2(2), 2001, pp. 255-287.

[4] Barwise J. and Etchemendy J., *The Language of First-Order Logic*. Center for the study of Language and Information 1993.

[5] 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.

[6] Bassiliades N., Kontopoulos E., Antoniou G., "A Visual Environment for Developing Defeasible Rule Bases for the Semantic Web", *Proc. RuleML-2005*, pp. 172-186, Springer-Verlag, LNCS 3791, Galway, Ireland, 2005.

[7] Bassiliades N., Vlahavas I., "R-DEVICE: An Object-Oriented Knowledge Base System for RDF Metadata", *Int. Journal on Semantic Web and Information Systems*, 2(2), pp. 24-90, 2006.

[8] Bechhofer S., van Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A., *OWL web ontology language reference*, `www.w3.org/TR/ owl-ref/`, W3C Recommendation, 10 February 2004.

[9] Boley H., Tabet S., *The Rule Markup Initiative*, `www.ruleml.org`.

[10] Eiter T., Ianni G., Schindlauer R., Tompits H., "dlvhex: A System for Integrating Multiple Semantics in an Answer-Set Programming Framework.". *Proc. WLP 2006*, pp. 206-210.

[11] Grosof B. N., "Prioritized conflict handing for logic programs", *Proc. of the 1997 Int. Symposium on Logic Programming*, pp. 197-211, 1997.

[12] Grosof B. N., Gandhe M. D. and Finin T. W., "SweetJess: Translating DAMLRuleML to JESS", *Proc. RuleML 2002*.

[13] Grosof B. N., Horrocks I., Volz R. and Decker S., "Description Logic Programs: Combining Logic Programs with Description Logic", *Proc. 12th Intl. Conf. on the World Wide Web (WWW-2003)*, ACM Press, 2003, pp. 48-57.

[14] Horrocks I., Patel-Schneider P. F., Bechhofer S., Tsarkov D., "OWL Rules: A Proposal and Prototype Implementation", *Journal of Web Semantics*, 3(1), pp. 23-40, 2005.

[15] Kontopoulos E., Bassiliades N., Antoniou G., "Visualizing Defeasible Logic Rules for the Semantic Web", *1st Asian Semantic Web Conf. (ASWC'06)*, Beijing, China, 2006, Springer-Verlag, LNCS 4185, pp. 278-292.

[16] Levy A. and Rousset M.-C., "Combining Horn rules and description logics in CARIN", *Artificial Intelligence*, 104(1-2), 1998, pp. 165 – 209.

[17] Maher M.J., "A Model-Theoretic Semantics for Defeasible Logic", *Proc. Workshop on Paraconsistent Computational Logic*, pp. 67-80, 2002.

[18] McGuinness D. L. and Borgida A., "Explaining Subsumption in Description Logics", *Proc. IJCAI 1995*, pp. 816-821.

[19] McGuinness D. L. and da Silva P., "Explaining answers from the Semantic Web: the Inference Web approach", Journal of Web Semantics, 1(4), 2004, pp. 397-413.

[20] Rosati R., "On the decidability and complexity of integrating ontologies and rules", *Journal of Web Semantics*, 3(1), 2005, pp. 41-60.

[21] Shortliffe E., *Computer-based medical consultations: MYCIN*, Elsevier, 1976.

[22] Sintek M. and Decker S., "TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web", *Proc. Int. Semantic Web Conference*, 2002, pp. 364-378.

[23] Swartout W., Paris C. and Moore J., "Explanations in Knowledge Systems: Design for Explainable Expert Systems", *IEEE Expert*, 6(3), 1991, pp. 58-64.

[24] Wagner G., "Web Rules Need Two Kinds of Negation", *In Proc. First Workshop on Semantic Web Reasoning*, LNCS 2901, Springer 2003, pp. 33-50.