DR-DEVICE: A Defeasible Logic RDF Rule Language

Nick Bassiliades[1], Grigoris Antoniou[2], and Ioannis Vlahavas[1]

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

1. Introduction

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 modeling of business rules and policies, where rules with exceptions are often used. In this demonstration we present a prototype system for defeasible reasoning on the Web. The system is called DR-DEVICE [4] and is capable of reasoning about RDF metadata over multiple Web sources using defeasible logic [1] rules. The system is implemented on top of CLIPS production rule system and builds upon R-DEVICE [5], an earlier 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 [9] syntax. The operational semantics of defeasible logic are implemented through compilation into the generic rule language of R-DEVICE. This demonstration includes a complete use case of a semantic web broker that reasons about apartment renting.

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.

2. DR-DEVICE System Architecture

The DR-DEVICE system consists of two major components (Fig. 1): the *RDF loader/translator* and the *rule loader/translator*. The former accepts from the latter (or the user) requests for loading RDF documents. The *RDF triple loader* downloads the RDF document from the Internet and uses the ARP parser [10] to translate it to triples in the N-triple format. Both the RDF/XML and N-triple files are stored locally for future reference. Furthermore, the RDF document is recursively scanned for namespaces which are also translated. The rationale for translating namespaces is to obtain a more detailed RDF Schema. Fetching multiple RDFS files aggregates multiple RDF-to-OO schema translations into a single OO schema redefinition. If namespaces are not RDFS documents, then the parser does not produce triples and DR-DEVICE will make assumptions, based on the RDF semantics about non-resolved properties, resources, classes, etc. All N-triples are loaded into memory, while the resources that have a `URI#anchorID` format are transformed into a `ns:anchorID` format if `URI` belongs to the initially collected namespaces, to save memory space. The transformed RDF triples are fed to the *RDF triple translator* which maps them into COOL objects and then deletes them.

The *rule loader* accepts from the user a URI that contains a defeasible logic rule program in RuleML notation. The RuleML document may also contain the URI of the input RDF document on which the rule program will run, which is forwarded to the RDF loader. The RuleML program is translated into the native DR-DEVICE rule notation using the Xalan XSLT processor [11] and an XSLT stylesheet. The DR-DEVICE rule program is then forwarded to the rule translator. The *rule translator* accepts from the rule loader (or directly from the user) a set of rules in DR-DEVICE notation and translates them into a set of CLIPS production rules. The translation of the defeasible logic rules is performed in two steps: first, the defeasible logic rules are translated into sets of deductive, derived attribute and aggregate attribute rules of the basic R-DEVICE rule language, and then, all these rules are translated into CLIPS production rules ([5]). When the translation ends, CLIPS runs the production rules and generates the objects that constitute the result of the initial rule program or query. Finally, the result-objects are exported to the user as an RDF/XML document through the RDF extractor.

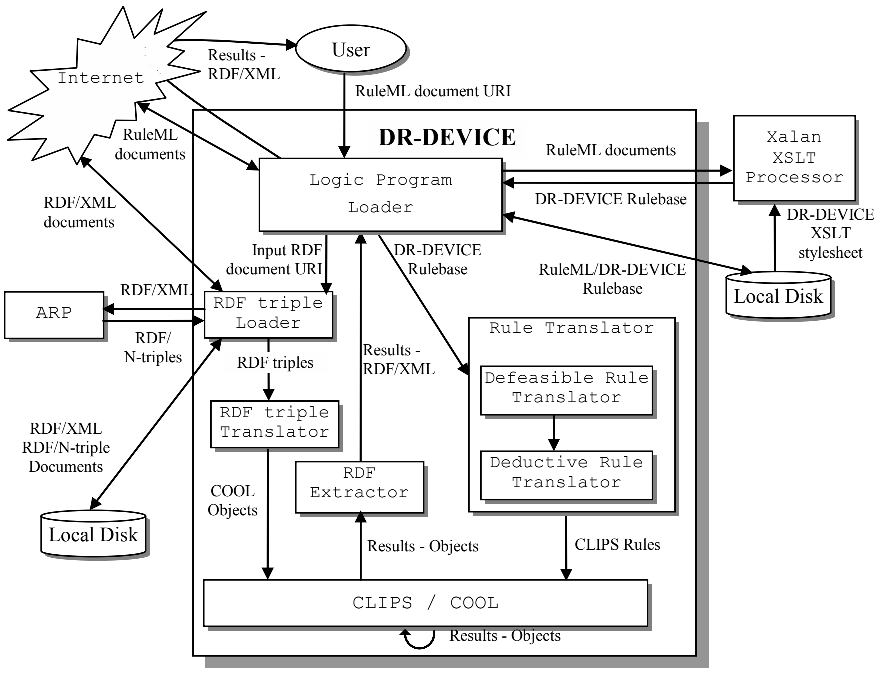

Fig. 1. Architecture of the DR-DEVICE system.

3. The R-DEVICE System

R-DEVICE ([5]) is a deductive object-oriented knowledge base system, which transforms RDF triples into objects and uses a deductive rule language for querying and reasoning about them. R-DEVICE includes features such as normal and generalized path expressions, stratified negation, aggregate, grouping, and sorting, functions. The rule language supports a second-order syntax, where variables can range over classes and properties. However, second-order variables are compiled away into sets of first-order rules, using instantiations of the metaclasses. Users can define views which are materialized and, optionally, incrementally maintained by translating deductive rules into CLIPS production rules. Users can choose between an OPS5/CLIPS-like or a RuleML-like syntax. Examples of rules can be found in [8]. R-DEVICE belongs to a family of previous such deductive object-oriented rule languages ([6]).

R-DEVICE is based on a OO RDF data model, different than the established graph model, which maps resources to objects and encapsulates properties inside resource objects, as traditional OO attributes. In this way, less joins are required to access the properties of a single resource resulting in better inferencing/querying performance. The descriptive semantics of RDF may call for dynamic re-definitions of resource classes and objects, which are handled by R-DEVICE effectively. The main features of this mapping scheme are the following:

- Resource *classes* are represented both as COOL classes and as direct or indirect instances of the `rdfs:Class` class.
- All *resources* are represented as COOL objects, direct or indirect instances of the `rdfs:Resource` class.
- *Properties* are direct or indirect instances of the class `rdf:Property`. Furthermore, properties are defined as slots (attributes) of their domain class(es). The values of properties are stored inside resource objects as slot values.

4. The Rule Language of DR-DEVICE

There are three types of rules in DR-DEVICE, closely reflecting defeasible logic: strict rules, defeasible rules, and defeaters. For example, the rule construct in Fig. 2 represents the following defeasible rule, which is adopted from the use case in next section:

```
r2: apartment(X), bedrooms(X,Y), Y<2 => ¬acceptable(X).
```

```
(defeasiblerule r2
      (declare (superior r1))
      (carlo:apartment (carlo:name ?x) (carlo:bedrooms ?y&:(< ?y 2)))
  =>
      (not (acceptable (apartment ?x))
)
```

Fig. 2. Sample defeasible DR-DEVICE rule in CLIPS-like syntax.

Predicates have named arguments, called slots, since they represent CLIPS objects. DR-DEVICE has also a RuleML-like syntax. The same rule is represented in RuleML notation (version 0.85) as shown in Fig. 3. Several features of defeasible logic and its DR-DEVICE implementation could not be captured by the latest RuleML DTDs, so we have developed a new DTD using the modularization scheme of RuleML, extending the Datalog with negation DTD (both classical and NAF) with OO features.

```
<!DOCTYPE rulebase SYSTEM "http://.../dr-device/defeasible.dtd" [
   <!ENTITY carlo "http://.../dr-device/carlo/carlo.rdf#">
   <!ENTITY carlo_rb "http://.../dr-device/carlo/carlo-rbase.ruleml#"> ]>
<rulebase xmlns:carlo_rb="&carlo_rb;" xmlns:carlo="&carlo;" rdf_import="&carlo;"
           rdf_export_classes="acceptable rent"  rdf_export="http://.../dr-device/carlo/export-carlo.rdf">
   <_rbaselab>
      <ind type="defeasible" href="&carlo_rb;">carlo-rules</ind>
   </_rbaselab>
   ...
   <imp>
      <_rlab ruleID="r2" ruletype="defeasiblerule" superior="r1"><ind href="&carlo_rb;r2">r2</ind>  </_rlab>
      <_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:bedrooms"> <_and>  <var>y</var>
                                                            <function_call> <fname>&lt;</fname>
                                                                            <var>y</var>
                                                                            <ind>2</ind>
                                                            </function_call>
                                                    </_and>
                      </_slot>
              </atom>
      </_body>
   </imp>
   ...
</rulebase>
```

Fig. 3. Sample defeasible DR-DEVICE rule in RuleML-like syntax.

Classes and objects (facts) can also be declared in DR-DEVICE; however, the focus in this demo is the use of RDF data as facts. The input RDF file(s) are declared in the `rdf_import` attribute of the `rulebase` (root) element of the RuleML document. There exist two more attributes in the `rulebase` element: the `rdf_export` attribute that declares the address of the RDF file with the results of the rule program to be exported, and the `rdf_export_classes` attribute that declares the derived classes whose instances will be exported in RDF/XML format. Further extensions to the RuleML syntax, include function calls that are used either as constraints in the rule body or as new value calculators at the rule head. Furthermore, multiple constraints in the rule body can be expressed through the logical operators: `_not`, `_and`, `_or`.

The translation of defeasible rules into R-DEVICE rules is based on the translation of defeasible theories into logic programs through a meta-program ([2]). We use the meta-program to guide defeasible rule compilation. Each *defeasible rule* in DR-DEVICE is translated into a set of 5 R-DEVICE rules. Correct order of execution is guaranteed by predefined ordering among different R-DEVICE rule types and by stratification. For non-stratified programs the correct result is guaranteed through "truth maintenance" rules that undo (retract) the conclusions of rules when their condition is no longer met. In this way, even if rules are not executed in the correct order, the correct result will be eventually deduced because conclusions of rules that should have not been executed can be later undone.

5. A Brokered Trade Use Case

The demonstrator use case employs DR-DEVICE rules 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 (which has been adopted from [3]) is apartment renting and the landlord takes the role of the abstract seller.

Available apartments reside in an RDF document (Fig. 4). The requirements of a potential renter are expressed in DR-DEVICE's defeasible logic rule language. For example, rule `r2` (Fig. 2, Fig. 3) represents the requirement that an acceptable apartment must have at least two bedrooms. After the RuleML document in is loaded into DR-DEVICE, it is transformed into the native CLIPS-like DR-DEVICE syntax. DR-DEVICE rules are further translated into R-DEVICE rules, which in turn are translated into CLIPS production rules. All compiled rule formats are kept into local files, so that the next time they are needed they can be directly loaded, increasing speed. After rule compilation/loading the RDF document(s) is(are) loaded and transformed into CLIPS (COOL) objects. Finally, the reasoning can begin, which ends up with 3 acceptable apartments and one suggested apartment for renting, according to Carlo's requirements and the available apartments [3]. The results (i.e. objects of derived classes) are exported in an RDF file (Fig. 5) according to the specifications posed in the RuleML document (Fig. 3).

```
<!DOCTYPE rdf:RDF [   ...   <!ENTITY carlo "http://.../dr-device/carlo/carlo.rdf#"> ]>
<rdf:RDF   ...   xmlns:carlo="&carlo;">
   <carlo:apartment rdf:about="&carlo;a1">
      <carlo:name>a1</carlo:name>
      <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:pets>yes</carlo:pets>
      <carlo:price rdf:datatype="&xsd;integer">300</carlo:price>
      <carlo:size rdf:datatype="&xsd;integer">50</carlo:size>
   </carlo:apartment>
   ...
</rdf:RDF>
```

Fig. 4. RDF document for available apartments

```
<!DOCTYPE rdf:RDF [ ... <!ENTITY dr-device "http://.../dr-device/export/export-carlo.rdf#"> ]>
<rdf:RDF   ...          xmlns:dr-device='&dr-device;'>
...
   <dr-device:acceptable rdf:about="&dr-device;acceptable2">
      <dr-device:apartment>a2</dr-device:apartment>
      <dr-device:truthStatus>defeasibly-not-proven</dr-device:truthStatus>
   </dr-device:acceptable>
...
   <dr-device:acceptable rdf:about="&dr-device;acceptable5">
      <dr-device:apartment>a5</dr-device:apartment>
      <dr-device:truthStatus>defeasibly-proven</dr-device:truthStatus>
   </dr-device:acceptable>
   <dr-device:rent rdf:about="&dr-device;rent1">
      <dr-device:apartment>a5</dr-device:apartment>
      <dr-device:truthStatus>defeasibly-proven</dr-device:truthStatus>
   </dr-device:rent>
</rdf:RDF>
```

Fig. 5. Results of defeasible reasoning exported as an RDF document

Table 1 shows the time measured for executing the demo on a Pentium 4 PC with WinXP Professional (SP2) and 512MB main memory. Times exclude network latencies, i.e. all files are stored locally. Each line shows the time for each task that DR-DEVICE performs. The last line excludes from the total time the execution time of external programs, i.e. the Xalan XSLT processor and the ARP2 RDF parser. When rules are already compiled the time needed to perform the reasoning is 5 times faster (including external programs) or 9 times faster (excluding external programs). In this demo we have included 7 apartments and a total of 15 defeasible rules.

Table 1. Performance measurement for the demo

Task	**Execution Time (sec)**	
	Un-compiled Rules	**Compiled Rules**
Loading DR-DEVICE system	0.769231	0.769231
Translating RuleML syntax to DR-DEVICE native syntax (Xalan)	0.934066	-
Translating DR-DEVICE rules to R-DEVICE rules	0.604396	-
Parsing RDF files (ARP2)	1.868132	1.868132
Loading RDF into CLIPS	0.219780	0.219780
Loading R-DEVICE rules	6.648352	0.329670
Translating R-DEVICE rules	5.659341	-
Running R-DEVICE rules	0.274725	0.274725
Extracting results	~0	~0
Total Execution time	**16.978023**	**3.461538**
Total DR-DEVICE Execution time	**14.175825**	**1.593406**

6. Conclusions and Future Work

In this demonstration we presented a prototype system for defeasible reasoning on the Semantic Web. Defeasible reasoning is very useful for resolving conflicts among rules on the Semantic Web. The DR-DEVICE system is based on CLIPS production rules, and supports RuleML syntax. The system is freely available for downloading and experimentation at the following address: `http://lpis.csd.auth.gr/systems/dr-device.html`.

Planned future work includes:

- Developing a visual editor for the RuleML-like rule language.
- Deploying the reasoning system as a Web Service.
- Using appropriate constraint solvers in conjunction with logic programs.
- Implementing load/upload functionality in conjunction with an RDF repository.

- Study in more detail integration of defeasible reasoning with description logic based ontologies. Starting point of this investigation will be the Horn definable part of OWL [7].
- Applications of defeasible reasoning and the developed implementation for brokering, bargaining, automated agent negotiation, and personalization.

7. References

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

[2] Antoniou G., Billington D., Governatori G., Maher M.J, "A Flexible Framework for Defeasible Logics", Proc. AAAI/IAAI 2000, AAAI/MIT Press, pp. 405-410.

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

[4] Bassiliades N., Antoniou G., Vlahavas I., "A Defeasible Logic Reasoner for the Semantic Web", *Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, Springer-Verlag, Hiroshima, Japan, 8 Nov. 2004.

[5] Bassiliades N., Vlahavas I., "R-DEVICE: A Deductive RDF Rule Language", *Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, Springer-Verlag, Hiroshima, Japan, 8 Nov. 2004.

[6] Bassiliades N., Vlahavas I., Elmagarmid A.K., "E-DEVICE: An extensible active knowledge base system with multiple rule type support", *IEEE TKDE*, 12(5), pp. 824-844, 2000.

[7] 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.

[8] Seaborne A., Reggiori A., "RDF Query and Rule languages Use Cases and Examples survey", `rdfstore.sourceforge.net/2002/06/24/rdf-query/`

[9] `http://www.ruleml.org`

[10] `http://www.hpl.hp.com/personal/jjc/arp/`

[11] `http://xml.apache.org/xalan-j/`