DR-DEVICE[1]: A Defeasible Logic Reasoner for the Semantic Web

User Guide and Installation Manual

Nick Bassiliades[*], Grigoris Antoniou[**], and Ioannis Vlahavas[*]

[*]**Dept. of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece**

{nbassili|vlahavas}@csd.auth.gr

[**]**Institute of Computer Science, FO.R.T.H., P.O. Box 1385, GR-71110, Heraklion, Greece**

antoniou@ics.forth.gr

Table of Contents

INTRODUCTION **1**

DR-DEVICE SYSTEM ARCHITECTURE **2**

THE RULE LANGUAGE OF DR-DEVICE **3**

INSTALLATION INSTRUCTIONS **4**

USER GUIDE **4**

RUNNING A REMOTE DR-DEVICE RULEML FILE 4
RUNNING A LOCAL DR-DEVICE RULEML FILE 6
RUNNING THE DEMONSTRATION (BROKERED TRADE) EXAMPLE 6
USEFUL DR-DEVICE FUNCTIONS 6

DR-DEVICE RULE SYNTAX **7**

REFERENCES **8**

ACKNOWLEDGMENTS **8**

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 pre-sent a prototype system for defeasible reasoning on the Web. The system is called DR-DEVICE ([1], [2], [3]) and is capable of reasoning about RDF metadata over multiple Web sources using defeasible logic rules. The system is implemented on top of CLIPS production rule system and builds upon R-DEVICE ([4], [2]), 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[2] 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.

[1] http://iskp.csd.auth.gr/systems/dr-device.html

[2] http://www.ruleml.org/

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.

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[3] 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 trans-formed 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.

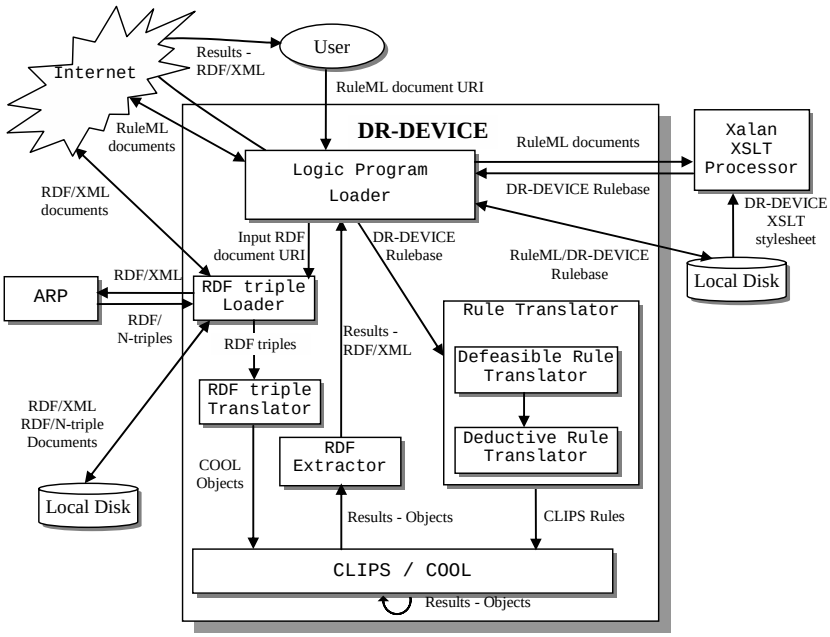


Fig. 1. Architecture of the DR-DEVICE system.

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[4] 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

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

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

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.

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.

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`.

```
<!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.

[5] `http://www.ghg.net/clips/CLIPS.html`

The translation of defeasible rules into R-DEVICE rules is based on the translation of defeasible theories into logic programs through a meta-program. 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.

Installation Instructions

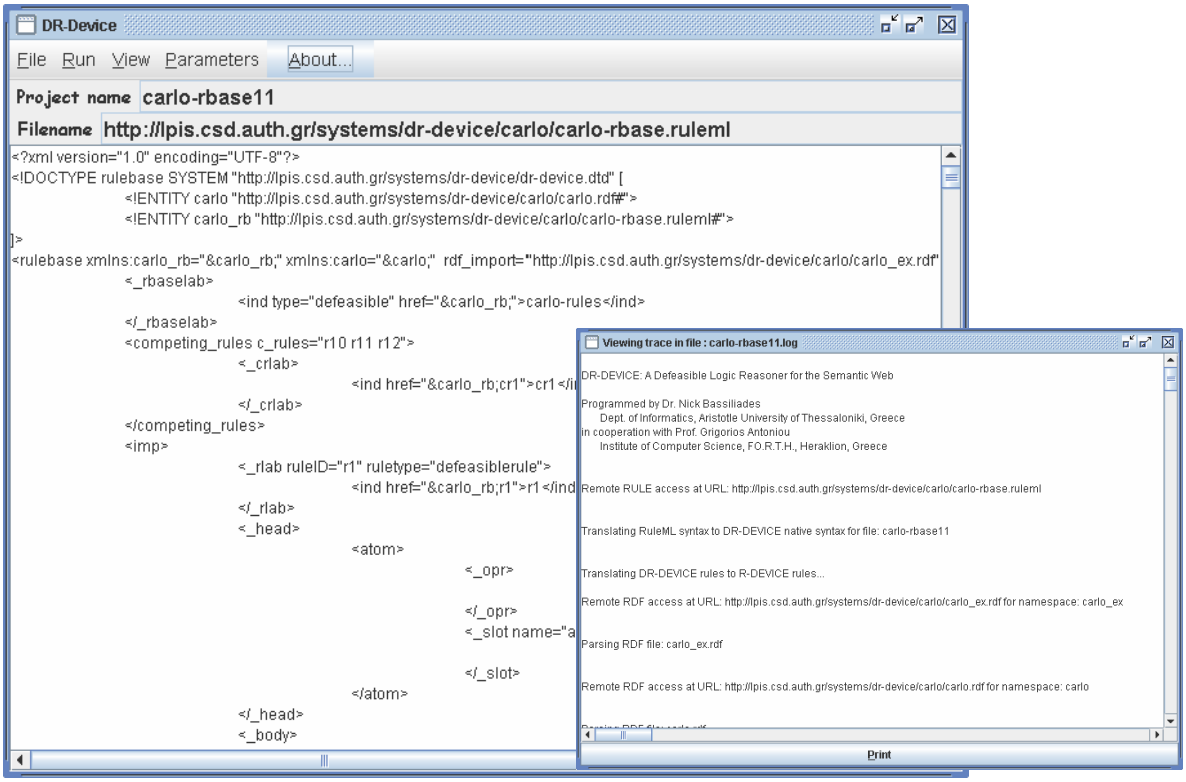
- Download `DR-Device setup.exe` from `http://iskp.csd.auth.gr/systems/dr-device.html`
- Run the setup program and follow the instructions
- Reboot after installation ends.

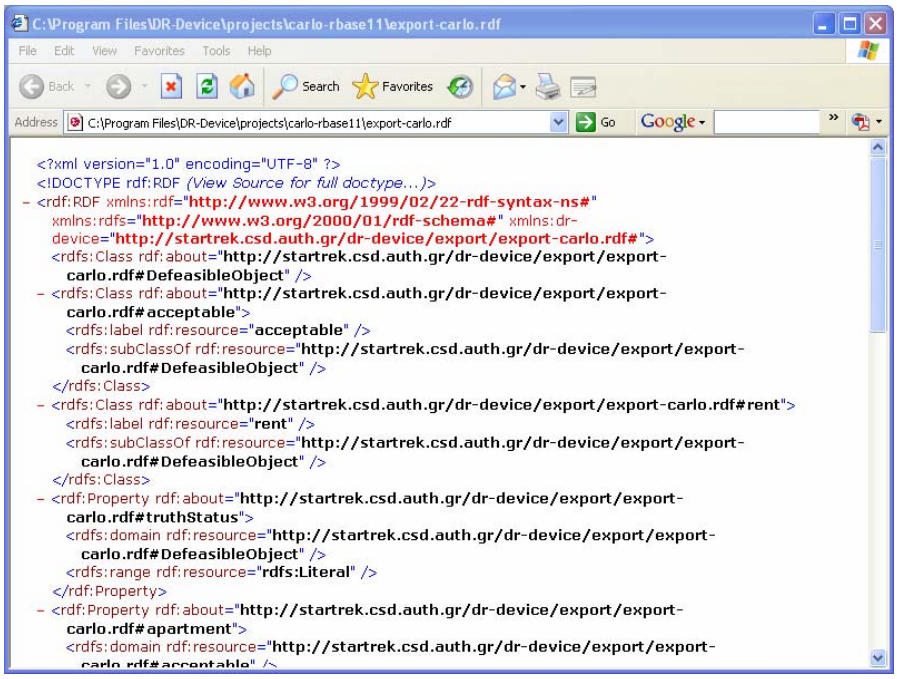
User Guide

Running a remote DR-Device RuleML file

- From the *File...* menu, choose *New Project*.
- Click on the *URL* choice at the radio button.
- Fill-in the *Project Name* and *Filename* fields:
  - *Filename* is the URL address of a DR-Device RuleML file.
  - *Project Name* is the corresponding local filename that will be used to address to this file. A new directory *Project Name* is generated for each new project, under the *projects* directory inside the Dr-Device installation directory. All files related to this project are stored inside this directory.
- Click on the *OK* button and wait until the corresponding `ruleml` file is downloaded from the specified address. The file can be edited and saved locally using the *File -> Save As...* menu item.
- Choose the *Run -> Compile & Run* menu item. Watch the execution trace on the DOS window that appears. After the demo runs successfully press any key to close the DOS Window.
  - The *Parameters* menu contains some parameters that can modify the detail of the trace. When the *Verbose* parameter is "on", DR-Device reports on the major steps that it performs, such as:
    - ✓ Accessing Remote rule and RDF files
    - ✓ Translating rules between different rule formats (RuleML, DR-DEVICE, R-DEVICE, CLIPS)
    - ✓ Loading and translating RDF files
    - ✓ Running rules
    - ✓ Extracting results
  - When the *Debug* parameter is "on" DR-Device reports on many minor steps that it performs, such as:
    - ✓ Loading RDF triples
    - ✓ Details on translating RDF triples into COOL objects (class/object creation, schema re-configuration, RDF Semantics entailment rules, etc.)
    - ✓ Details on translating high-level rules (DR-DEVICE, R-DEVICE) into low-level CLIPS production rules.
  - When the *Time-report* parameter is "on" DR-Device reports on how much time some steps take (mainly concerning loading & translating RDF triples).

- Choose the *View -> View Results* menu item to view the result of the defeasible reasoning process that the loaded DR-Device RuleML file performed. Results are presented as an RDF/XML file on a separate Internet Explorer window.
- Choose the *View -> View Trace* menu item to (re-)view (on a separate text window) the trace of execution that appeared earlier in the DOS window.
- All compiled rule formats are kept into local files, so that the next time they are needed they can be directly loaded and run, using the *Run -> Run Compiled* menu item, increasing, thus, speed.
- Choose *File -> Exit* to quit, closing both the Java GUI window and the background DOS window. If you close first the background DOS window, then the Java GUI window closes as well.





Running a local DR-Device RuleML file

Follow the same procedure as above, except for the following:

- Click on the *Local* choice at the radio button.
- Click on the *Browse...* button and browse the local file system to open a local DR-Device RuleML file. The file can be edited and saved locally using the *Save As...* button.
- Fill-in the *Project Name* field. *Project Name* is the filename that will be used by Dr-Device to address to this local file. A new directory *Project Name* is generated for each new project, under the *projects* directory inside the Dr-Device installation directory. All files related to this project are stored inside this directory. The local RuleML file is copied to this directory.

Running the demonstration (Brokered Trade) example[6]

- Click on the *DEMO* button. The *Project Name* and *Filename* fields are filled in with a name for the Brokered Trade example and the URL address (`http://iskp.csd.auth.gr/systems/dr-device/carlo/carlo-rbase.ruleml`) of the `ruleml` file, correspondingly.
- Perform the rest of the steps as above.

Running a Saved Project

Choose *File -> Open Project* and select a project folder under the *projects* subdirectory of the Dr-Device installation directory. Then either choose *Compile & Run* or *Run Compiled*.

Useful DR-Device Functions

In this section we briefly outline a number of useful DR-Device functions:

- `(load-ruleml-defeasible <filename> <address>)`

  Load, translate, and run a remote DR-Device RuleML file from URL `<address>` and save it locally under the name `<filename>`.

- `(load-ruleml-defeasible-local <filename> <path-file>)`

  Load, translate, and run a local DR-Device RuleML file from path & filename denoted by `<path-file>`. Copy the file in the current directory under the name `<filename>`.

- `(load-defeasible <filename>)`

  Load, translate, and run the local DR-Device rule file `<filename>` in native CLIPS-like syntax.

- `(load-only-defeasible <filename>)`

  Load and translate (but not run) the local DR-Device rule file `<filename>` in native CLIPS-like syntax.

- `(load-compiled-defeasible <filename>)`

  Load (but not run) the local DR-Device rule file `<filename>` in native CLIPS-like syntax, that has been already been compiled with one of the previous commands.

- `(go-defeasible)`

  Run all the DR-Device rules that have been already loaded with one of the previous commands.

- `(defeasibly_export_rdf <export-rdf-file> <export-rdf-classes>)`

  Export at file `<export-rdf-file>` the schema and instances of the defeasible classes `<export-rdf-classes>` in RDF/XML format. The exported instances are supposed to represent the results of

[6] `http://iskp.csd.auth.gr/systems/dr-device/carlo/carlo.html`

the inference process. Usually, `<export-rdf-file>` is indicated in the `rdf_export` attribute of the root `rulebase` element of the DR-Device RuleML file, and `<export-rdf-classes>` are indicated in the `rdf_export_classes` attribute of the same element (see Apendix).

- `(import-rdf-files <URL-or-files>)`

  Import (load and transform to objects) all the RDF files `<URL-or-files>` that are identified either by local names or URLs. The import of each file is performed in a single step (no streaming). This can be slow for large RDF files. Usually the RDF files to be imported are indicated in the `rdf_import` attribute of the root `rulebase` element of the DR-Device RuleML file (see Apendix).

- `(import-rdf <URL-and-files>)`

  Import (load and transform to objects) all the RDF files `<URL-and-files>` that are identified by pairs `(filename address)` where filename is the local filename under which the remote RDF file will be saved and address is the URL address of the remote RDF file. The import of each file is performed in a single step (no streaming). This can be slow for large RDF files.

- `(inc-import-rdf <limit> <hash-buckets> <URL-and-files>)`

  Import (load and transform to objects) all the RDF files `<URL-and-files>` that are identified by pairs `(filename address)` where filename is the local filename under which the remote RDF file will be saved and address is the URL address of the remote RDF file. The import of each file is performed in a streaming fashion. `<limit>` is an integer indicating how many RDF triples are translated at each step. If the `def` value is given, the default value 10000 is assumed. `<hash-buckets>` is an integer indicating the number of buckets that are internally used for keeping track of useful information when loading the RDF triples. If the `def` value is given, the default value 100 is assumed.

DR-DEVICE Rule Syntax

```
<!ELEMENT rulebase (((_rbaselab, (imp | competing_rules)*) |
                    ((imp | competing_rules)+, _rbaselab?))?)>
<!ATTLIST rulebase
   xmlns CDATA #IMPLIED
   xsi:schemaLocation CDATA #IMPLIED
   xmlns:xsi CDATA #IMPLIED
   <!-- The URLs of the RDF files to load -->
   rdf_import CDATA #IMPLIED
   <!-- The name of the file that the results of the reasoning process will be written to -->
   rdf_export_classes NMTOKENS #IMPLIED
   <!-- The names of the classes whose instances will constitute the results of the inference process -->
   rdf_export CDATA #IMPLIED
>
<!ELEMENT _rbaselab (ind)>
<!ELEMENT imp ((_rlab, ((_head, _body) | (_body, _head))) |
               (_head, ((_rlab, _body) | (_body, _rlab?))) |
               (_body, ((_rlab, _head) | (_head, _rlab?))))>
<!ELEMENT competing_rules (_crlab, _slots?)>
<!ATTLIST competing_rules
   c_rules IDREFS #REQUIRED
>
<!ELEMENT _slots (slotname+)>
<!ELEMENT slotname (#PCDATA)>
<!ELEMENT _rlab (ind)>
<!ATTLIST _rlab
   ruleID ID #REQUIRED
   ruletype (strictrule | defeasiblerule | defeater) #REQUIRED
   superior IDREFS #IMPLIED
>
<!ELEMENT _crlab (ind)>
<!ELEMENT _head (calc?, (atom | neg))>
<!ELEMENT _body (atom | neg | naf | and | or)>
<!ELEMENT atom ((_opr, (_slot)*, ((ind | var)+, (_slot)*)?) |
     ((((_slot)+, ((ind | var)+, (_slot)*)?) | ((ind | var)+, (_slot)*)), _opr))>
<!ELEMENT and ((atom | neg | naf | or)*)>
```

```
<!ELEMENT or ((atom | neg | naf | and)*)>
<!ELEMENT neg (atom)>
<!ELEMENT naf (atom | and)>
<!ELEMENT _opr (rel)>
<!ELEMENT rel (#PCDATA)>
<!ATTLIST rel
   href CDATA #IMPLIED
>
<!ELEMENT _slot (ind | var | _not | _or | _and)>
<!ATTLIST _slot
   name CDATA #REQUIRED
   card CDATA #IMPLIED
   weight CDATA #IMPLIED
>
<!ELEMENT ind (#PCDATA)>
<!ATTLIST ind
   type CDATA #IMPLIED
   href CDATA #IMPLIED
>
<!ELEMENT var (#PCDATA)>
<!ATTLIST var
   type CDATA #IMPLIED
>
<!ELEMENT calc (function_call+)>
<!ELEMENT function_call (fname, (ind | var | function_call)*)>
<!ELEMENT fname (#PCDATA)>
<!ENTITY % term "(_not | ind | var | function_call)">
<!ELEMENT _not (ind | var)>
<!ELEMENT _or (%term;, (%term;)+)>
<!ELEMENT _and (%term;, (%term;)+)>
```

Acknowledgments

The graphical user interface for DR-Device has been developed by Stratos Kontopoulos. Thomas Skylogiannis has helped in evaluating DR-Device, by using it as a back-end reasoning system for price negotiating agents [6].

References

[1] N. Bassiliades, G. Antoniou, I. Vlahavas, “DR-DEVICE: A Defeasible Logic System for the Semantic Web”, *Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR04)*, Sebastian Schaffert (Ed.), Springer-Verlag, LNCS 3208, pp. 134-148,St Malo, France, Sept. 2004.

[2] N. Bassiliades, G. Antoniou, I. Vlahavas, “A Defeasible Logic Reasoner for the Semantic Web”, *Third International Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, G. Antoniou, H. Boley (Ed.), Springer-Verlag, LNCS 3323, pp. 49-64, Hiroshima, Japan, 8 Nov. 2004.

[3] N. Bassiliades, G. Antoniou, I. Vlahavas, “DR-DEVICE: A Defeasible Logic RDF Rule Language", Demo, *Demonstration at 3rd International Semantic Web Conference (ISWC2004)*, 7-11 November 2004, Hiroshima, Japan (available at: `http://iswc2004.semanticweb.org/demos/index.html`).

[4] N. Bassiliades, I. Vlahavas, "Capturing RDF Descriptive Semantics in an Object Oriented Knowledge Base System", *Electronic Poster Proc. 12th Int. WWW Conf. (WWW2003)*, 20-24 May 2003, Budapest, Hungary.

[5] N. Bassiliades, I. Vlahavas, “R-DEVICE: A Deductive RDF Rule Language”, *Third International Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, G. Antoniou, H. Boley (Ed.), Springer-Verlag, LNCS 3323, pp. 65-80, Hiroshima, Japan, Nov. 2004.

[6] T. Skylogiannis, G. Antoniou, N. Bassiliades, G. Governatori, “DR-NEGOTIATE – A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies”, *IEEE International Conference on E-Technology, E-Commerce and E-Service*, IEEE, pp. 44-49, Hong Kong, China, 2005.