R-DEVICE: A Deductive RDF Rule Language

Nick Bassiliades, Ioannis Vlahavas

Dept. of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{nbassili,vlahavas}@csd.auth.gr

Abstract. In this paper we present R-DEVICE, a deductive rule language for reasoning about RDF metadata. 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. 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. 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.

1 Introduction

Semantic Web is the next step of evolution for the Web [7], where information is given well-defined meaning, enabling computers and people to work in better cooperation. Currently information found on the Web is mainly built around visualization, and is not machine-understandable. It is difficult to automate things on the Web, and because of the volume of information the Web contains, it is even more difficult to manage it manually. The solution that has been proposed by the WWW Consortium was to use metadata to describe the data contained on the Web. The Resource Description Framework (RDF) is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web [23].

Conveying the content of documents is just a first step for achieving the full potential of the Semantic Web. Additionally, it is mandatory to be able to reason with and about information spread across the WWW. Rules provide the natural and wide-accepted mechanism to perform automated reasoning, with mature and available theory and technology. This has been identified as a Design Issue for the Semantic Web, as clearly stated in [7].

Rules and rule markup languages, such as RuleML [8], will play an important role for the success of the Semantic Web. Rules will act as a means to draw inferences, to

express constraints, to specify policies, to react to events/changes, to transform data, etc. Rule markup languages will allow to enrich web ontologies by adding definitions of derived concepts, to publish rules on the web, to exchange rules between different systems and tools, etc. The applications range from electronic commerce applications, data integration and sharing, information gathering, security access and control, law, diagnosis, B2B, and of course, to modeling of business rules and processes.

It seems natural to add rules "on top" of web ontologies. However, as it is argued in [2], putting rules and description logics together poses many problems, and may be an overkill, both computationally and linguistically. Another possibility is to start with RDF/RDFS, and extend it by adding rules.

In this paper we present R-DEVICE, a deductive rule language for reasoning about RDF metadata. R-DEVICE is able to draw inferences both on the RDF schema and data. R-DEVICE includes features such as ground and generalized path expressions, stratified negation, aggregate, grouping, and sorting, functions. All these can be combined with a second-order syntax, where variables can range over classes and properties. Such variables are grounded at compile-time using metadata so second-order rules are safely and efficiently translated into sets of first-order rules. Furthermore, users can define views with R-DEVICE rules which are materialized and, optionally, incrementally maintained by translating deductive rules into CLIPS production rules [10]. Users can use built-in functions of CLIPS or can define their own arbitrary functions. The syntax of R-DEVICE rules follows the OPS5/CLIPS paradigm. Furthermore, an XML syntax is provided that extends RuleML [8] and especially the version that supports OO features and negation-as-failure.

R-DEVICE employs a novel OO-RDF data model [4] that maps RDF documents into COOL objects inside the CLIPS production rule system. The main difference between the RDF graph model and our data model is that we treat properties mainly as attributes encapsulated inside resource objects, as in traditional OO programming languages. In this way properties about a single resource are gathered together in one object, resulting in superior inference/query performance than the performance of a triple-based model, as it has been experimentally shown elsewhere [5]. Most other RDF inferencing/querying systems that are based on a triple model scatter resource properties across several triples and they require several joins to access the properties of a single resource. The descriptive semantics of RDF data may call for dynamic redefinitions of resource classes and objects, which are handled by R-DEVICE.

In the rest of this paper we briefly review related work in RDF rule languages in section 2; Section 3 presents the architecture of the R-DEVICE system, including the OO RDF model of R-DEVICE; Section 4 describes the R-DEVICE rule language, including its RuleML syntax and how inference results are exported as RDF documents. Finally, section 5 concludes this paper and discusses future work.

2 Related Work

Many RDF rule languages exist in the literature. Some on-line surveys of RDF Inference and Query systems can be found in [24] and [22]. In this section, we will refer to some of the most representative ones and we will compare them to R-DEVICE.

TRIPLE [25], an extension of the SiLRI system [11], is an RDF rule (query, inference, and transformation) language, with a layered and modular nature, that is based on Horn Logic and F-Logic and aims to support applications in need of RDF reasoning and transformation, i.e., to provide mechanisms to query web resources in a declarative way. However, in contrast with many other RDF rule/query languages, TRIPLE allows the semantics of languages on top of RDF to be defined with rules, instead of supporting the same functionality with built-in semantics. Wherever the definition of language semantics is not easily possible with rules (e.g., OWL [26]), TRIPLE provides access to external programs, like description logic classifiers.

TRIPLE permits the usage of path expressions, but not generalized path expressions, i.e. the path length and composition must be entirely known. Furthermore, compared to R-DEVICE, TRIPLE does not support aggregate, grouping, sorting and user-defined functions. Rules in TRIPLE are used for transient querying and cannot be used for defining and maintaining views. As its name implies, the query and data model of TRIPLE is triples, therefore TRIPLE requires multiple joins for collecting all the properties of a resource, since property instances and resource instances are stored in different database relations (or in different tuples of the same relation). In [5] we have shown that the OO-RDF data model of R-DEVICE is superior in performance compared to the triple-based data model of most RDF query and rule languages. Finally, TRIPLE does not have a RuleML compatible syntax.

SweetJess [16] is an implementation of a defeasible reasoning system (situated courteous logic programs) based on Jess. R-DEVICE is a deductive rule language that supports non-monotonicity in terms of negation-as-failure. Furthermore, recently we have developed a defeasible logic extension to R-DEVICE [3]. SweetJess integrates well with RuleML, as R-DEVICE. However, SweetJess rules can only express reasoning over ontologies expressed in DAMLRuleML (a DAML-OIL like syntax of RuleML) and not on arbitrary RDF data, like R-DEVICE. Furthermore, SweetJess is restricted to simple terms (variables and atoms). R-DEVICE can support a limited form of functions in the following sense: (a) path expressions are allowed in the rule condition, which can be seen as complex functions, where allowed function names are object referencing slots; (b) aggregate and sorting functions are allowed in the conclusion of aggregate rules. Finally, R-DEVICE can also support conclusions in non-stratified rule programs due to the presence of truth-maintenance rules.

SWRL [18] is a rule language based on a combination of the OWL DL and Lite sublanguages of OWL [26] with the Unary/Binary Datalog sublanguages of RuleML [8]. SWRL enables Horn-like rules to be combined with an OWL knowledge base. SWRL provides several types of syntaxes, including RuleML and RDF-like. SWRL is also based on the triple model of RDF and is a first-order, negation free logic language specification with no concrete implementation. Its main purpose is to provide a formal meaning of OWL ontologies and extend OWL DL.

The Edutella project [21] provides a family of Datalog like languages, called RDF-QEL-i, that support different levels of query capabilities among distributed, heterogeneous RDF repositories. The highest level language RDF-QEL-5 is equivalent to stratified Datalog. Furthermore, aggregation and foreign functions are supported. Actually, the RDF-QEL-i languages provide a common query and inference syntax and semantics for the heterogeneous peers and are translated into the base rule/query language of each peer. Several query language wrappers have been implemented, such

as RQL, TRIPLE, etc. The common data model of Edutella is based on triples and an RDF-like syntax is provided. Path expressions and view maintenance are not supported.

CWM [6] is a general-purpose data processor for the semantic web. It is a forward chaining reasoner which can be used for querying, checking, transforming and filtering information. Its core language is RDF, extended to include rules, and it uses RDF/XML or RDF/N3. CWM supports path expressions, like TRIPLE, but only concrete ones, i.e. the path length should be known in advance and every step in the path should be ground. Furthermore, CWM does not support negation. CWM allows aggregated functions but not grouping and sorting.

Jena [20] is based on the RDF triple data model has an inference subsystem that allows a range of inference engines or reasoners to be plugged into Jena. The inference mechanism is designed to be quite general and it includes a generic rule engine that can be used for many RDF processing or transformation tasks. The generic rule reasoner supports user defined rules under forward chaining, tabled backward chaining and hybrid execution strategies. The rule language allows a limited form of functors, but does not support either negation, aggregation or path expressions. Jena rules do not have a RuleML-like syntax, but the extensibility of the system allows for different syntaxes. Finally, the Jena rule system allows maintenance of asserted conclusions, which however is trivial due to the lack of negation.

ROWL [12] is a system that enables users to express rules in RDF/XML syntax using an ontology in OWL. Using XSLT stylesheets, the rules in RDF/XML are transformed into forward-chaining production rules in JESS. ROWL also uses stylesheets to transform ontology and instance files into Jess unordered triple facts, which is also the model followed by the rules. ROWL does not maintain the assertions derived by the rules and does not support either negation, path expressions or aggregate functions. The rule language has been used in a Semantic Web environment for pervasive computing where agents reason about context and privacy concerns of the user [13].

Bossam [19] is a RETE-based forward-chaining production rule engine that has an RDF logic-like rule language, called Buchingae. Bossam has an RDF-like knowledge representation scheme and supports both strong and weak negation and second-order typed predicates. The Bossam data model is based on triples, therefore second order syntax is actually translated into first-order querying over property and/or type definition triples. Negation is supported by the rule language; however, no hint on how it is implemented by the rule engine is given. Bossam also provides a RuleML-like language, called LogicML, which however overrides several of the RuleML elements, hindering compatibility with standard RuleML. Finally, inference results exported by Bossam are flat, i.e. there is no notion of derived classes and properties.

3 R-DEVICE Architecture

The R-DEVICE system consists of two major components (**Fig. 1**): the RDF loader/translator and the rule loader/translator. The former accepts from the user requests for loading specific RDF documents. The RDF triple loader downloads the RDF document from the Internet and uses the ARP parser [20] to translate it to triples

in the N-triple format. Both the RDF/XML and RDF/N-triple files are stored locally for future reference.

The RDF document is scanned for namespaces that have not already been imported/translated into the system. Some of the untranslated namespaces may already exist on the local disk, while others are fetched from the Internet. All namespaces (both fetched and locally existing) are recursively scanned for namespaces, which are also fetched if not locally stored. Finally, all untranslated namespaces are also parsed using the ARP parser. The reason for doing this is explained below.

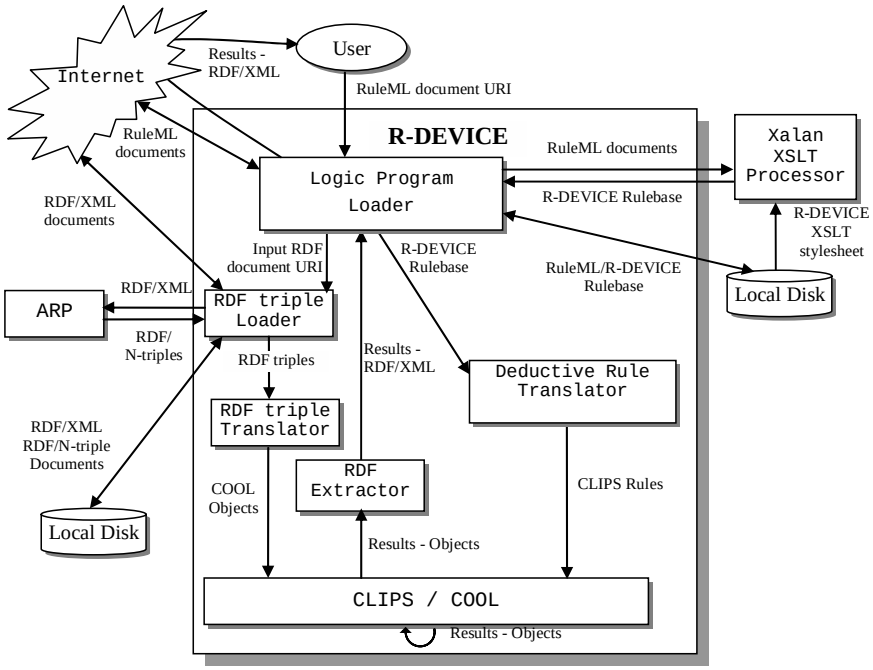

Fig. 1. Architecture of the R-DEVICE system.

RDF documents usually refer to existing RDF Schema documents through the namespace mechanism. Although the semantics of an RDF document is precisely defined by [17], the knowledge of the RDF Schema that an RDF document follows provides better understanding of its content both for the human and the machine (reasoning engine). Furthermore, the knowledge of the schema allows for more knowledgeable (and thus more efficient) rules/queries over the contents of the RDF document.

The rationale for recursively translating all namespaces is to minimize the number of OO schema redefinitions. Fetching multiple RDF schema files will aggregate multiple RDF-to-OO schema translations into a single OO schema redefinition. Namespace resolution is not guaranteed to yield an RDF schema document; therefore, if the namespace URL is not an RDF document, then the ARP parser will not produce tri-

ples and R-DEVICE will make assumptions, based on the RDF semantics, about non-resolved properties, resources, classes, etc.

Notice that the scheme we use is "nondeterministic", because if a resource is temporarily unavailable, then when loading an RDF document its namespace (i.e. RDF Schema) will not be fetched and the RDF descriptions will be translated differently than if the schema were available. However, we find that this "nondeterminism" is compatible with the unstable nature of the Web.

If the automatic namespace/schema handling is not desired by the user, the feature can be simply turned off. Users can then manually and explicitly import RDF Schema documents before the loading of the actual RDF instance documents. This explicit import of a schema is not available in RDF/S, but the issue is resolved in OWL [26], where explicit import of remote ontologies is supported.

All N-triples are loaded into memory, while the resources that have a `URI#anchorID` format are transformed into a `namespace:anchorID` format if `URI` belongs to the initially collected namespaces, in order to save memory space. The transformed RDF triples are fed to the RDF triple translator which maps them into COOL objects, according to the mapping schema that is briefly sketched in the following subsection. Notice that when an RDF triple is consumed it is deleted.

The rule loader accepts from the user a URI (or a local file name) that contains a deductive rule program in RuleML notation [8]. 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 R-DEVICE rule notation using the Xalan XSLT processor [27] and an XSLT stylesheet. The R-DEVICE rule program is then forwarded to the rule translator.

The rule translator accepts from the user a set of R-DEVICE rules and translates them into a set of CLIPS production rules. When the translation ends, CLIPS runs the production rules and generates the objects that constitute the result of the initial rule program. Finally, the result-objects are exported to the user as an RDF/XML document through the RDF extractor. An example of the results produced by R-DEVICE inferencing can be found in **Fig. 4**.

3.1 The Object-Oriented RDF Model

In this subsection we briefly describe how the RDF data model is mapped onto the COOL object-oriented model of the CLIPS language. More details can be found at [4], [5]. 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. This binary representation is due to the fact that COOL does not support meta-classes. Class names follow the `namespace:anchorID` format, while their corresponding instances have an object identifier with the same name, surrounded by square brackets.

All *resources* are represented as COOL objects, direct or indirect instances of the `rdfs:Resource` class. The identifier of a resource object is either in `namespace:anchorID` format, if its URI can be resolved in this way, or its complete address otherwise.

The specific class of each resource object depends on the `rdf:type` property of the resource. When a resource has multiple `rdf:type` properties then the resource object belongs to multiple classes. This cannot be handled directly in COOL (and in most OO programming languages), therefore a dummy class is generated which is a subclass of all the classes that the object should belong to. Then the resource object is made an instance of this class. The slot `source` indicates whether an object is a proper RDF resource or a system-generated object.

Properties are direct or indirect instances of the class `rdf:Property`. Furthermore, properties whose domain is a single class are defined as slots (attributes) of this class. The values of properties are stored inside resource objects as slot values. Actually, RDF properties are multislots, i.e. they store lists of values, because a resource can have multiple times the same property attached to it (with a different value, of course). When a property has multiple domains, then a dummy class is generated which is a subclass of all the classes of the property domain. The property is then made a slot of this dummy class, since resource objects that have this property must be instances of all the classes in the domain.

Properties with no domain constraint become slots of the `rdfs:Resource` class. However, this class is already defined, which means that it should be dynamically re-defined. This is a consequence of RDF descriptive semantics which may add new properties to already existing classes and is treated in R-DEVICE by re-defining classes at run-time. Dynamic class re-definition requires backing-up to a file all CLIPS constructs (functions, rules, instances, sub-classes) that depend on the re-defined class, un-defining these constructs (including the class), re-defining the class adding the new property(-ies), and re-inserting the backed-up constructs into the knowledge base. Similar dynamic re-definitions occur in several other occasions in R-DEVICE, because new triples can be added at any time. In general, R-DEVICE does not reject any RDF triple because every asserted triple is considered to be true. In [5] a performance comparison between having and not having dynamic class re-definition can be found, which shows that schema re-definition does not incur a very high overhead at the total triple import time.

The `rdfs:range` constraint of properties defines the type of the values of slots. When this constraint is absent, there is no type constraint for the slots. If the value of the constraint is `rdfs:Literal` then the corresponding slot is of type `STRING`. Some of the XML Schema data types have been mapped to COOL data types through the `rdfs:Datatype` class. Specifically, `xsd:integer`, `xsd:long`, etc. are casted to `INTEGER`, `xsd:float`, `xsd:decimal`, etc. are casted to `FLOAT`, while all other data types are treated as strings. Finally, when the range of a property is a class, the type of the slot is `INSTANCE`, i.e. slot values are OIDs of resource objects. When there are multiple range constraints, R-DEVICE creates a dummy class (similarly to the case of multiple domain constraints) which becomes the type of the slot.

The RDF triple translator is actually implemented as a CLIPS production rule program. Some production rules consume RDF triples and create COOL resource objects, filling up their slots with properties, while other rules examine these resource objects and enforce RDF model theory, i.e. they create COOL classes and they treat property hierarchies using an aliasing mechanism.

4 The Deductive Rule Language of R-DEVICE

The deductive rule language of R-DEVICE supports inferencing over RDF data represented as objects and defines materialized views over them. Views can be maintained incrementally or not, based on users' preferences. The conclusions of deductive rules represent derived classes, i.e. classes whose objects are generated by evaluating these rules over the current set of objects. Furthermore, the language supports recursion, stratified negation, path expressions over the objects, generalized path expressions (i.e. path expressions with an unknown number of intermediate steps), derived and aggregate attributes. Finally, users can call out to arbitrary built-in or user-defined functions of the implementation language, i.e. CLIPS.

Each deductive rule in R-DEVICE is implemented as a pair of CLIPS production rules: one for inserting a derived object when the condition of the deductive rule is satisfied and one for deleting the derived object when the condition is no more satisfied, due to base object deletions and/or slot modifications. The latter is only generated when the user prefers the derived class to be maintainable. R-DEVICE uses RETE algorithm to match production rule conditions against the objects. More details on the translation algorithm can be found in [5].

Below is an example of a deductive rule that finds the latest published articles on a Web site whose title contains the word 'RDQL'. We notice that most of the rule examples have been obtained from [24].

```
(deductiverule q5
  (rss:item (rss:title ?title&:(str-index "RDQL" ?title))
            (rss:link ?link))
=>
  (result (link ?link)))
```

We notice that the assertion of a derived object is based on a counter mechanism which counts how many derivations exist for a certain derived object, based on the values of its slots. The latter also define the identifier of the derived object. Derived objects are created only when not already exist, otherwise their counter is just increased by one. Furthermore, derived objects are deleted when their counter is one, otherwise their counter is decreased by one. Production rules that watch out for possible deletion of derived objects, have the negation of the original deductive rule condition in their condition. Finally, derived objects also keep the OIDs of their derivators, i.e. the base objects to which they owe their existence, and the name of the rule that derived them. This is needed to correctly maintain the derived view.

4.1 Rule Syntax

The syntax of R-DEVICE deductive rules is a variation of the syntax for CLIPS production rules [10] and can be found in [5]. Although R-DEVICE uses COOL objects, the syntax of rules is as if deductive rules query over CLIPS templates, because the syntax is simpler. Specifically, each condition element follows the following format:

```
?OID <- (classname (path-expr value-expr) …)
```

where `?OID` is the (optional) object identifier (or instance name, *not* address) of an object of class `classname`, and `(path-expression value-expression)` are zero, one, or more conditions to be tested on each object that matches this pattern.

When the name of the class is unknown, a variable can be used instead. Classnames can consist of a namespace prefix followed by a colon and a local part name. R-DEVICE allows the use of variables in both the place of the namespace prefix and the local part name. The following condition element applies to instances of classes of the `rss` namespace: `(rss:?c (rss:title ?t))`.

A value expression can be a constant or a variable or a constraint or a combination of those, as defined by CLIPS rule syntax. A path expression is an extension of CLIPS's single ground slot expression. Specifically, in R-DEVICE a path expression can be one of the following:

- A single slot of the class `classname`. The following single condition element accesses slots `rss:title` and `rss:link` of objects of class `rss:item`:
  ```
  (rss:item (rss:title ?t) (rss:link ?l))
  ```
- A single variable denoting *any* slot of class `classname`. The following condition element searches for a resource object with an unknown slot whose value is "Smith": `(rdfs:Resource (?s "Smith"))`.
- A ground path that consists of a list of multiple slots surrounded by brackets. The following rule shows an example of such a path:
  ```
  (deductiverule path-example
    (dmoz:Topic (dc:title "Arts") ((dc:title dmoz:link) ?t)
  =>
    (art-titles (title ?t)))
  ```

The right-most slot should be a slot of the "departing" class. Moving to the left, slots belong to classes that represent the range of the predecessor slots. The value expression in such a pattern (e.g. variable `?t`) actually describes a value of the left-most slot of the path.

- A path that contains one or more single-field variables, i.e. a path whose length is known but some of the steps are not. The above ground path can be turned into such a path: `((dc:title ?x) ?t)`.
- A generalized path that contains one or more multi-field variables, i.e. variables that their value is a list. These non-ground paths have an unknown length. The path below can have at least zero steps and at most 1 (given the specific example): `((dc:title $?p) ?t)`.
- A path that contains an encapsulated recursive sub-path, i.e. a sub-path that is traversed an unknown number of times. The following path contains the recursive sub-path `(dcq:references)` which recursively follows resources that reference each other: `((dc:title (dcq:references)) ?t)`.

Recursive paths can be used to express transitive closure queries. The following rule collects all resources (pages) recursively referenced by a certain resource. Notice that URIs that are reachable following many paths will only be included once in the result and that infinite loops will be avoided, due to the counter mechanism.

```
(deductiverule collect_refs
  (? (uri "http://lpis.csd.auth.gr")
     ((uri (dcq:references)) ?uri))
```

```
 =>
  (result (uri ?uri)))
```

Recursive sub-paths can be implicitly included in a path of unknown length. For example in the following generalized path, the multifield variable `$?p` can represent both linear and recursive sub-paths: `((dc:title $?p) ?t)`.

Multifield variables can also occur at the place of value expressions, since all RDF properties are treated as multislots. The following pattern retrieves in a list `$?l` all the values for the `rss:link` property of a resource object: `(rss:link $?l)`. On the other hand, if we know that a resource object has many values for one property and we want to iterate over them, the pattern should be: `(rss:link $? ?l $?)`, which means that variable `?l` will eventually become instantiated with all the values of the property `rss:link`. This retrieval pattern is so common that a shortcut `(rss:link ??l)` is provided which expands to the above pattern during a macro expansion phase.

When the value of a specific variable is of no interest then an anonymous variable '?' can be used, which is replaced by a singleton system-generated variable during the macro expansion phase.

Selection conditions can be placed inside value expressions, as in CLIPS. For example, the following pattern retrieves the family name in a variable and, at the same time, tests if the slot value does not equal "Smith":

```
(vcard:Family ?last&~"Smith")
```

Conditions can also express disjunction and negation. Only stratified negation is allowed. Rule conclusion can also contain a set of function calls that calculate the values to be stored at the slots of the derived object. Such calls are placed inside a `calc` construct before the derived class template. For example, the following variation of rule `q9` retrieves the given and family name of a resource object and using a CLIPS function concatenates them into a single string that is stored in the slot `full-name` of the derived objects of class `person`.

```
(deductiverule q9-variation
  (? (vcard:Family ?f) (vcard:Given ?v))
 =>
  (calc (bind ?full (str-cat ?v " " ?f)))
  (person (full-name ?full)))
```

Finally, R-DEVICE supports aggregate functions and grouping in the form of aggregate attribute rules. These rules express how values are accumulated and combined into attributes of existing objects. For example, the following rule iterates over all resources and generates one object for each distinct creator, which holds in the `URIs` slot all the resources that he/she has created.

```
(deductiverule ex1-aggregate
  (? (dcq:creator ?c) (uri ?uri))
 =>
  (pages (author ?c) (URIs (list ?uri))))
```

Function `list` is an aggregate function that just collects values in a list. There are several other aggregate functions, such as `sum`, `count`, `avg`, etc. Notice that in the above example a grouping is performed because the conclusion contains the slot `author` in addition to the aggregate slot `URIs`.

4.2 RuleML syntax of R-DEVICE rules

R-DEVICE rule language has also a RuleML [8] compatible syntax. We have tried to keep as close as possible to the DTD-based RuleML version 0.85[1]. However, several features of R-DEVICE could not be captured by the latest RuleML DTDs, so we have developed a new DTD (see **Fig. 2**) using the modularization scheme of RuleML, extending the Datalog with negation as failure DTD with OO features.

Rule `q5` that has been presented above is represented in RuleML notation as:

```
<imp>
  <_rlab ruletype="deductiverule" maintainable="yes">
    <ind>q5</ind>
  </_rlab>
  <_head>
    <atom>
      <_opr><rel><ind>result</ind></rel></_opr>
      <_slot name="link">  <var type="single">link</var>
      </_slot>
    </atom>
  </_head>
  <_body>
    <atom>
      <_opr><rel><ind>rss:item</ind></rel></_opr>
      <_slot name="rss:title">
        <_and>  <var type="single">title</var>
                <function_call name="str-index">
                  <ind>"RDQL"</ind>
                  <var type="single">title</var>
                </function_call>
        </_and>
      </_slot>
      <_slot name="rss:link">  <var type="single">link</var>
      </_slot>
    </atom>
  </_body>
</imp>
```

There are three types of rules: deductive rules, derived attribute rules and aggregate attribute rules. Classes and objects (facts) can also be declared in R-DEVICE; however, the focus in this paper 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`. Variables belong to three types, single, multi, and a combined form to reflect variables expressions in the previous subsection.

[1] In the future we will upgrade to newer XSD-based versions of RuleML (e.g. 0.87).

Finally, simple slot expressions have been augmented with the ability to declare path expressions according to the R-DEVICE abilities, i.e. simple ground path expressions, simple path expressions with variables, generalized path expressions, recursive path expressions, etc. Notice that the relation name of the operator can be either a constant (class name) or a variable, since R-DEVICE allows variable to range over class and slot names. Furthermore, each `atom` element has been augmented with an optional `_id` element to represent the OID of the corresponding resource object.

```
<!ENTITY % CLASSes "NMTOKENS">
<!ATTLIST _rlab
      ruletype (deductiverule | derivedattrule | aggregateattrule) #REQUIRED
      maintainable (yes | no) "yes">
<!ATTLIST var type (single | multi | single-multi) #REQUIRED>
<!ENTITY % recpath.content "(slotname+)">  <!ELEMENT recpath %recpath.content;>
<!ENTITY % genpath.content "(var)">        <!ELEMENT genpath %genpath.content;>
<!ENTITY % slotname.content "(ind|var)"> <!ELEMENT slotname %slotname.content;>
<!ELEMENT _varslot %_slot.content;>
<!ENTITY % _path.content "(slotname|genpath|recpath)+">
<!ELEMENT _path (%_path.content;, %_slot.content;)>
<!ENTITY % rel.content "(ind | var)">
<!ENTITY % _id.content "(ind | var)">      <!ELEMENT _id %_id.content;>
<!ENTITY % atom.content "((_id?,_opr,(_path|_slot|_varslot)*, ...))">
<!ENTITY % _calc.cont "(function_call+)">    <!ELEMENT calc %_calc.cont;>
<!ENTITY % _head.content "(calc?, atom)">
<!ELEMENT aggregate_function_call (var)>
<!ATTLIST aggregate_function_call
        name (sum|count|list|avg|max|min|ord_list|set|string|phrase) #REQUIRED>
<!ELEMENT function_call (%pos_term;)*>
<!ATTLIST function_call name CDATA #REQUIRED>
<!ENTITY % pos_term "(ind | var | function_call)">
<!ENTITY % term "(_not | %pos_term;)">     <!ELEMENT _not (ind | var)>
<!ELEMENT _or (%term;, (%term;)+)>         <!ELEMENT _and (%term;, (%term;)+)>
<!ENTITY % constraint "(_not | _or | _and)">
<!ENTITY % _slot.content "(ind | var | %constraint;|aggregate_function_call)">
<!ENTITY % nafurdatalog_include SYSTEM
                      "http://www.ruleml.org/0.85/dtd/naf/nafurdatalog.dtd">
%nafurdatalog_include;
<!ATTLIST rulebase  xmlns %URI; #IMPLIED  xsi:schemaLocation %URI; #IMPLIED
             xmlns:xsi %URI; #IMPLIED  rdf_import CDATA #IMPLIED
             rdf_export_classes %CLASSes; #IMPLIED rdf_export CDATA #IMPLIED>
```

Fig. 2. DTD for the RuleML syntax of the R-DEVICE rule language.

4.3 Extracting Inference Results as RDF Documents

After production rules have been executed and all the derived objects have been generated, the RDF extractor generates an RDF/XML document and returns it to the user through a Web server, as the result of the user's program. The document contains both RDF definitions for the schema of the desired derived classes and, of course, for the instances of the derived classes.

Initially the result document contains namespace definitions for rdf/rdfs and for the exported document. Then the derived classes are defined as `rdfs:Class` elements, followed by `rdfs:Property` elements for each of their slot that was defined at the deductive rule conclusion. Domains and ranges of the properties are obtained from the COOL definition of each derived class. The domain for all the properties of a class is the class itself, while the range can be one of the following:

- If the slot is of type `STRING`, the property range is `rdfs:Literal`.
- If the slot is of type `INTEGER`, the property range is `xsd:integer`.
- If the slot is of type `FLOAT`, the property range is `xsd:float`.
- If the slot is of type `INSTANCE`, the property range is a class whose name is obtained by the metadata. When the referenced class is a dummy class that was generated from the system to cater for multi-range properties, then multiple property ranges are generated for each one of the superclasses of the dummy class.
- If the slot has any other combination of types, then there is no range constraint.

Notice that in the fourth case the RDF schema for the referenced class(es) must be included in the result document, unless it is not a derived but a base RDF class, i.e. an RDF class whose definition has been imported from a namespace. In this case, the document header is enriched with the namespace address and no further schema definition is included. Otherwise, definitions for the referenced class and its properties are included in the RDF result document. The same actions are recursively repeated for all classes, that are reachable by reference slots from the initial class.

Below all class and property definitions, appear RDF statements about the instances of those classes. For each of the initial derived classes, all its objects are included using class names as outside elements and property names as inside elements. Only slots/properties that do have a value are included. Properties with multiple values are represented as multiple consecutive elements. Finally, objects recursively referenced from the above objects are also included in the result document. When instances of base RDF classes are included the outside element is `rdf:Description`, because type information is included as one or more `rdf:type` properties. The URIs of the derived objects are constructed from the URI of the R-DEVICE system (`http://startrek.csd.auth.gr/r-device/export/`), the name of the exported file and a uniquely generated anchor ID, constructed from the class name and consecutive integers. The URIs of instances of base classes are taken from the `uri` slot of each object.

```
<!DOCTYPE rdf:RDF [
    <!ENTITY dmoz "http://directory.mozilla.org/rdf/">
]>
<rdf:RDF xmlns:rdf="http://www.w3c.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:dmoz="&dmoz;">
  <dmoz:Topic rdf:about="&dmoz;Top">
    <dmoz:catid>1</dmoz:catid>
    <dc:title>Top</dc:title>
    <dmoz:narrow rdf:resource="&dmoz;Top/Arts"/>
  </dmoz:Topic>
  <dmoz:Topic rdf:about="&dmoz;Top/Arts">
    <dmoz:catid>2</dmoz:catid>
    <dc:title>Arts</dc:title>
    <dmoz:link rdf:resource="http://www3.bc.sympatico.ca/GlassPage.html"/>
  </dmoz:Topic>
  <dmoz:ExternalPage rdf:about="http://www3.bc.sympatico.ca/GlassPage.html">
    <dc:title>John Phillips Blown glass</dc:title>
    <dc:description>A small display of glass by John Phillips</dc:description>
  </dmoz:ExternalPage>
</rdf:RDF>
```

Fig. 3. Sample RDF/XML document.

The following R-DEVICE rule `example` retrieves the title of an ODP topic that has at least one associated page, along with the titles of all associated pages, and when applied on the RDF document of **Fig. 3**, exports the results shown in **Fig. 4**.

```
(deductiverule example
  (dmoz:Topic (dc:title ?t) (dmoz:link $? ?l $?))
  ?l <- (dmoz:ExternalPage (dc:title ?lt))
 =>
  (result (title ?t) (link_title ?lt)))
```

```
<!DOCTYPE rdf:RDF [
    <!ENTITY rdf "http://www.w3c.org/1999/02/22-rdf-syntax-ns#">
    <!ENTITY rdfs "http://www.w3c.org/2000/01/rdf-schema#">
    <!ENTITY r_device
             "http://startrek.csd.auth.gr/r-device/export/example-result.rdf#">
]>
<rdf:RDF  xmlns:rdf='&rdf;'  xmlns:rdfs='&rdfs;'
          xmlns:r_device='&r_device;'>
  <rdfs:Class rdf:about='&r_device;result'> </rdfs:Class>
  <rdf:Property rdf:about='&r_device;title'>
     <rdfs:domain rdf:resource='&r_device;result'/>
  </rdf:Property>
  <rdf:Property rdf:about='&r_device;link_title'>
     <rdfs:domain rdf:resource='&r_device;result'/>
  </rdf:Property>
  <r_device:result rdf:about=" r_device;result1">
     <r_device:title>Arts</r_device:title>
     <r_device:link_title>John phillips Blown glass</r_device:link_title>
  </r_device:result>
</rdf:RDF>
```

Fig. 4. Exported results for R-DEVICE rule `example`.

5 Conclusions and Future Work

In this paper we have presented R-DEVICE, a deductive rule language for reasoning about RDF metadata. 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. Users can define views which are materialized and, optionally, incrementally maintained. Users can also choose between an OPS5/CLIPS-like or a RuleML-like syntax. 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.

Regarding potential applications, R-DEVICE could be used as an inference mechanism on top of an RDF repository. The RDF data would be pre-loaded and external users would submit rule programs into the system either through a form-based HTML interface or using R-DEVICE remotely as a Web-service through SOAP messaging. Changes to the base RDF metadata of the repository would be incrementally

propagated to R-DEVICE, as well. Another use for R-DEVICE could be an on-the-fly RDF inferencing service, provided that the input RDF documents are not very large, since parsing very large RDF/XML documents into triples at run-time and then importing them in R-DEVICE would not be very efficient.

We are currently developing a defeasible logic extension of R-DEVICE [3] which will be used as a backend reasoning mechanism of negotiating agents to express and apply various negotiation strategies [14]. Among our plans for further developing R-DEVICE is to extend the system to handle ontologies in OWL [26], by using the R-DEVICE rule language to define the extended semantics of OWL, rather than building-in the semantics into the system, from scratch. Of course, the starting point of this investigation will be the Horn definable part of OWL [15]. Furthermore, we are planning to develop a visual editor for the RuleML-like rule language. Another extension would be to turn R-DEVICE into a Web Service. Finally, we aim to develop an interface to RDF storage systems, such as ICS-FORTH RDFSuite [1] or Sesame [9].

6 References

[1] Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., and Tolle K., "The ICSFORTH RDFSuite: Managing Voluminous RDF Description Bases", *Proc. 2nd Int. Workshop on the Semantic Web*, pp. 1-13, Hong Kong, 2001.

[2] Antoniou G., Wagner G., "Rules and Defeasible Reasoning on the Semantic Web", in *Proc. RuleML Workshop 2003*, Springer-Verlag, LNCS 2876, pp. 111–120, 2003.

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

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

[5] Bassiliades N., Vlahavas I., "R-DEVICE: An Object-Oriented Knowledge Base System for RDF Metadata", *Technical Report TR-LPIS-141-03*, LPIS Group, Dept. of Informatics, Aristotle University of Thessaloniki, Greece, 2003.

[6] Berners-Lee T., "CWM - closed world machine", `http://www.w3c.org/2000/10/swap/doc/cwm.html`, 2000.

[7] Berners-Lee T., Hendler J., Lassila O., "The Semantic Web", Scientific American, May 2001.

[8] Boley, H., Tabet, S., Wagner, G., "Design Rationale of RuleML: A Markup Language for Semantic Web Rules", *Proc. Int. Semantic Web Working Symp.*, pp. 381-402, 2001.

[9] Broekstra J., Kampman A., van Harmelen F., "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema", *Proc. 1st Int. Semantic Web Conf.*, Springer-Verlag, LNCS 2342, pp. 54-68, 2002.

[10] *CLIPS Basic Programming Guide*, Version 6.20, March 31st 2002, `http://www.ghg.net/clips/Download.html`.

[11] Decker S., Brickley D., Saarela J., Angele J., "A query and inference service for RDF", in *QL'98 - The Query Languages Workshop*, Boston, USA, 1998.

[12] Gandon F. L., Sheshagiri M., Sadeh N. M., "ROWL: Rule Language in OWL and Translation Engine for JESS", `http://mycampus.sadehlab.cs.cmu.edu/public_pages/ROWL/ROWL.html`

[13] Gandon F., Sadeh N., “Semantic Web Technologies to Reconcile Privacy and Context Awareness”, *Web Semantics Journal*, Vol. 1, No. 3, 2004.

[14] Governatori G., Dumas M., Hofstede A. ter and Oaks P., “A formal approach to legal negotiation”, *Proc. ICAIL 2001*, pp. 168-177.

[15] 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.

[16] Grosof B.N., Gandhe M.D., Finin T.W., “SweetJess: Translating DAMLRuleML to JESS”, *Proc. RuleML Workshop*, 2002.

[17] Hayes P., "RDF Semantics", *W3C Recommendation*, 10 Feb. 2004, `http://www.w3c.org/TR/rdf-mt/`

[18] Horrocks I., Patel-Schneider P. F., Boley H., Tabet S., Grosof B., Dean M., "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", Version 0.5, 19 Nov 2003, `http://www.daml.org/2003/11/swrl/`

[19] Jang M., "Bossam - A Java-based Rule Processor for the Semantic Web", `http://mknows.etri.re.kr/bossam`

[20] McBride B., "Jena: A Semantic Web Toolkit", *IEEE Internet Computing*, 6(6), pp. 55-59, 2002.

[21] Nejdl W., Wolf B., Qu C., Decker S., Sintek M., Naeve A., Nilsson M., Palmer M., Risch T., "Edutella: A P2P networking infrastructure based on RDF", in *Proc. of WWW-2002*, ACM Press, 2002, pp. 604-615.

[22] Prud'hommeaux E., “RDF Query and Rules Status”, `http://www.w3c.org./2001/11/13-RDF-Query-Rules/`

[23] *Resource Description Framework (RDF)*, `http://www.w3c.org/RDF/`

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

[25] Sintek M., Decker S., “TRIPLE-A Query, Inference, and Transformation Language for the Semantic Web”, *Proc. 1st Int. Semantic Web Conf.*, Springer-Verlag, LNCS 2342, pp. 364-378, 2002.

[26] *Web Ontology Language (OWL)*, `http://www.w3c.org/2004/OWL/`

[27] *Xalan-Java XSLT processor*, `xml.apache.org/xalan-j/`