O-DEVICE: An Object-Oriented Knowledge Base System for OWL Ontologies

Georgios Meditskos, Nick Bassiliades

Department of Informatics, Aristotle University of Thessaloniki, Greece
{gmeditsk, nbassili}@csd.auth.gr

Abstract. This paper reports on the implementation of a rule system, called O-DEVICE, for reasoning about OWL instances using deductive rules. O-DEVICE exploits the rule language of the CLIPS production rule system and transforms OWL ontologies into an object-oriented schema of COOL. During the transformation procedure, OWL classes are mapped to COOL classes, OWL properties to class slots and OWL instances to COOL objects. The purpose of this transformation is twofold: a) to exploit the advantages of the object-oriented representation and access all the properties of instances in one step, since properties are encapsulated inside resource objects; b) to be able to use a deductive object-oriented rule language for querying and creating maintainable views of OWL instances, which operates over the object-oriented schema of CLIPS, and c) to answer queries faster, since the implied relationships due to the rich OWL semantics have been pre-computed. The deductive rules are compiled into CLIPS production rules. The rich open-world semantics of OWL are partly handled by the incremental transformation procedure and partly by the rule compilation procedure.

1. Introduction

The vision of the Semantic Web is to provide the necessary standards and infrastructure for transforming the Web into a more automatic environment where agents would have the ability to search for requested information automatically. This is feasible by describing appropriately the already available data on the Web in a way that could be machine-understandable. Ontologies can be considered as a primary key towards this goal since they provide a controlled vocabulary of concepts, each with an explicitly defined and machine processable semantics.

The development of Semantic Web proceeds in layers where each layer is built on top of the others [6]. Currently, the *ontology* layer has reached a sufficient level of maturity, having OWL [19] as the basic form for ontology definition. The next step is to move on the higher levels of *logic* and *proof*, which are built on top of *ontology* layer, where rules now are considered as the primary key, since (a) they can serve as extensions of, or alternatives to, description logic based ontology languages; and (b) they can be used to develop declarative systems on top of (using) ontologies.

A lot of effort is undertaken to define a rule language for the Semantic Web on top of ontologies in order to combine already existing information and deduce new knowledge. Currently, RuleML [8] is the main standardization effort for rules on the Web to specify queries and inferences in Web ontologies, mappings between ontologies, and dynamic Web behaviors of workflows, services, and agents. Furthermore, very recently the Rule Interchange Format Working Group [18] has been formed to produce a core rule language plus extensions which together allow rules to be translated between rule languages and thus transferred between rule systems.

One approach to implement a rule system on top of the Semantic Web ontology layer is to start from scratch and build inference engines that draw conclusions directly on the OWL data model. However, such an approach tends to throw away decades of research and development on efficient and robust rule engines. In this paper we follow a different approach: we re-use an existing rule system (CLIPS [10]) for reasoning on top of OWL data. However, before an existing rule system is used, careful design must be made on how OWL data and semantics are going to be treated in the host system. The design should be sufficient enough to (a) draw the right conclusions stemming from the semantics of the language and (b) complete the inferencing procedure in a reasonable amount of time.

The O-DEVICE system inferences over (on top of) OWL documents. O-DEVICE exploits the advantages of the object-oriented programming model by transforming OWL ontologies into classes, properties and objects of the OO programming language provided within CLIPS, called COOL. The system also features a powerful deductive rule language which supports inferencing over the transformed OWL descriptions. Users can either use this deductive language to express queries or a RuleML-like syntax. The deductive rule language is implemented by translating deductive rules into CLIPS production rules. The semantics of OWL constructors are appropriately handled by O-DEVICE, either by the OWL transformation procedure, using corresponding COOL constructs, or by the deductive rule compilation procedure, rewriting parts of the rule condition.

Our main motivation for doing such a transformation from OWL to objects is to be able to exploit our existing deductive object-oriented rule language ([3], [4], [2]) for querying and creating maintainable views of OWL instances, taking into consideration the complex, implied relationships between classes and instances, due to the rich OWL semantics. Notice that our purpose is not to build another OWL reasoner, i.e. we do not aim to classifying instances under classes, but rather to infer and materialize in advance as much properties for OWL instances as possible under the semantics of OWL constructors. In this way we are able to answer deductive queries at run-time much faster, since all the implied relationships have been pre-computed. Finally, although the host system is restricted by the closed-world assumption, our current mapping scheme is able (in most situations) to cope with the open-world semantics of OWL, due to our incremental transformation algorithms.

This paper extends the work first presented in [16] by adding more OWL constructs to the mapping scheme. However, the work described in this paper is still work in progress. The rest of the paper is organized as follows: Section 2 presents the functionality of the system. Section 3 describes the transformation procedure of OWL constructors into COOL. Section 4 briefly describes the rule language of O-DEVICE. Section 5 presents related work on rule systems on top of ontologies. Finally, Section 6 concludes with a summary and potential future work.

2. O-DEVICE Functionality

In this section we describe in details of the O-DEVICE system architecture and functionality (Fig. 1) and the way of each component participates in the data flow.

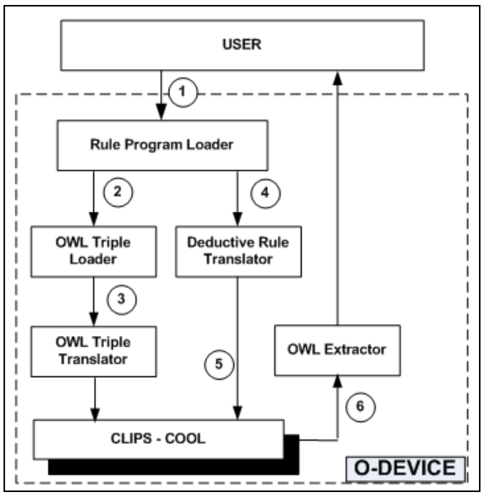

Fig. 1. Architecture of O-DEVICE.

System components: The system consists of five basic modules.
i) Rule Program Loader: Accepts from the user the URL of a RuleML file and saves it locally. The rule file also contains information about the location of the OWL files, the names of the derived classes to be exported as results and the name of the output OWL file. The *Rule Program Loader* scans the rule file and collects the appropriate information for later use. The RuleML program is translated into the native O-DEVICE rule notation using an XSLT stylesheet.
ii) OWL Triple Loader: Accepts from the *Rule Program Loader* the URLs of the OWL files that has found from the RuleML document and saves them locally. Furthermore, it uses the ARP Parser [15] to translate the OWL document in the N-Triple format and saves them locally too. The triple loader has been implemented as an extension of the R-DEVICE system [2], which imports RDF Schema ontologies and RDF data into CLIPS.
iii) Deductive Rule Translator: Accepts from the *Rule Program Loader* the set of O-DEVICE rules and translates them into a set of CLIPS production rules. CLIPS runs the production rules and generates the objects that constitute the result of the rule program.
iv) OWL Triple Translator: Accepts from the *OWL Triple Loader* the produced triples from the ARP parser and transforms them into classes, properties and objects of COOL according to the mapping scheme which is described later.
v) OWL Extractor: Accepts objects generated (derived) by the production rules and exports them to the user as an OWL document.

Data flow: The data flow of the system can be considered as a 6-step procedure (Fig. 1): the user inputs (*step 1*) the URL of the RuleML rule file to the *Rule Program Loader*, which downloads it. The *Rule Program Loader* scans the rule file to target the relevant OWL documents and passes theirs URLs to the *OWL Triple Loader* (*step 2*). It uses the ARP Parser to translate the OWL document in the N-Triple format and passes the produced triples to the *OWL Triple Translator* (*step 3*) which transforms them into classes, properties and objects of COOL. The O-DEVICE rule program (from the translation of the RuleML file) is then forwarded to the *Deductive Rule Translator (step 4)* which translates them into a set of CLIPS production rules. After the translation of deductive rules or the loading of the compiled rules, CLIPS runs the production rules (*step 5*) and generates the objects that constitute the result of the rule program. The result-objects are exported to the user (*step 6*) as an OWL document through the *OWL Extractor*.

3. OWL Constructor Transformation

The transformation procedure of OWL constructors is a critical task which affects both the quality of results and the performance of the system. Careful design must be made in order (a) to preserve the open-world OWL semantics by exploiting the available constructs of COOL, whenever possible, (b) to define incremental, rule-based algorithms to emulate some of the semantics that could not be directly mapped to COOL and (c) to make the system efficient enough to complete the tasks in a reasonable amount of time.

3.1. Basic Transformation Principles

The mapping scheme of OWL ontologies and data to objects tries to exploit as many built-in features of COOL as possible, in order to query and reason about OWL objects faster. The main features of the mapping scheme are the following:

Built-in OWL classes: These classes are represented both as classes and as objects, instances of the `rdfs:Class` class. This binary representation is due to the fact that COOL does not support meta-classes, so the role of meta-class is played by the instances of `rdfs:Class` class.
meta-classes: Meta-classes are needed in order to store certain information about a class. So, for example, the OWL class *Male* (in section 3.2.1) is represented in O-DEVICE both by a `defclass Male` construct and a `[Male]` object that is an instance of the `owl:Class` class.

User-defined classes: They follow the same scheme except for the fact that the "meta-class" objects are instances of the class `owl:Class.` Inheritance issues of class hierarchies are treated by the class-inheritance mechanism of COOL, for inheriting properties from superclasses to subclasses, for including the extensions of subclasses to the extensions of the superclasses and for the transitivity of the `rdfs:subClassOf` property.

OWL data: All OWL data (resources) are represented as COOL objects, direct or indirect instances of the `owl:Thing` class.

Properties: Properties are instances of the class `owl:DatatypeProperty` or `owl:ObjectProperty`. This also includes subclasses of the above classes, such as `owl:TransitiveProperty`. Furthermore, properties are defined as slots (attributes) of their domain class(es). The values of properties are stored inside resource objects as slot values. OWL properties are multislots, i.e. they store lists of values, because a resource can have multiple times the same property attached to it.

3.2. Preserving OWL Semantics

O-DEVICE currently handles ontologies in OWL DL, which supports rich expressiveness and gives computational guarantees. In the subsections below, we describe how the system handles some of the OWL constructors, in order to preserve their semantics, giving for each case a short example. A complete list of all transformations can be found in [17].

3.2.1 Property Restrictions

Value constraints are declared with the properties `owl:allValuesFrom`, `owl:someValuesFrom`, `owl:hasValue` and the cardinality constrains with the properties `owl:cardinality`, `owl:minCardinality`, `owl:maxCardinality`.

Restriction `owl:hasValue`

The `owl:hasValue` constraint is partly implemented using the built-in mechanism of COOL for handling default values. We can declare a *default* value for a slot, making all the instances of the class to have by default this value, if a value is not provided when creating the instance. The following example describes the class of `Male`:

```
<owl:Class rdf:ID="Male">
   <rdfs:subClassOf rdf:resource="#Human"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#hasGender" />
         <owl:hasValue rdf:resource="#male" />
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Assuming that there is a class named `Human` with a property `hasGender`, the above example is represented in COOL as follows:

```
(defclass example:Male (is-a example:Human gen1)
   (multislot example:hasGender (type INSTANCE-NAME)
            (default [example:male])))
```

For all instances of class `Male` that do not have any value for slot `example:hasGender`, the value of property `hasGender` will be `[example:male]`. If an instance is created that does have a value for that slot, then a check is performed to see whether the default value is a member of the multislot, otherwise it is added. Such a behaviour is an extension of the simple semantics of the CLIPS default mechanism.

Restriction `owl:cardinality`

Cardinality restrictions are handled directly via the *cardinality* mechanism of COOL. Consider the following example of `owl:cardinality` property stating that a `Human` has only one biological mother (`hasBiologicalMother` property):

```
<owl:Class rdf:ID="Human">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasBiologicalMother" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The representation of class `Human` in COOL is as follows:

```
(defclass example:Human (is-a gen1)
     (multislot example:hasBiologicalMother
        (type INSTANCE-NAME)(cardinality 1 1)))
```

By this definition, the property `hasBiologicalMother` can take only one value. If more than one values are to be placed in the slot, the system will ignore the others, keeping only the first. For `owl:maxCardinality` and `owl:minCardinality` we follow the same implementation using `(cardinality ?VARIABLE <value>)` and `(cardinality <value> ?VARIABLE)` respectively.

3.2.2 Boolean Combination of Classes

In OWL it is possible to create new classes by combining existing classes through Boolean operators. For example, the `owl:unionOf` property links a class to a list of class descriptions and defines the new class extension as those individuals that occur in at least one of the class extensions of the class descriptions in the list. We describe the use of this property using the following simple example.

```
<owl:Class rdf:ID="Fruit">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SweetFruit" />
    <owl:Class rdf:about="#NonSweetFruit" />
  </owl:unionOf>
</owl:Class>
```

Union of classes is implemented as a common superclass. O-DEVICE handles the above example as follows:

```
(defclass example:Fruit (is-a owl:Thing))
(defclass example:NonSweetFruit (is-a example:Fruit))
(defclass example:SweetFruit (is-a example:Fruit))
```

Notice that if the classes `NonSweetFruit` and `SweetFruit` have already been defined, then the OO schema should be re-defined at run-time. This includes backing-up all instances and definitions of the re-defined class(es), deleting all instances of the re-defined class(es), including their subclasses, un-defining the re-defined classes (and subclasses) and, finally, re-defining the class(es) and restoring their instances.

3.2.3 Special Properties

In OWL several special characteristics of properties can be defined such as transitivity, symmetry, etc. For example, when a property *P* is symmetric then if the pair (x,y) is an instance of *P*, then the pair (y,x) is also an instance of *P*. Consider the example:

```
<owl:Class rdf:ID="Human" />
<owl:SymmetricProperty rdf:ID="friendOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range  rdf:resource="#Human"/>
</owl:SymmetricProperty>
<Human rdf:ID="george"><friendOf rdf:resource="#nick"/></Human>
<Human rdf:ID="nick" />
```

The above example states that `george` is `friendOf` `nick` but because `friendOf` is symmetric, the system infers that `nick` is also `friendOf george`. The corresponding instances in O-DEVICE are:

```
[example:nick] of example:Human       [example:george] of example:Human
(uri example:nick)                    (uri example:george)
.....                                 .....
(example:friendOf [example:george])   (example:friendOf [example:nick])
```

Notice that the materialization of special property characteristics is incremental, i.e. their algorithms will be applied to all future individuals. In this way, our mapping scheme is compatible to the open-world semantics of OWL. Notice, however, that currently our mapping scheme does not handle the existential OWL construct.

4. The Deductive Rule Language of O-DEVICE

The deductive rule language of O-DEVICE supports inferencing over OWL instances represented as objects and defines materialized views over them, possibly incrementally maintained. 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 ([2], [3], [4]). Each deductive rule in O-DEVICE is implemented as a CLIPS production rule that inserts a derived object when the condition of the deductive rule is satisfied.

The following rule retrieves the names of all `Woman` instances that have a value less than 22 in the `age` property by deriving instances of class `young-woman` with the value `?fname` in the `fname` property:

```
(deductiverule young-women
  (test:Woman (test:age ?x&:(< ?x 22)) (test:fname ?fname))
 =>
  (young-woman (fname ?fname)))
```

The above deductive rule refers to the following OWL document:

```
<owl:Class rdf:ID="Human" />
<owl:Class rdf:ID="Man">
   <rdfs:subClassOf rdf:resource="#Human" />
</owl:Class>
<owl:Class rdf:ID="Woman" >
   <owl:complementOf rdf:resource="#Man" />
   <rdfs:subClassOf rdf:resource="#Human" />
</owl:Class>
```

Assuming that there are two datatype properties in class `Human`, namely `fname` (string) and `age` (integer), we can see that the class `Woman` is `complementOf` the class `Man` and both classes are `subClassOf` the class `Human`.

The above deductive rule is translated into the following CLIPS production rule:

```
(defrule gen1-gen3
  (object (name ?gen2) (is-a test:Human & ~test:Man)
          (test:age ?x&:(< ?x 22)) (test:fname ?fname))
 =>
  (bind ?oid (symbol-to-instance-name (sym-cat young-woman ?fname)))
  (make-instance ?oid of young-woman (fname ?fname)))
```

Notice that the class `Woman` of the deductive rule is replaced by the "not" connective constraint `~Man` in the `is-a` constraint of the production rule condition, meaning that objects of all but the `Man` class are retrieved. In this way, we are able to implement the strong negation of OWL into a production rule environment where the closed world assumption holds and only negation-as-failure exists. Of course, the answers to the above rule depend on the time the query runs. If further OWL instances are added and the query is re-run, a different answer will be obtained. This means that the answer involves only the currently existing instances, i.e. it follows the closed-world assumption. However, the non-monotonic semantics of our rule language (incremental materialization) compensates for future changes in the knowledge base, thus we are able to cope with the open-world semantics of OWL. Furthermore, notice that the superclass `Human` of `Woman` class is also added in the `is-a` constraint to avoid searching for all completely irrelevant to this taxonomy objects.

The action-part of the above production rule simply creates the derived object, after generating an OID based on the class name and the derived object's property values. Maintainable deductive rules have a more complex translation.

The semantics of CLIPS production rules are the usual production rule semantics: rules whose condition is successfully matched against the current data are triggered and placed in the conflict set. The conflict resolution mechanism selects a single rule for firing its action, which may alter the data. In subsequent cycles, new rules may be triggered or un-triggered based on the data modifications. The criteria for selecting rules for the conflict set may be priority-based or heuristically based. Rule condition matching is performed incrementally, through the RETE algorithm.

5. Related Work

A lot of effort has been made to develop rule engines for reasoning on top of OWL ontologies. SweetJess [12] is an implementation of a defeasible reasoning system (situated courteous logic programs) based on Jess that integrates well with RuleML. However, SweetJess rules can only express reasoning over ontologies expressed in DAMLRuleML (a DAML-OIL like syntax of RuleML) and not on arbitrary OWL data. Furthermore, SweetJess is restricted to simple terms (variables and atoms).

SweetProlog [14] is a system for translating rules into Prolog. This is achieved via a translation of OWL ontologies and rules expressed in OWLRuleML into a set of facts and rules in Prolog. It makes use of three languages: Prolog as a rule engine, OWL as an ontology and OWLRuleML as a rule language. It enables reasoning (through backward chaining) over OWL ontologies by rules via a translation of OWL subsets into simple Prolog predicates which a JIProlog engine can handle. There are five principle functions that characterize SweetProlog: a) translation of OWL and

OWLRuleML ontologies into RDF triples, b) translation of OWL assertions into Prolog, c) translation of OWLRuleML rules into CLP, d) transformation of CLP rules into Prolog and e) interrogation of the output logic programs.

DR-Prolog [7] is a Prolog-based system for defeasible reasoning on the Web. The system is a) syntactically compatible with RuleML, b) features strict and defeasible rules, priorities and two kinds of negation, c) is based on a translation to logic programming with declarative semantics, and d) can reason with rules, RDF, RDF Schema and part of OWL ontologies. It supports monotonic and non-monotonic rules, open and closed world assumption and reasoning with inconsistencies.

SWRL [13] is a rule language based on a combination of OWL with the Unary/Binary Datalog sublanguages of RuleML. SWRL enables Horn-like rules to be combined with an OWL knowledge base. Negation is not explicitly supported by the SWRL language, but only indirectly through OWL DL (e.g. class complements). Its main purpose is to provide a formal meaning of OWL ontologies and extend OWL DL. There is a concrete implementation of SWRL, called Hoolet. Hoolet translates the ontology to a collection of axioms (based on the OWL semantics) which is then given to a first order prover for consistency checking. Hoolet has been extended to handle rules through the addition of a parser for an RDF rule syntax and an extension of the translator to handle rules, based on the semantics of SWRL rules.

SWSL [5] is a logic-based language for specifying formal characterizations of Web services concepts and descriptions of individual services. It includes two sublanguages: SWSL-FOL and SWSL-Rules. The latter is a rule-based sublanguage, which can be used both as a specification and an implementation language. It is designed to provide support for a variety of tasks that range from service profile specification to service discovery, contracting and policy specification. It is a layered language and its core consists of the pure Horn subset of SWSL-Rules.

WRL [1] is a rule-based ontology language for the Semantic Web. It is derived from the ontology component of the Web Service Modeling Language WSML. The language is located in the Semantic Web stack next to the Description Logic based Ontology language OWL. WRL consists of three variants, namely Core, Flight and Full. WRL-Core marks the common core between OWL and WRL and is thus the basic interoperability layer with OWL. WRL-Flight is based on the Datalog subset of F-Logic, with negation-as-failure under the Perfect Model Semantics. WRL-Full is based on full Horn with negation-as-failure under the Well-Founded Semantics.

ROWL [11] system enables users to frame rules in RDF/XML syntax using ontology in OWL. Using XSLT stylesheets, the rules in RDF/XML are transformed into forward-chaining rules in JESS. Further stylesheets transform ontology and instance files into Jess unordered facts that represent triplets. The file with facts and rules are then fed to JESS which enables inferencing and rule invocation.

F-OWL [9] is an ontology inference engine for OWL, which is implemented using Flora-2, an object-oriented knowledge base language and application development platform that translates a unified language of F-logic, HiLog, and Transaction Logic into the XSB deductive engine. Key features of F-OWL include the ability to reason with the OWL ontology model, the ability to support knowledge consistency checking using axiomatic rules defined in Flora-2, and an open application programming interface (API) for Java application integrations.

6. Conclusions and Future Work

In this paper we have presented O-DEVICE, a deductive object-oriented knowledge base system for reasoning over OWL documents. O-DEVICE imports OWL documents into the CLIPS production rule system by transforming OWL ontologies into an object-oriented schema and OWL instances into objects. In this way, when accessing multiple properties of a single OWL instance, few joins are required. The system also features a powerful deductive rule language which supports inferencing over the transformed OWL descriptions. The transformation scheme of OWL to COOL objects is partly based on the underlying COOL object model and partly on the compilation scheme of the deductive rule language. One of the purposes of the transformation is to infer and materialize in advance as much properties for OWL instances as possible under the rich semantics of OWL constructors. In this way we are able to answer deductive queries at run-time much faster, since all the implied relationships have been pre-computed.

Certain features of the descriptive semantics of OWL are still under development. For example, inverse functional properties are currently not handled at all, whereas they should be handled similarly to key properties, as in databases. Furthermore, when two objects have the same value for an inverse functional property it should be concluded that they stand for the same object. Finally, the existential restriction has not also been implemented.

All these interpretations of OWL constructs are currently being implemented by appropriately extending the *OWL Triple Translator* (Fig. 1) with production rules that assert extra triples, which are further treated by the translator. Notice that asserting new properties to an already imported ontology might call for object and/or class redefinitions, which are efficiently handled by the core triple translator of R-DEVICE [2]. Therefore, the triple translator is non-monotonic, and so is the rule language, since it supports stratified negation as failure and incrementally maintained materialized views. The non-monotonic nature of our transformation algorithms is a key to overcome the closed-world nature of the host system and allows us to emulate OWL's open-world semantics.

In the future we plan to deploy the reasoning system as a Web Service and to implement a Semantic Web Service composition system using OWL-S service descriptions and user-defined service composition rules.

Acknowledgments

This work was partially supported by a PENED program (EPAN M.8.3.1, No. 03EΔ73).

7. References

1. Angele J, Boley H., J. de Bruijn, Fensel D., Hitzler P., Kifer M., Krummenacher R., Lausen H., Polleres A., Studer R., "Web Rule Language (WRL)", Technical Report, http://www.wsmo.org/wsml/wrl/wrl.html
2. Bassiliades N., Vlahavas I., "R-DEVICE: An Object-Oriented Knowledge Base System for RDF Metadata", *International Journal on Semantic Web and Information Systems*, 2(2) (to appear), 2006.
3. Bassiliades N., Vlahavas I., and Elmagarmid A.K., "E DEVICE: An extensible active knowledge base system with multiple rule type support", *IEEE TKDE*, 12(5), pp. 824-844, 2000.
4. Bassiliades N., Vlahavas I., and Sampson D., "Using Logic for Querying XML Data", in *Web-Powered Databases*, Ch. 1, pp. 1-35, Idea-Group Publishing, 2003
5. Battle S., Bernstein A., Boley H., Grosof B., Gruninger M., Hull R., Kifer M., Martin D., McIlraith S., McGuinness D., Su J., Tabet S., "SWSL-rules: A rule language for the semantic web", *W3C rules workshop*, Washington DC, USA, April 2005
6. Berners-Lee T., Hendler J., and Lassila O., "The Semantic Web", *Scientific American*, 284(5), 2001, pp. 34-43.
7. Bikakis A., Antoniou G., DR-Prolog: A System for Reasoning with Rules and Ontologies on the Semantic Web 2005, *Proc. 25th American National Conference on Artificial Intelligence* (AAAI-2005).
8. Boley, H., Tabet, S., and Wagner, G., "Design Rationale of RuleML: A Markup Language for Semantic Web Rules", *Proc. Int. Semantic Web Working Symp*., pp. 381-402, 2001.
9. Chen H., Zou Y., Kagal L., Finin T., "F-OWL: An OWL Inference Engine in Flora-2", http://fowl.sourceforge.net/
10. CLIPS 6.23 Basic Programming Guide, http://www.ghg.net/clips
11. 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
12. Grosof B.N., Gandhe M.D., Finin T.W., "SweetJess: Translating DAMLRuleML to JESS", *Proc. RuleML Workshop*, 2002.
13. Horrocks I., Patel-Schneider P.F., Boley H., Tabet S., Grosof B., Dean M., "SWRL: A semantic web rule language combining OWL and RuleML", Member submission, May 2004, W3C. http://www.w3.org/Submission/SWRL/
14. Laera L., Tamma V., Bench-Capon T. and Semeraro G., "SweetProlog: A System to Integrate Ontologies and Rules", *3rd Int. Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, Springer-Verlag, LNCS 3323, pp. 188-193.
15. McBride B., "Jena: Implementing the RDF Model and Syntax Specification", *Proc. 2nd Int. Workshop on the Semantic Web*, 2001
16. Meditskos G., Bassiliades N., "Towards an Object-Oriented Reasoning System for OWL", *Int. Workshop on OWL Experiences and Directions,* 11-12 Nov. 2005, Galway, Ireland, 2005.
17. O-DEVICE web page, http://iskp.csd.auth.gr/systems/o-device/o-device.html
18. Rule Interchange Format Working Group, W3C, http://www.w3.org/2005/rules/wg
19. Web Ontology Language (OWL), http://www.w3.org/2004/OWL/