Capturing RDF Descriptive Semantics in an Object Oriented Knowledge Base System

Nick Bassiliades
Dept. of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
nbassili@csd.auth.gr

Ioannis Vlahavas
Dept. of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
vlahavas@csd.auth.gr

ABSTRACT

In this paper, we present a deductive object-oriented knowledge base system, called R-DEVICE, which imports RDF data into the CLIPS production rule system as objects and uses a deductive rule language for querying and reasoning about them. In our model properties of resources are not scattered across several triples as in most other RDF storage and querying systems, resulting in increased query performance due to less joins. R-DEVICE features a powerful deductive rule language which is able to express arbitrary queries both on the RDF schema and data, including generalized path expressions, stratified negation, aggregate, grouping, and sorting, functions, mainly due to the second-order syntax of the rule language which is efficiently translated into sets of first-order logic rules using metadata.

Keywords

RDF, Object Model, CLIPS, Descriptive Semantics, Deductive Rules, Production Rules

1. INTRODUCTION

The automatic management of information in the Semantic Web can be achieved by using metadata to describe the data contained on the Web. RDF is a general-purpose language for representing metadata about Web resources, such as the title, author, etc. RDF data model includes resources, literals and properties that link resources to literals or other resources. RDF has a schema definition language (RDFS), for creating classes of resources and property types.

The semantics of the RDF data model differ from traditional data structures, such as the object data model. Specifically, RDF is an assertional language, i.e. each assertion declares that certain information about resources is true, including schema information, and its meaning is not changed by future assertions. This kind of semantics is called descriptive. Traditional data models define certain constraints in their schema definitions and schema instances have to obey these constraints, i.e. no information entry that violates these constraints is allowed. This kind of semantics is called prescriptive.

A challenging task, in order to re-use existing query and inference systems with prescriptive semantics, is to be able to capture the descriptive RDF semantics in a traditional data model. In this paper, we present R-DEVICE, a deductive object-oriented knowledge base system, which transforms RDF triples into objects and uses a deductive rule language for querying and reasoning about them.

R-DEVICE imports RDF data into the CLIPS production rule system as COOL objects. In our object model, contrasted to the RDF data model, properties are treated both as first-class objects and as attributes of resource objects. In this way resource properties are gathered together in one object, resulting in superior query performance than the performance of a triple-based query model, as it has been experimentally shown. Most other RDF storage and querying systems that are based on a triple model scatter resource properties across
several triples and they require several joins to query 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.

R-DEVICE features a powerful deductive rule language which is able to express arbitrary queries both on the RDF schema and data, including generalized path expressions, stratified negation, aggregate, grouping, and sorting, functions, mainly due to the second-order syntax of the rule language, i.e. variables ranging over class and slot names, which is efficiently translated into sets of first-order logic rules using metadata. Furthermore, R-DEVICE rules define views which are materialized and incrementally maintained. Finally, users can use CLIPS functions or can define their own arbitrary functions using the CLIPS host language.

2. System Architecture

The R-DEVICE system consists of two major components (Figure 1): the RDF loader/translator and the rule 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 [1] to translate it to triples in the N-triple format. Both the RDF/XML and RDF/N3 files are stored locally for future reference. Furthermore, 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.

All N-triples are loaded into memory, while the resources that have a URI#anchorID or 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. 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 or query. Finally, the result-objects are exported to the user as an RDF document through the RDF extractor.

![Figure 1. Architecture of the R-DEVICE system.](image)

3. Mapping RDF Triples to Objects
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 a unique system-generated identifier. The URI address of resources is stored in the uri slot. Resources that their address has a namespace:label format do not store their address in the uri slot since the address can be reconstructed from the URI of the namespace. R-DEVICE also represents the documents of the namespaces as resource objects, storing their URI in the uri slot. The 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, 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 rdf:Property class. Additionally, properties with a single domain class are defined as slots (attributes) of this class. The values of properties are stored slot values of resource objects. 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. When a property has multiple domains, then a dummy class is generated which 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 must be attached to all resource objects, therefore they should become slots of the rdfs:Resource class, which is the root of the resource object hierarchy.

RDF descriptive semantics can cause dynamic class and/or object re-definitions in several occasions: a) add new properties to already existing classes, b) add a new rdfs:subClassOf property to an existing class, c) add a new rdf:type property to an existing object. R-DEVICE does not reject any RDF triple because every asserted triple is considered to be true.

The rdfs:range constraint of properties defines the type of the slots. Specifically, when this constraint is absent, then there is no type constraint for the slots, while if the value of the constraint is rdfs:Literal then the corresponding slot is of type STRING. Finally, when the value of the constraint is the name of a resource 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 which becomes the type of the slot.

Property hierarchies are treated in R DEVICE as follows: when property A is a subproperty of B, then property B can be used wherever property A can be used, but not vice-versa. Therefore, we consider property B as an alias of property A. The aliasing mechanism is implemented with an alias multislot that contains the explicit transitive closure of the property hierarchy. When a property is added as a slot in a class, the property hierarchy is navigated upwards and all the superproperties are added in the alias slot as aliases of the new property. Furthermore, when a new class is created the value of the alias slot of its superclass(es) is inherited.

The RDF triple translator is 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 the aliasing mechanism. The performance of triple translator (on a 2GHz Pentium 4) varies from 1.5 msec per triple for 1K triples to 37.9 msec per triple for 100K triples.

4. The R-DEVICE Rule Language

R-DEVICE belongs to a family of previous such deductive object-oriented rule languages ([2], [3]) and supports querying over RDF data represented as objects and define materialized views over them that are maintained incrementally. 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 rule 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 in 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 met and one for deleting the derived object when the condition is no more met, due to base object deletions and/or slot modifications. 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 syntax of R-DEVICE deductive rules is a variation of the syntax for CLIPS production rules. The following is an example of the query that retrieves the address of a page from the ODP repository [4] if that page has the same title with the topic that it is classified.

(deductiverule
  =>
  (result (title ?t) (page ?u))
)

The full syntax of R-DEVICE along with the full version of this paper that contains details on translating deductive rules into production rules can be found at [5]. Several query examples can be found at [5] and [6]. Performance comparison between our model and a triple-based query approach using several queries on ODP data have shown a considerable (sometimes over 10-fold) speed-up, which does not deteriorate with the size of RDF data.

5. ACKNOWLEDGEMENTS

This work was partially supported by the Greek R&D General Secretariat through a bilateral Greek-Ukrainian project (EPAN-M.4.3, No. 2013555). The first author was supported by a postdoctoral scholarship from the Greek Foundation of State Scholarships (F.S.S. - I.K.Y.).

6. REFERENCES