Design and Implementation of an RDF Triple Store *

Ching-Long Yeh and Ruei-Feng Lin
Department of Computer Science and Engineering
Tatung University
40 Chungshan N. Rd., Sec. 3 Taipei, 104 Taiwan
E-mail: chingyeh@cse.ttu.edu.tw

Abstract

In this paper we describe the design and implementation of an RDF indexing mechanism. An RDF document is parsed into the corresponding sequence of triples which are then stored in tables of relational database. To facilitate the access of RDF content, we develop a conceptual search program having inferencing capability based on the RDF schema. The parser and the conceptual search programs are implemented using Prolog. We have built a web site to demonstrate the creation of RDF store and conceptual search.

1 Introduction

Due to information overloading and explosion, the current browser-based user interfaces, that provide full text search and directory-based browsing, is not efficient for information consumption. To solve the above problem, it is essential to provide more machine-understandable information for computer instead of HTML documents, so that more effective applications can be developed, for example, conceptual search, intelligent question and answer (to replace FAQ systems), and automatic generation of multimedia document. Furthermore using the semantic information can enhance the management of web site content. XML can be used to separate the structure and presentation of document, which improves the shortcoming of HTML. However, it is not sufficient to represent the semantic information in XML document. RDF [1], a directed graph data model encoded in XML syntax, is designed as a language to represent semantic information. A vocabulary, containing class, subclass, property, subProperty, etc., is added to RDF to become RDF Schema (RDFS) [2], which is used as the basis to design ontology languages, for exam-

*This research was supported by the Taiwan National Science Council under Contract No. 91-2422-H-036-322.

ple, DAML+OIL [3]. An ontology language is used to constrain the property labels when creating RDF files. An annotation tool, for example, OilEd [4], is used to create the biographic information of a person by importing a biography schema in RDFS. Due to lack of indices in RDF files, we are not able to make full use of the meaning in RDF documents. In this paper, we attempt to develop an RDF store to facilitate the access of semantic information in RDF documents.

RDF data model is a directed graph, where a node is either a resource in the Web or a literal value of some kind, and an arc connecting two nodes represents a predicate (or property) [1]. A pair of nodes and the arc forms a subject-predicate-object statement. Thus an RDF document can be viewed as a collection of triple statements. The goal of this paper is to design a store to organize triples obtained from RDF documents and provide both user and application interfaces to access the semantic content in RDF documents.

We start with building an RDF parser that takes RDF document as input and produces the corresponding triple statements. The parser is implemented using the DCG in Prolog [5]. We choose relational database as the persistent store of triples because of its efficiency and reliability. We create five tables to stores the names space, resources, literals, predicates and triples in RDF documents. User can then retrieve the triples stored in the relational database through the ODBC interface. However, this approach is not convenient for user to access the content in RDF documents. In practice, we need a higher-level interface that user can access in conceptual level the content in RDF documents [6, 7]. We provide a simple syntax for user to express their conditions, in which optional attribute value pairs are appended to the class-name. We have designed a graphical user interface that user either input query statement directly using the above syntax or through the class directory provided by the interface. A user of the interface is assumed to have common sense of the referenced domain. For example, when querying the domain of bibliography, user should be aware of the concepts of books, articles, etc., and the associated properties, for example, title and author of book. Having the interested class (or concept) and the constraints in mind, user forms a query statement according to the above syntax. When the interface gets a query statement, it first checks syntax and then, if it is valid, converts the statement into a number of templates for retrieving entities in the triple statements. The templates are Prolog clauses that access through the ODBC interface the triples stored in the relational database. On the other hand, if user just knows little about the domain in question, she can choose the second approach. By entering a class (or concept), the interface responses with a list of properties associated with the class (or concept) for user's convenience to choose the required constraints. After the input is finished, the interface does the same task of the former approach.

In Section 2, we describe the background knowledge related to this research. In Sec-

tion 3, we describe the design of an RDF triple store. In Sections 4, we describe the design of a conceptual search program. In Section 5, we describe the implementation result. Finally conclusions are made.

2 Related Background

RDF and RDFS

RDF (Resource Description Framework) is a foundation for processing metadata and provides the basis of interoperability between applications that exchange machine-understandable information. The RDF specification consists of a data model and a syntax in XML. The RDF data model is a directed graph, where the nodes in the graph represent either resources in the Web or literal values and the label of an edge represents the association between the corresponding pairs of nodes. For example, in Figure 1, the ovals represent resources and rectangles are literal values.

A edge connecting two nodes represents a statement or a (subject, predicate, object) triple. For example, the diagrams in Figure 1 represent statements as shown in the following tables,respectively.

Subject:	http://www.cse.ttu.edu.tw/chingyeh
Predicate:	Creator
Object:	Ching-Long Yeh

Subject:	http://www.cse.ttu.edu.tw/chingyeh
Predicate:	Creator
Object:	*some_id*
Subject:	*some_id*
Predicate:	name
Object:	Ching-Long Yeh
Subject:	*some_id*
Predicate:	email
Object:	chingyeh@cse.ttu.edu.tw

RDF provides reference to a collection of entities in a container of order or unordered

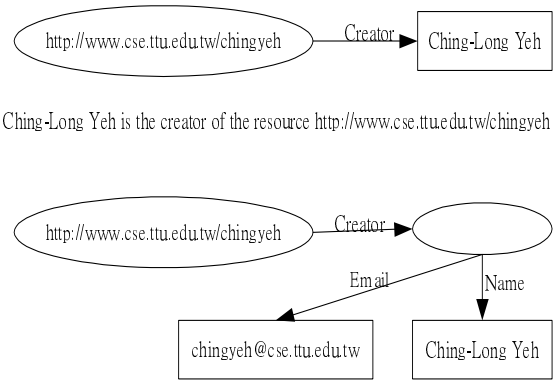

Figure 1: RDF diagrams with simple and structured values

list of objects, or an alternative list of objects. In addition to making statements about web resources, RDF can be used to make a statement about a statement.

The RDF data models described above can be encoded in an XML syntax. For example, the first diagram shown in Figure ẽfrdfDataModel can be encoded in XML as shown below. In the example, we omit XML declarations and namespaces for convenience. The same data model can be encoded in different XML forms. See [1] for the complete formal grammar and more detailed examples.

```
<rdf:RDF>
  <rdf:Description about="http://www.cse.ttu.edu.tw/chingyeh">
    <s:Creator>Ching-Long Yeh</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

RDF defines models for describing the relationship among resources and values. The relationships are represented in terms of

property names. The data model itself provides no mechanism for describing the properties. RDF Schema defines a set of vocabularies describing classes of resources and relationships between resources. It provides a basic type system for use in RD models. The RDFS type system contains primitive classes like Resource, Class, Property, ConstraintProperty and Literal and properties like subClassOf, subPropertyOf, domain, and range. Each class is described by certain properties . As shown below is a sample RDFS that defines a Person class along with three properties.

```
<rdf:RDF>
  <rdfs:Class rdf:ID="Person">
  <rdfs:comment>The class of people.</rdfs:comment>
  <rdfs:subClassOf rdf:resource=
      "http://www.w3.org/2000/03/example/classes#Animal" />
</rdfs:Class>
<rdfs:Property rdf:ID="maritalStatus">
  <rdfs:range rdf:resource="MaritalStatus" />
  <rdfs:domain rdf:resource="#Person" />
</rdfs:Property>
<rdfs:Property rdf:ID="socialSecurityNumber">
  <rdfs:range rdf:resource=
      "http://www.w3.org/2000/03/example/classes#Integer" />
  <rdfs:domain rdf:resource="#Person" />
</rdfs:Property>
<rdfs:Property rdf:ID="age">
  <rdfs:range rdf:resource=
      "http://www.w3.org/2000/03/example/classes#Integer" />
  <rdfs:domain rdf:resource="#Person" />
</rdfs:Property>
</rdf:RDF>
```

RDF triple stores and query

To make use of RDF content, it is necessary to provide indexing mechanism for accessing RDF document. W3C has provided a survey about various RDF triple data stores [11]. Relational database technology is the common basis for most of the data stores. SQL-like languages are provided to access the triples in data store. In addition to SQL query, there are other research of RDF query using RDF syntax or logic-based language [12].

RDF query

3 Building RDF Triple Store

As described in Section ?refsurvey, an RDF document is represented in XML. Some indexing mechanism must be provided to facilitate the access of RDF content. In this section, we describe the indexing mechanism that transforms an RDF document into the corresponding sequence of triple statements and then stores the triples in a relational database.

The indexing subsystem can be divided into two components. First is an RDF processor that takes RDF documents as input and produces the corresponding triple statements. Thus the RDF processor consists of a analysis and a generation part. In this paper, we choose Definite Clause Grammar (DCG) of Prolog to implement the analysis part. DCG is a notational extension of Prolog, that implements context-free grammar. Grammar rules represented in DCG can be executed directly by Prolog as a syntax analyzer [5]. The left-hand side of a DCG rule is a single term and the right-hand side consists of a sequence of terms, which correspond to both sides of a grammar rule, respectively. Within the RHS, a comma or semicolon is inserted between two terms to indicate the logical `and` and `or` relations, respectively. The sequence of terms in the RHS corresponds to the structure of the term in the LHS. Semantic actions, that are Prolog execution calls, can be inserted appropriately in the RHS to perform the generation tasks. In brief, the resulting RDF processor is a DCG program. In RDF Specification there are grammar rules for RDF

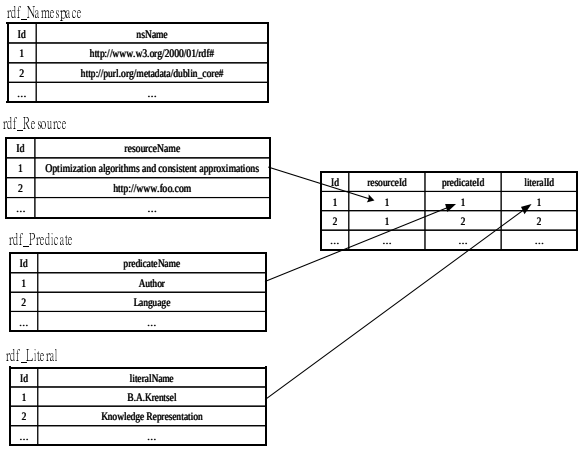

Figure 2: Tables for storing triples

in XML [1]. The construction of RDF processor is therefore representing each of the grammar rule in DCG with appropriate semantic actions inserted in the RHS. For example, the DCG rule as show below corresponding to Rule [6.2] in RDF Specification [1].

```
obj(Obj) -->
 container(Obj);
 description(_),{getAllTriples(Obj)}.
```

The second component takes the resulting triples as input and stores in five relational database tables, summarized in Figure 2.

After stored in relational database, user can access the content using SQL query or Prolog statements along with the ODBC interface. However, this approach is not convenient for user to access the content in RDF documents. In practice, we need some higher-level interface, that is, conceptual search as described later in Section ñefConceptual. In summary, the schematic

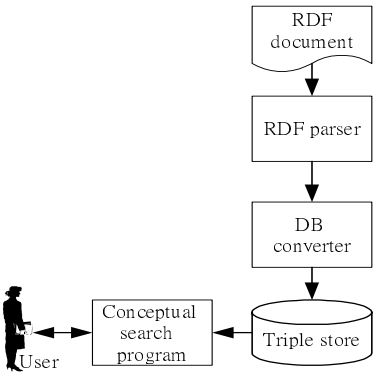

Figure 3: RDF triple store and high-level interface

diagram of the RDF triple store and the high-level interface is shown in Figure 3

4 Conceptual Search Interface

In the previous section, we have described how to index RDF documents using relational database technology. In this section, we describe the design of a high-level interface to facilitate user accessing the RDF content.

From user's point of view, an RDF document consists of a number of resources each of which belongs to some class, and the associated arcs along with the connecting targets forming the property-value set of the resources. The higher-level interface therefore supports class-based query, where user specify the class having certain property-value as the constraints. As mentioned previously, RDFS is an RDF vocabulary description language that defines properties of

classes and inheritance of classes and properties. The facts defined in an RDFS provide the basis for inference in conceptual search. To make use of the facts defined in the RDFS, we first of all encode the facts using the representation of an inference mechanism. In this paper, we employ Prolog as the inference mechanism of the conceptual search interface. Therefore we define the following Prolog statements to represent the facts in an RDFS.

- `class(ClassName).` This statement is to define the classes occurring in an RDFS, where `ClassName` is the name of a class.
- `property(PropertyName, [Domain, Range]).` There are two arguments of a property declaration. The first is the property name and the second is a list that contains the identifiers of domain and range of that property.
- `subClassOf(ParentClassName, SubclassName).` This declaration defines the relationship of parent and child classes.
- `subPropertyOf(ParentPropertyName, SubpropertyName, [Domain, Range]).` This is similar to `property` except that the parent property name is inserted as the first argument.

Having defined the class schema in Prolog, we then describe the conceptual search program. The conceptual search program provides a simple syntax for user to express query command. In general, a query command is composed three parts: the target class name, the constraint of the objects of the class, the required properties of the matched objects. The constraint part is a list of attribute-value pairs. The required properties is a list of property names specifying the resulting information of the matched objects. Before developing the inference rules for conceptual search, first of all we need to sort out the problem of the lexical mismatch between words having the same meaning. User may not know the identifiers of classes and properties occurring in the schema of ontology. The problem of lexical unmatch may happen between user's query command and the identifiers in the schema, which may lower the rate of successfully getting result. Thus employing a dictionary having semantic information with each lexical entry seems desirable to sort out the problem. However it is out of the scope of this research to build up such a dictionary. We employ a thesaurus dictionary for the moment to deal with the problem of lexical unmatch.

The conceptual search program as outlined below consists of a number of inference based on the class hierarchy in schema.

Input: A target class name, C; a list of attribute-value pairs as the constraints, $[a_1 : v_1, ..., a_n : v_n]$; and a list of attributes as the resulting fields, $[f_1, f_2, ..., f_m]$.

Output: A list of matched objects with the requested properties

Step 1: For the class name C and each of the attribute names, a_i, $i = 1, ..., n$, resolve the lexical unmatches by consulting the thesaurus dictionary. Let the results be C' and a'_i,

$i = 1, ..., n$.

Step 2: Find all objects belonging to class C', resulting a list of $Obj = [o_1, ..., o_l]$.

Step 3: For each object o_i in Obj, check if it satisfies each of the $a'_j : v_j$ in the constraint list. If it satisfies, then output o_i with the values of the requested properties.

Step 4: Find the subclasses of C'. For each subclass repeat Steps 2 and 3.

Step 5: Find the sibling classes of C'. For each sibling class repeat Steps 2 and 3.

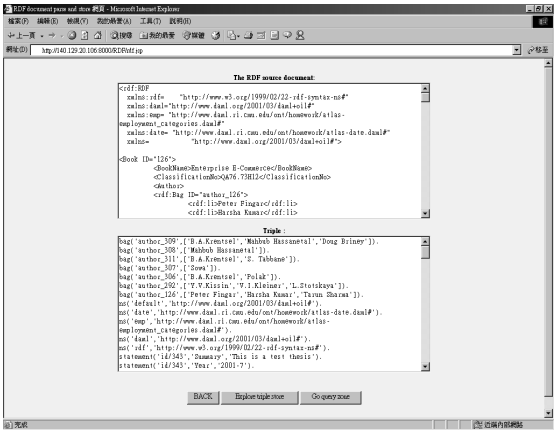

Figure 4: Result of parsing RDF document

5 Implementation

We have implemented all the components shown in Figure 3 using Sictus Prolog [8] and Microsoft SQL 2000 on MS Windows 2000. The interface between Prolog and browser is implemented using JSP technology. We choose a bibliography ontology from the DAML ontology library [9] as the schema and create RDF documents using Ontoedit [10], an ontology editor. User can send an RDF document to the parser to see the resulting triples in Prolog and further view the results stored in SQL database, as shown in Figure 4. In the conceptual search interface, user can either submit a query command, as shown in Figure 5 using the syntax described previously or using the schema provided by the interface, as shown in Figure 6.

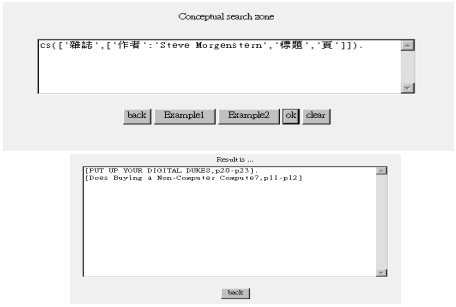

Figure 5: Conceptual search using command

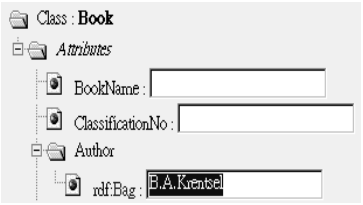

Figure 6: Conceptual search using schema

6 Conclusions

In this paper we describe the design and implementation of RDF triple store. We have finished the components to build a RDF triple store and a conceptual search program. The programs have been tested using a bibliography domain. The result of experiment is promising. In the future we will further improve the inference capability of the conceptual search and tested the programs with other domains. Prolog provides a good mechanism for inference. We will further add frame-based capability to Prolog to enhance the inference. This research focuses on the development of RDF indexing mechanism. We will develop other applications, such as semantic navigation, personalization, *etc.*

References

[1] W3C, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999. Available at http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[2] W3C, RDF Vocabulary Description Language 1.0: RDF Schema W3C Working Draft 30 April 2002. Available at http://www.w3.org/TR/rdf-schema/

[3] DAML+OIL, Available at http://www.daml.org/2001/03/daml+oil

[4] http://oiled.man.ac.uk/

[5] Ivan Bratko, *Prolog Programming for Artificial Intelligence*, 3rd ed., Addison-Wesley, 2001.

[6] G. Denker, et al., Accessing information and services on the DAML-enabled Web, *Proceedings of Second Int'l Workshop Semantic Web (SemWeb'2001)*, 2001. Available at http://citeseer.nj.nec.com/denker01accessing.html

[7] M. Klein and A. Bernstein, Searching for services on the semantic web using process ontologies, *Proceedings of International Semantic Web Working Symposium (SWWS)*, 2001.

[8] http://www.sics.se/sicstus/

[9] DAML Ontology Library, Available at http://www.daml.org/ontologies/

[10] Ontoedit, Available at http://www.ontoprise.de/com/start_downlo.htm

[11] W3C, Survey of RDF/Triple Data Stores, http://www.w3.org/2001/05/rdf-ds/DataStore

[12] G. Karvounarakis, RDF query languages: a state-of-the-art, http://139.91.183.30:9090/RDF/publications/state.h