The RDFSuite: Managing Voluminous RDF Description Bases

Sofia Alexaki Vassilis Christophides Greg Karvounarakis Dimitris Plexousakis

Institute of Computer Science,

FORTH, Vassilika Vouton, P.O.Box 1385, GR 711 10, Heraklion, Greece

{alexaki, christop, gregkar, dp}@ics.forth.gr

Karsten Tolle

Johann Wolfgang Goethe-University, Robert-Mayer-Str. 11-15,

P.O.Box 11 19 32, D-60054 Frankfurt/Main, Germany

tolle@dbis.informatik.uni-frankfurt.de

Abstract

Metadata are widely used in order to fully exploit information resources available on corporate intranets or the Internet. The Resource Description Framework (RDF) aims at facilitating the creation and exchange of metadata as any other Web data. The growing number of available information resources and the proliferation of description services in various user communities, lead nowadays to large volumes of RDF metadata. Managing such RDF resource descriptions and schemas with existing low-level APIs and file-based implementations does not ensure fast deployment and easy maintenance of real-scale RDF applications. In this paper, we advocate the use of database technology to support declarative access, as well as, logical and physical independence for voluminous RDF description bases.

We present RDFSuite, a suite of tools for RDF validation, storage and querying. Specifically, we introduce a formal data model for RDF description bases created using multiple schemas. Compared to the current status of the W3C standard, our model relies on a complete set of validation constraints for the core RDF/S (without reification) and provides a richer type system including several basic types as well as union types. Next, we present the design of a persistent RDF Store (RSSDB) for loading resource descriptions in an object-relational DBMS by exploring the available RDF schema knowledge. Our approach preserves the flexibility of RDF in refining schemas and/or enriching descriptions at any time, whilst it ensures good performance for storing and querying voluminous RDF descriptions. Last, we briefly present RQL, a declarative language for querying both RDF descriptions and schemas, and sketch query evaluation on top of RSSDB.

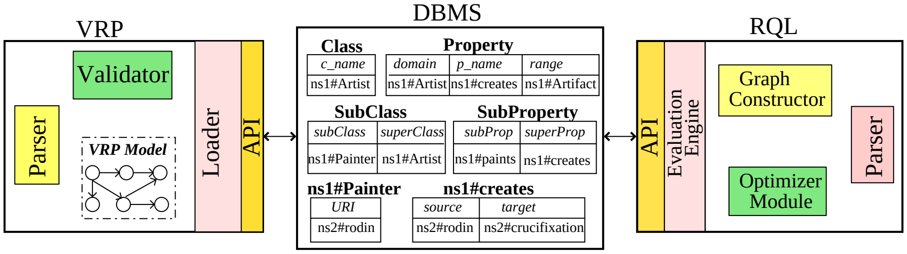
Keywords: RDF Formal Models, Storage Systems, Declarative Query Languages, Tools for the Semantic Web, Portal Applications.

1 Introduction

Metadata are widely used in order to fully exploit information resources (e.g., sites, documents, data, images, etc.) available on corporate intranets or the Internet. The Resource Description Framework (RDF) [15] aims at facilitating the creation and exchange of metadata as any other Web data. More precisely, RDF provides i) a *Standard Representation Language* for metadata based on *directed labeled graphs* in which nodes are called *resources* (or *literals*) and edges are called *properties* and ii) an *XML syntax* for expressing metadata in a form that is both humanly readable and machine understandable. Due to its flexible model, RDF is playing a central role in the next evolution step of the Web - termed the *Semantic Web*. Indeed, RDF/S enable the provision to different target communities (corporate, inter-enterprise, e-marketplace, etc.) of various kinds of metadata (for administration, recommendation, content rating, site maps, push channels, etc.) about resources of quite diverse nature (ranging from PDF or Word documents, e-mail or audio/video files to HTML pages or XML data). To interpret resource descriptions within or across communities, RDF allows for the definition of schemas [4] i.e., vocabularies of labels for graph nodes (i.e., *classes*) and edges (i.e., *properties*) that can be used to describe and query *RDF description bases*. Furthermore, RDF schema vocabularies can be easily extended to meet the description needs of specific (sub-)communities (e.g., through specialization of both entity classes and properties) while preserving the autonomy of description services for each (sub-)community.

Many content providers (e.g., ABCNews, CNN, Time Inc.) and Web Portals (e.g., Open Directory, CNET, XMLTree[1]) or browsers (e.g., Netscape 6.0, W3C Amaya) already adopt RDF. In a nutshell, the growing number of available information resources and the proliferation of description services in various user communities, lead nowadays to large volumes of RDF metadata (e.g., the Open Directory Portal of Netscape export in RDF around 170M of Subject Topics and 700M of indexed URIs). It becomes evident that managing such voluminous RDF resource descriptions and schemas with existing low-level APIs and file-based implementations [19] does not ensure fast deployment and easy maintenance of real-scale RDF applications. Still we want to take benefit from three decades of research in database technology to support *declarative access* and *logical and physical independence* for *RDF description bases*. In this way, RDF applications have to specify in a high-level language only which resources need to be accessed, leaving the task of determining how to efficiently store or access them to the underlying RDF database engine.

[1] www.dmoz.org, home.cnet.com, www.xmltree.com



Class

c_name
ns1#Artist

Property

domain	p_name	range
ns1#Artist	ns1#creates	ns1#Artifact

SubClass

subClass	superClass
ns1#Painter	ns1#Artist

SubProperty

subProp	superProp
ns1#paints	ns1#creates

ns1#Painter

URI
ns2#rodin

ns1#creates

source	target
ns2#rodin	ns2#crucifixation

Figure 1: Overview of the ICS-FORTH RDFSuite

In this paper we present `ICS-FORTH RDFSuite`[2], a suite of tools for RDF validation (*Validating RDF Parser*-VRP), storage (*RDF Schema Specific DataBase*-RSSDB), and querying (*RDF Query Language*-RQL) using an object-relational DBMS (see Figure 1). To illustrate the functionality and the performance of `RDFSuite` we use as testbed the catalog of the Open Directory Portal exported in RDF (see Section 2). Then we make the following contributions:

- Section 3 introduces a formal data model for *description bases* created according to the RDF Model & Syntax and Schema specifications [15, 4]. The main challenge for this model is the representation of several interconnected RDF schemas, as well as the introduction of a graph instantiation mechanism permitting multiple classification of resources. Compared to the current status of the W3C standard, our graph model relies on a complete set of validation constraints for the core RDF/S (without reification) and provides a richer type system including several basic types as well as union types.
- Section 4 presents our persistent RDF Store (RSSDB) for loading resource descriptions in an object-relational DBMS (ORDBMS) by exploring the available RDF schema knowledge. Our approach preserves the flexibility of RDF in refining schemas and/or enriching descriptions at any time, whilst it ensures better performance for storing and querying voluminous RDF descriptions than those proposals using a monolithic table [18, 16, 5] to represent resource descriptions and schemas under the form of triples.
- Section 5 briefly presents a declarative language, called *RQL*, to query both RDF descriptions and related schemas. *RQL* is a typed language that follows a functional approach (á la OQL [7]) and supports *generalized path expressions* featuring variables on both labels for nodes (i.e., classes) and edges (i.e., properties). Finally, we describe the implementation of *RQL* on top of RSSDB and sketch how *RQL* queries are translated into SQL3 by the *RQL* interpreter in order to push as much as possible path expressions evaluation to the underlying ORDBMS.

Finally, Section 6 presents our conclusions and draws directions for further research.

[2] www.ics.forth.gr/proj/isst/RDF

2 The Open Directory Portal Catalog

Portals are nowadays becoming increasingly popular by enabling the development and maintenance of specific communities of interest (e.g., enterprise, professional, trading) [11] on corporate intranets or the Internet. Such *Community Web Portals* essentially provide the means to select, classify and access, in a semantically meaningful and ubiquitous way various information resources. Portals may be distinguished according to the breadth of the target community (corporate, inter-enterprise, e-marketplace, web, etc.), the complexity of information resources (e.g., sites, documents, data), as well as, the quality of the relationships aiming to establish with the community members (horizontal, vertical). In all cases, the key Portal component is the *Knowledge Catalog* holding descriptive information, i.e., *metadata*, about the community resources.

For instance, the catalog of Internet (or horizontal) Portals, like Yahoo![3] or Open Directory[4], mainly uses huge hierarchies of topics (e.g., ODP uses 16 hierarchies with 248236 topics) to semantically classify Web resources (e.g., ODP index 2,251,641 sites). Additionally, various administrative metadata (e.g., titles, mime-types, modification dates) of resources are usually created using a Dublin-Core like schema. Then, users can either navigate through the topics of the catalog to locate resources of interest or issue a full-text query on topic names and the URIs or the titles of described resources. In Section 5 we will illustrate how our query language can be used to provide a declarative access to the catalog content. In the sequel, we will illustrate how Portal Catalogs can be easily and effectivelly represented using RDF/S.

The middle part of Figure 2 depicts the two schemas employed by the Open Directory: the first is intended to capture the semantics of web resources while the second is intended for Portal administrators. The scope of the declarations is determined by the corresponding *namespace* definition of each schema, e.g., **`ns1`** (`www.dmoz.org/topics.rdfs`) and **`ns2`** (`www.oclc.com/dublincore.rdfs`). For simplicity, we will hereforth omit the namespaces as well as the paths from the root of the topic hierarchies (since topics have non-unique names) prefixing class and property names. In the ODP topics schema, we can see two out of the sixteen hierarchies, namely, *Regional* and *Recreation* whose topics are represented as RDF classes (see the RDF/S default namespaces in the upper part of Figure 2). Various semantic relationships exist between these classes either within a topic hierarchy (e.g., *subtopics*) or across hierarchies (e.g., *related* topics). The former, is represented using the RDF subclass

[3] www.yahoo.com

[4] www.dmoz.org

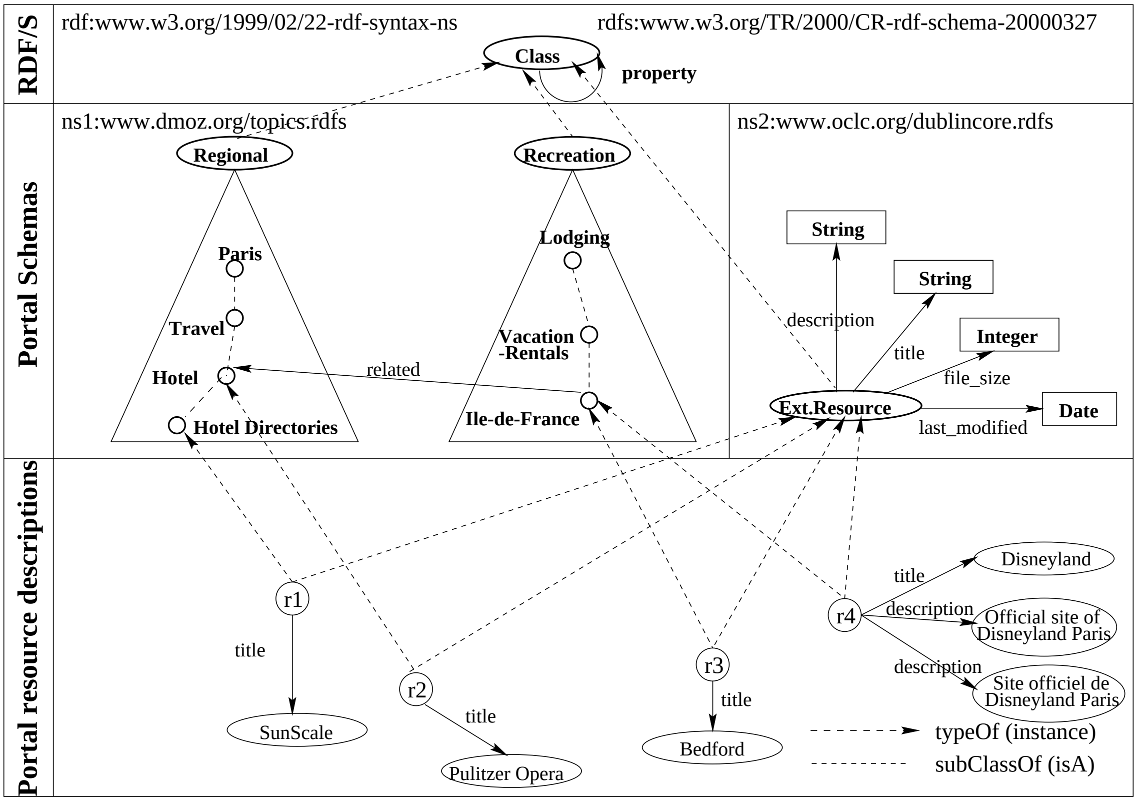


Figure 2: Modeling in RDF the Catalog of the Open Directory Portal

relationship (e.g., *Travel* is a, not necessarily direct, subclass of *Paris*) and the latter using an RDF property named *related* (e.g., between the classes *Ile-de-France* and *Hotel*). Finally, the ODP administrative metadata schema represents the various OCLC Dublin-Core [22] descriptive elements (with the execption of the *Subject* element), as literal RDF properties defined on class *ExtResource*. Note that properties serve to represent *attributes* of resources as well as *relationships* between resources. Properties can also be organized in taxonomies in a manner similar to the organization of classes.

Using these schemas, we can see in the lower part of Figure 2, the descriptions created for four sites (resources `&r1-&r4`). For instance, `&r4` is a resource classified under both the classes *Paris* and *ExtResource* and has three associated literal properties: a *title* property with value “Disneyland” and two *description* properties with values “Official site of Disneyland Paris” and “Site officiel de Disneyland Paris” respectively. In the RDF jargon, a specific resource (i.e., node) together with a named property (i.e., edge) and its value (i.e., node) form a *statement*. Each statement is represented by a *triple* having a *subject* (e.g., `&r4`), a *predicate* (e.g., `title`), and an *object* (e.g., “Disneyland ”). The subject and object should be of a type compatible (under class specialization) with the domain and range of the predicate (e.g., `&r4` is of type `ExtResource`). In the rest of the paper, the term *description base* will be used to denote a set of RDF statements. Although not illustrated in Figure 2, RDF also supports structured values called *containers* (i.e., bag, sequence) for grouping statements, as well as, higher-order statements (i.e., *reification*[5]). Finally, both RDF graph schemas and descriptions can be serialized in XML using various forests of XML trees (i.e., there is no root XML node).

We can observe that properties are *unordered* (e.g., the property *title* can be placed before or after the property *description*), *optional* (e.g., the property *file_size* is not used), can be *multi-valued* (e.g., we have two *description* properties), and they can be *inherited* (e.g., to subclasses of *ExtResource*), while resources can be multiply classified (e.g., `&r4`). These modeling primitives provide all the flexibility we need to represent heterogeneous descriptions of community resources (for different purposes), while preserving a conceptually unified view of the community's description base through one or the union of all related schemas. It becomes clear that the RDF data model differs substantially from standard (object or relational) models [2] or the recently proposed models for semistructured or XML databases [1]. Therefore existing systems cannot be used, as such, to manipulate voluminous RDF *description bases*. In the sequel, we will present a logical data model allowing users to issue high-level queries on RDF/S graphs while several physical representations of these graphs can be implemented by the underlying DBMS to improve storage volumes and optimize queries.

3 A Formal data model for RDF

Since the notion of resource is somehow overloaded in the RDF M&S and RDFS specifications [15, 4], we distinguish, **RDF resources** w.r.t. their *nature* into *individual entities* (i.e., *nodes*) and *properties* of entity resources (i.e., *edges*).

- **Nodes** : a set of individual resources, representing abstract or concrete entities of independent existence, e.g., class *ExtResource* defined in an RDF Schema or a specific web resource e.g., `&r4` (see Figure 2).
- **Edges** : a set of properties, representing both *attributes* of and binary *relationships* between nodes, either abstract or concrete, e.g., the property *title* defined in our example schema and used by the specific resource `&r4`.

RDF Resources are also distinguished according to their *concreteness* into *tokens* and *classes*.

- **Tokens** : a set of concrete resources, either objects, or literals (e.g., `&r4`, "SunScale").
- **Classes** : a set of abstract entity or property resources, in the sense that they collectively refer to a set of objects similar in some respect (e.g., *ExtResource*).

[5] We will not treat reification in this paper.

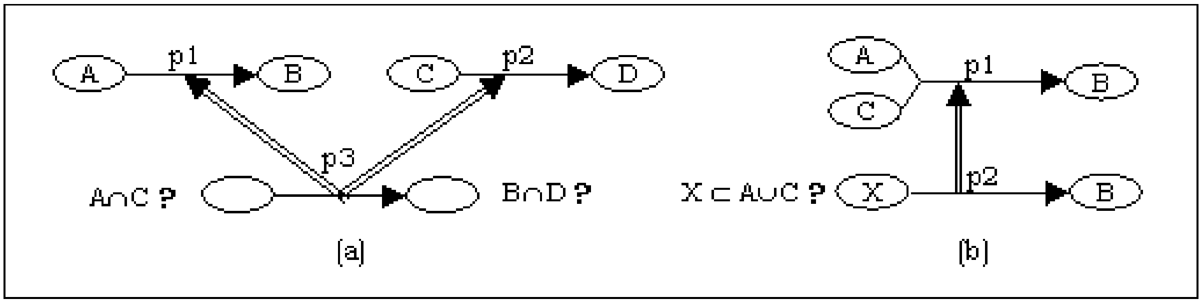


Figure 3: Semantic inconsistencies in specialization of properties

To label abstract (i.e., schema) or concrete (i.e., token) RDF nodes and edges, we assume the existence of the following countably infinite and pairwise disjoint sets of symbols: $\mathcal{C}$ of *Class names*, $\mathcal{P}$ of *Property names*, $\mathcal{U}$ of *Resource URIs* as well as a set $\mathcal{L}$ of *Literal type names* such as *string*, *integer*, *date*, etc. Each literal type $t \in \mathcal{L}$ has an associated domain, denoted $dom(t)$ and $dom(\mathcal{L})$ denotes $\bigcup_{t \in \mathcal{L}} dom(t)$ (i.e., the `rdfs:Literal` declaration). Without loss of generality, we assume that the sets $\mathcal{C}$ and $\mathcal{P}$ are extended to include as elements the class name `Class` and the property name `Property` respectively. The former captures the root of a class hierarchy (i.e., the `rdfs:Class` declaration) while the latter captures the root of a property hierarchy (i.e., the `rdf:Property` declaration) defined in RDF/S [15, 4]. The set $\mathcal{P}$ also contains integer labels ($\{\mathtt{1}, \mathtt{2}, \ldots\}$) used as property names (i.e., the `rdfs:ContainerMembershipProperties` declaration) by the members of container values (i.e., the `rdfs:Bag`, `rdfs:Sequence` declarations).

Each RDF schema uses a finite set of class names $C \subseteq \mathcal{C}$ and property names $P \subseteq \mathcal{P}$ whose scope is determined by one or more namespaces. Property types are then defined using class names or literal types so that: for each $p \in P$, $domain(p) \in C$ and $range(p) \in C \cup \mathcal{L}$. We denote by $H = (N, \prec)$ a hierarchy of class and property names, where $N = C \cup P$. H is *well-formed* if $\prec$ is a smallest partial ordering such that: if $p_1, p_2 \in P$ and $p_1 \prec p_2$, then $domain(p_1) \preceq domain(p_2)$ and $range(p_1) \preceq range(p_2)$[6].

Three remarks are noteworthy. First, unlike the current RDF M&S and RDFS specifications [15, 4] the *domain and range of properties should always be defined.* This additional constraint is required mainly because the sets of RDF/S classes $\mathcal{C}$ and literal types $\mathcal{L}$ are disjoint. Then, a property with undefined range may take as values both a class instance (i.e. a resource) or a literal. Since, the *union* of `rdfs:Class` and `rdfs:Literal` is meaningless in RDF/S, this freedom leads to semantic inconsistencies. Additional semantic problems arise in the case of specialization of properties with undefined domains and ranges. More precisely,

[6] The symbol $\preceq$ extends $\prec$ with equality.

Alphabets:	**C1**	$\mathcal{C} \cap \mathcal{P} \cap T = \emptyset$
	C2	$\mathcal{L} \cap \mathcal{U} = \emptyset$
Schema:	**C3**	$\forall p \in P, \exists! c_1 \in \mathcal{C}\ (c_1 = domain(p)) \wedge \exists! c_2 \in \mathcal{C} \cup T_L\ (c_2 = range(p))$
	C4	$\forall c, c', c'' \in \mathcal{C}$:
	C4.0	• $c \prec Class$
	C4.1	• $c \prec c', c' \prec c'' \Rightarrow c \prec c''$
	C4.2	• $c \prec c' \Rightarrow c \neq c'$
	C5	$\forall p, p', p'' \in \mathcal{P}$:
	C5.0	• $p \prec Property$
	C5.1	• $p \prec p', p' \prec p'' \Rightarrow p \prec p''$
	C5.2	• $p \prec p' \Rightarrow p \neq p''$
	C5.3	• $p \prec p' \Rightarrow domain(p) \preceq domain(p') \wedge range(p) \preceq range(p')$
	C5.4	• $p \prec p' \wedge p \prec p'' \Rightarrow \quad domain(p) \preceq domain(p') \wedge domain(p) \preceq domain(p'')$ $\wedge range(p) \preceq range(p') \wedge range(p) \preceq range(p'')$
Data:	**C6**	$\forall v \in \mathcal{U} \Rightarrow \lambda(v) \in \mathcal{C} \cup T$
	C7	$\forall p \in \mathcal{P} \cup \{1, 2, \ldots\}, [v_1, v_2] \in semp$:
	C7.1	• if $p \in \{1, 2, \ldots\} \Rightarrow \lambda(v_1) \in \{$rdf:Bag, rdf:Alt, rdf:Seq$\}$
	C7.2	• if $p \in \mathcal{P} \Rightarrow \lambda(v_1) \preceq domain(p), \lambda(v_2) \preceq range(p)$

Table 1: Formal definition of imposed constraints

to preserve the set inclusion requirement of specialized properties (binary predicates) their domain and range should also specialize the domain and range (unary predicates) of their superproperties. This is something which, in the case of multiple specialization of properties (see Figure 3 -a-), cannot always be ensured because RDF/S do not support *intersection* of classes (in a Description Logics style). The second constraint imposes that *both domain and range declarations of properties should be unique.* This is foremost required because RDF/S do not support *union* of classes, which can be considered as the domain of properties. Furthermore, it is not possible to infer domains in case of specialization of properties with multiple domains (see Figure 3 -b-).

Last, we need to consider an additional constraint of a syntactic nature imposing that *class and property definitions should be complete.* This means that, on the one hand superclass and superproperty definitions should accompany the class and property definitions respectively and, on the other, the definition of the domain and range of a property should be given along with that of the property. In this manner, the extension of existing RDF hierarchies of names by refining their classes and properties in a new namespace is still permitted, while at the same time semantic inconsistencies that may arise due to arbitrary unions of defined hierarchies are avoided. Such inconsistencies include the introduction of multiple ranges of properties or the introduction of cycles in class or property hierarchies. Unlike the current RDF M&S and RDFS specifications [15, 4] this constraint ensures that the *union of two valid RDF hierarchies of names is always valid.* The imposed constraints are summarized in

Table 1 using the notation introduced in this section.

In this context, RDF data can be atomic values (e.g., strings), resource URIs, and container values holding query results, namely *rdf:Bag* (i.e., multi-sets) and `rdf:Sequence` (i.e., lists). More precisely, the main types foreseen by our model are:

$$\tau = \tau_L \mid \tau_U \mid \{\tau\} \mid [\tau] \mid (1:\tau + 2:\tau + \ldots + n:\tau)$$

where τ_L is a literal type in $\mathcal{L}$, $\{.\}$ is the *Bag* type, $[.]$ is the *Seq*uence type, $(.)$ is the *Alt*ernative type, and τ_U is the type for resource *URI*s [7]. Alternatives capture the semantics of union (or variant) types [6], and they are also ordered (i.e., integer labels play the role of union member markers). Since there exists a predefined ordering of labels for sequences and alternatives, labels can be omitted (for bags, labels are meaningless). Furthermore, all types are mutually exclusive (e.g., a literal value cannot also be a bag) and no subtyping relation is defined in RDF/S (e.g., between bags of different types). The set of all type names is denoted by T.

It should be stressed that the main subtleties for typing RDF/S graphs are related to the following facts:

- *Classes do not define object or relation types*: an instance of a class is just a resource URI without any value/state;
- *Resources may belong to different classes* not necessarily pairwise related by specialization: the instances of a class may have quite different properties associated with them while there is no other class on which the union of these properties is defined;

The proposed type system offers all the arsenal we need to capture containers with both homogeneous and heterogeneous member types, as well as, to interpret RDF schema classes and properties. For instance, *unnamed ordered tuples* denoted by $[v_1, v_2, \ldots, v_n]$ (where v_i is of some type τ_i) can be defined as heterogeneous sequences[8] of type $[(\tau_1 + \tau_2 + \ldots + \tau_n)]$. Hence, RDF classes can be seen as unary relations of the type $\{\tau_U\}$ while properties as binary relations of type $\{[\tau_U, \tau_U]\}$ (for relationships) or $\{[\tau_U, \tau_L]\}$ (for attributes). As we will see in Section 5, RDF containers can be also used to represent n-ary relations (e.g., as a bag of sequences). Finally, assignment of a finite set of URIs (of type τ_U) to each class name[9] is captured by a *population function* $\pi : C \rightarrow 2^U$. The set of all values forseen by our model is denoted by V.

[7] In Section 5, we will see that our query language treats URIs, i.e., identifiers, as simple strings.

[8] Observe that, since tuples are ordered, for any two permutations $i_1, \ldots, i_n$ and $j_1, \ldots, j_n$ of $1, \ldots, n$, $[i_1 : v_1, \ldots, i_n : v_n]$ is distinct from $[j_1 : v_1, \ldots, j_n : v_n]$.

[9] Note that we consider here a non-disjoint object id assignment to classes due to multiple classification.

Definition 1 *The interpretation function* $[\![.]\!]$ *is defined as follows:*

- *for literal types:* $[\![\mathcal{L}]\!] = dom(\mathcal{L})$;
- *for the Bag type,* $[\![\{\tau\}]\!] = \{v_1, v_2, \ldots v_n\}$ *where* $v_1, v_2, \ldots v_n \in V$ *are values of type* τ;
- *for the Seq type,* $[\![[\tau]]\!] = [v_1, v_2, \ldots v_n]$ *where* $v_1, v_2, \ldots v_n \in V$ *are values of type* τ;
- *for the Alt type* $[\![(\tau_1 + \tau_2 + \ldots + \tau_n)]\!] = v_i$ *where* $v_i \in V$ $1 < i < n$ *is a value of type* τ_i;
- *for each class* $c \in C$, $[\![c]\!] = \pi(c) \cup \bigcup_{c' \prec c} [\![c']\!]$;
- *for each property* $p \in P$, $[\![p]\!] = \{[v_1, v_2] \mid v_1 \in [\![domain(p)]\!], v_2 \in [\![range(p)]\!]\} \cup \bigcup_{p' \prec p} [\![p']\!]$.

Definition 2 *An RDF schema is a 5-tuple* $RS = (V_S, E_S, \psi, \lambda, H)$, *where:* V_S *is the set of nodes and* E_S *is the set of edges,* H *is a well-formed hierarchy of class and property names* $H = (N, \prec)$, λ *is a labeling function* $\lambda : V_S \cup E_S \to N \cup T$, *and* ψ *is an incidence function* $\psi : E_S \to V_S \times V_S$.

The *incidence function* captures the `rdfs:domain` and `rdfs:range` declarations of properties[10]. Note that the incidence and labeling functions are *total* in $V_S \cup E_S$ and E_S respectively. This does not exclude the case of schema nodes which are not connected through an edge.

Definition 3 *An RDF description base, instance of a schema RS, is a 5-tuple* $RD = (V_D, E_D, \psi, \nu, \lambda)$, *where:* V_D *is a set of nodes and* E_D *is a set of edges,* ψ *is the incidence function* $\psi : E_D \to V_D \times V_D$, ν *is a value function* $\nu : V_D \to V$, *and* λ *is a labeling function* $\lambda : V_D \cup E_D \to 2^{N \cup T}$ *which satisfies the following:*

- *for each node* v *in* V_D, λ *returns a set of names* $n \in C \cup T$ *where the value of* v *belongs to the interpretation of each* n: $\nu(v) \in [\![n]\!]$;
- *for each edge* ϵ *in* E_D *going from node* v *to node* v', λ *returns a property name* $p \in P$, *such that:*
  - *if* $p \in P \setminus \{1, 2, \ldots\}$, *the values of* v *and* v' *belong respectively to the interpretation of the domain and range of* p: $\nu(v) \in [\![domain(p)]\!]$, $\nu(v') \in [\![range(p)]\!]$;
  - *if* $p \in \{1, 2, \ldots\}$, *the values of* v *and* v' *belong respectively to the interpretation of a Bag|Seq|Alt type and their corresponding member types:* $\nu(v) \in [\![Bag|Seq|Alt(\tau)]\!]$, $\nu(v') \in [\![\tau]\!]$.

Note that the *labeling function* captures the `rdf:type` declaration that associates each RDF node with one or more class names. It should be stressed that our model captures the majority of RDF/S modeling primitives with the exception of property *reification* given that it is not expressible in RDFS. Finally, built-in property types such as `rdfs:seeAlso`, `rdfs:isDefinedBy`, `rdfs:comment`, `rdfs:label` can be easily represented in our model, but due to space limitations they are not considered in this paper.

[10] Constraint **C3** of Table 1 ensures that `rdfs:domain` and `rdfs:range` are not any more relations as in the current RDF M&S and RDFS specifications [15, 4].

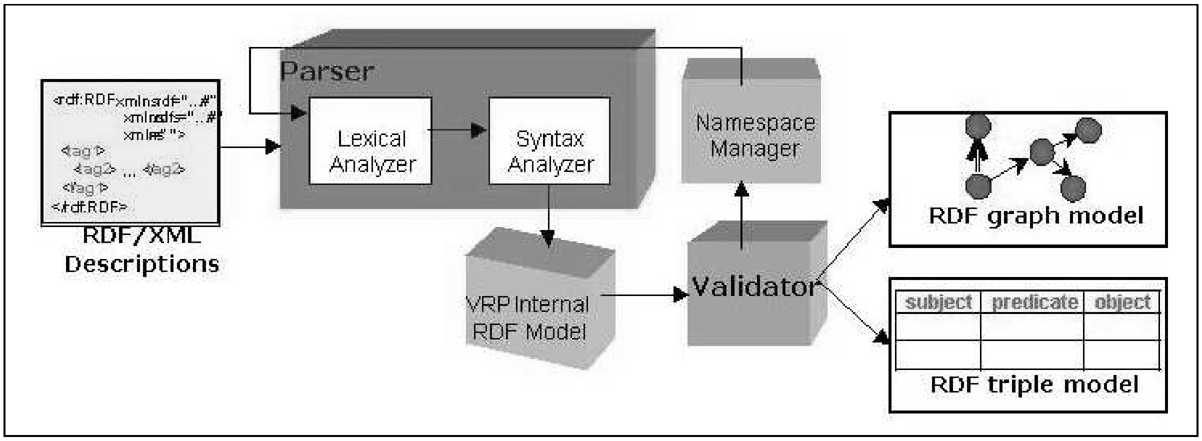


Figure 4: The Validating RDF Parser (VRP)

3.1 The Validating RDF Parser (VRP)

To conclude this section, we briefly describe the *Validating RDF Parser* (VRP) we have implemented to analyze, validate and process RDF descriptions. Unlike other RDF parsers (e.g., SiRPAC[11]), VRP (see Figure 4) is based on standard compiler generator tools for Java, namely CUP/JFlex (similar to YACC/LEX). As a result, users do not need to install additional programs (e.g., XML Parsers) in order to run VRP. The VRP BNF grammar can be easily extended or updated in case of changes in the RDF/S specifications. VRP is a 100% Java(tm) development understanding embedded RDF in HTML or XML and providing full Unicode support. The stream-based parsing support of JFlex and the quick LALR grammar parsing of CUP ensure good performance when processing large volumes of RDF descriptions. Currently VRP is a command line tool with various options to generate a textual representation of the internal model (either graph or triple based).

The most distinctive feature of VRP is its ability to verify the constraints specified in the RDF M&S and RDFS specifications [15, 4] as well as the additional constraints we introduced previously (see Table 1). This permits the validation of both the RDF descriptions against one or more RDFS schemas, and the schemas themselves. The VRP validation module relies on (a) a complete and sound algorithm [21] to translate descriptions from an RDF/XML form (using both the Basic and Compact serialization syntax) into the RDF triple-based model (b) an internal object representation of this model in Java, allowing to separate RDF schema from data information. As we will see in the sequel, this approach enable a fine-grained processing of RDF statements w.r.t. their type which is crucial to implement an *incremental loading* of RDF descriptions and schemas.

[11] www.w3.org/RDF/Implementations/SiRPAC

4 The RDF Schema Specific DataBase-RSSDB

This section describes the persistent RDF store (RSSDB) for loading resource descriptions in an object-relational DBMS. We begin by presenting our schema generation strategy.

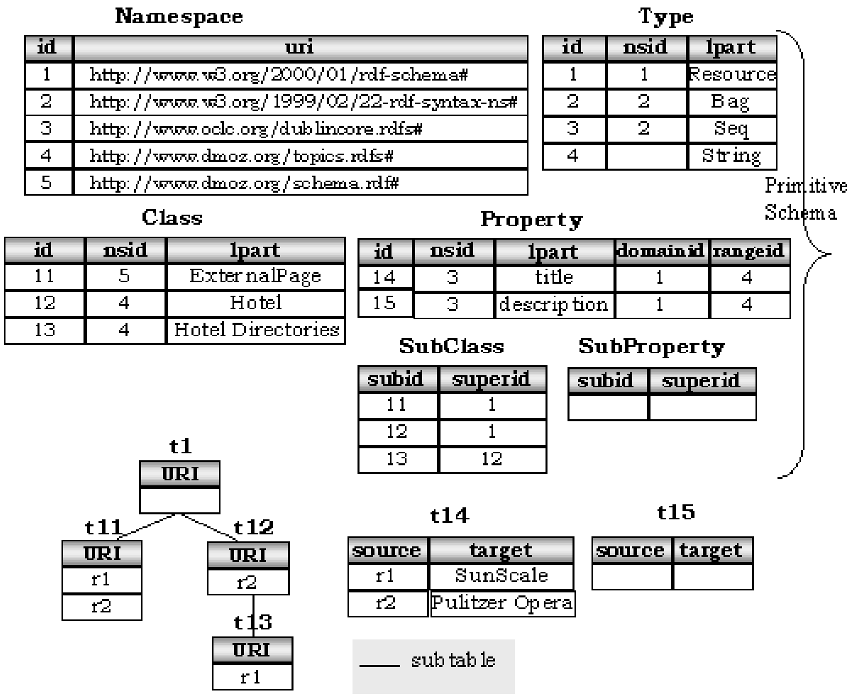
The core RDF/S model is represented by four tables (see Figure 5), namely, `Class`, `Property`, `SubClass` and `SubProperty` which capture the class and property hierarchies defined in an RDF schema. Table `NameSpace` keeps the namespaces of the RDF Schemas stored in the DBMS and is mainly used to save space when storing class and properties names. Table `Type` keeps the class names defined in RDF/S and Literal type names for all schemas. Subsequently, for every class or property used in a Portal Catalog, a new table is created to store its instances (recall that all names are unique). Specificallly, class tables store the URIs of resources while property tables store the URIs of the source and target nodes of the property. Indices are constructed on the attributes `URI`, `source` and `target` of the above tables in order to speed up joins and the selection of specific tuples of the tables. Indices are also constructed on the attributes `lpart` and `nsid` of the table `Class` and on the attribute `subid` of the tables `SubClass` and `SubProperty`. In other words, our RDF-enabled DBMS relies on a schema specific representation of resource descriptions similar to the *attribute-based* approach proposed for storing XML data [12, 20].

Three remarks are noteworthy. First, unlike XML, RDF graphs contain labels for both graph nodes and edges. Therefore, we need to generate tables for both properties and class instances. Second, RDF labels may be organized in taxonomies through multiple specialization (as opposed to element types defined in XML DTDs or Schemas). This information is captured by the `SubClass` and `SubProperty` tables, while the corresponding instance tables are also connected through the *subtable* relationship, supported by an object-relational. Note that the syntactic constraint imposing complete class and property definitions, ensures that the table hierarchy created in RSSDB can be only extended through specialization. Third, there is no real need for expensive (due to data fragmentation in several tables) RQL queries, reconstructing the initially loaded resource descriptions - as in XML query languages - since there are various ways to serialize RDF graph descriptions in a forest of XML trees.

4.1 The RDF Description Loader

Figure 6 depicts the architecture of our system for loading RDF metadata in an object-relational DBMS, namely PostgreSQL[12]. The loader has been implemented [3] in Java and

[12] www.postgresql.org



Namespace

id	uri
1	http://www.w3.org/2000/01/rdf-schema#
2	http://www.w3.org/1999/02/22-rdf-syntax-ns#
3	http://www.oclc.org/dublincore.rdfs#
4	http://www.dmoz.org/topics.rdfs#
5	http://www.dmoz.org/schema.rdf#

Type

id	nsid	lpart
1	1	Resource
2	2	Bag
3	2	Seq
4		String

Class

id	nsid	lpart
11	5	ExternalPage
12	4	Hotel
13	4	Hotel Directories

Property

id	nsid	lpart	domainid	rangeid
14	3	title	1	4
15	3	description	1	4

SubClass

subid	superid
11	1
12	1
13	12

SubProperty

subid	superid

t1

URI

t11

URI
r1
r2

t12

URI
r2

t13

URI
r1

t14

source	target
r1	SunScale
r2	Pulitzer Opera

t15

source	target

Figure 5: Relational Representation of RDF Description Bases

communication between loader and PostgreSQL relies on the JDBC protocol.

The loader comprises two main modules. The first module checks the consistency of analyzed schemas descriptions in comparison with the stored information in the DBMS. For example, in case that an analyzed property has already been stored, it checks whether its domain and range are the same as the ones stored in the DBMS. Another functionality of this module is the validation of RDF metadata based on stored RDF schemas instead of connecting to the respective namespaces. Thus, we avoid analyzing and validating repeatedly the RDF schemas used in metadata and reduce the required main memory, since only parts of RDF schemata are fetched. Consequently, our system enables *incemental* loading of RDF descriptions and schemas, which is crucial for handling large RDF schemas and even larger RDF description volumes created using multiple schemas (e.g., Netscape Open Directory exports 100M of class hierarchies and 700M of resource descriptions).

The second module implements the loading of RDF descriptions in the DBMS. To this end, a number of loading methods have been implemented as member functions of the related VRP internal classes. Specifically, for every attribute of the classes of the VRP model, a method is created for storing the attribute of the created object in the DBMS. For example, the method `storetype` is defined for the class `RDF_Resource`, in order to store object type information. The primitive methods of each class are incorporated in a storage method defined

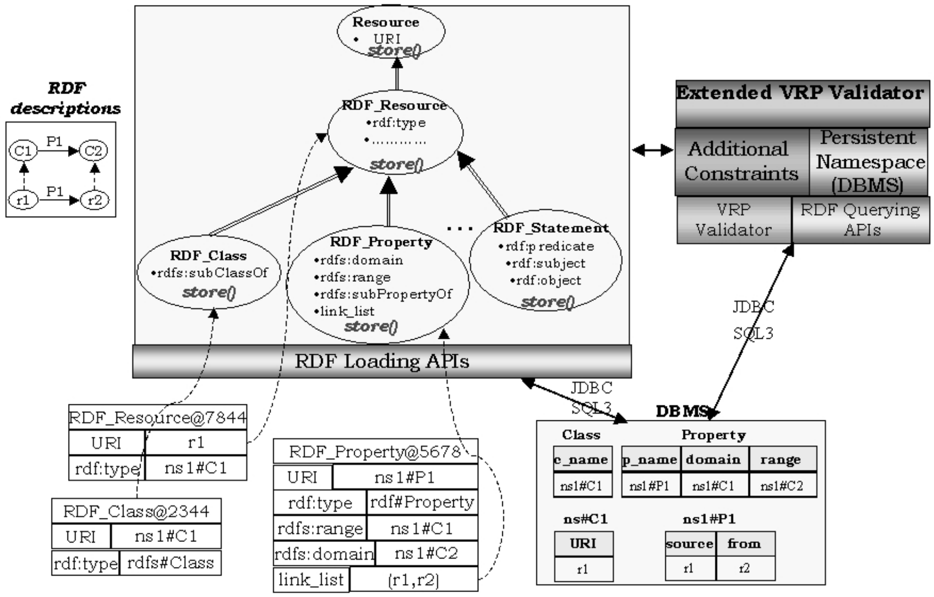


Figure 6: The RDF Schema Specific DataBase and Loader

in the respective class invoked during the loading process. A two-phase algorithm is used for loading the RDF descriptions. During the first phase, RDF class and properties are stored to create the corresponding schema. During the second phase the database is populated with resource descriptions.

4.2 Performance Tests

In order to evaluate the performance of our system, we used as testbed the RDF dump of the Open Directory Catalog consisting of 170 MBytes of class hierarchies and 700 MBytes of resource descriptions. Experiments have been carried out on a Sun ULTRA 60 machine with a 450MHz processor and 250 MBytes of main memory. The buffer size of the DBMS was set to 4 MBytes.

Initially, schema descriptions are loaded in the DBMS. In the sequel, data descriptions are loaded and indices on the tables containing the data descriptions are constructed. For each file, we measure the number of triples contained in it, the time required to load the triples and the resulting size of the DBMS. We take different measurements for data and schema descriptions. Indices are also constructed on the tables `Class` (or `Property`) and `SubClass` (or `SubProperty`) but their costs on storage space and time are not included in the measurements due to their minimal influence to the total DBMS size and loading time. Some preliminary performance results appear in the graphs of Figures 7 and 8.

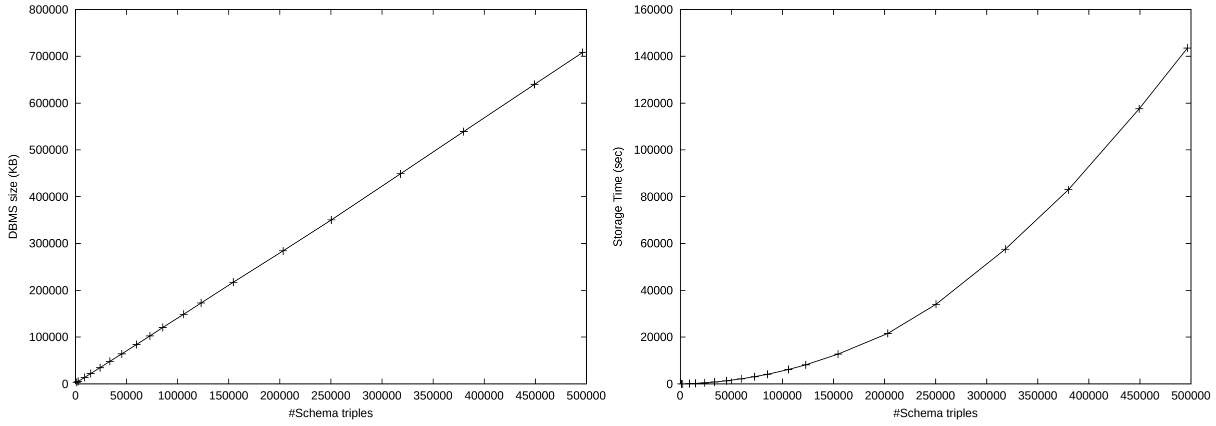


Figure 7: Statistics for loading schema descriptions

The first graph of Figure 7 depicts the size of the database in comparison with the number of the schema triples that are loaded. We observe that the size of the DBMS scales linearly with the number of triples. The tests show that the *database size is 14 times bigger* than that of the original file holding the descriptions. Specifically, 50 MBytes[13] of class hierarchies (248236 classes) have taken up 708 MBytes. This significant factor is mainly attributed to the information that the DBMS keeps for the tables. Approximately 65% of the total space (450 MBytes) is taken up for storing information about the attributes of the created tables. The second graph in Figure 7 shows the time required for storage in comparison with the number of the triples have been stored. As the number of stored triples increases, the time required for storing a triple also increases. This is due to the tests that the DBMS should carry out before creating a new table (e.g., to test if a table with the same name has already been created) or to the cost of updating indices created on the table storing the attributes of defined database tables. These tests demand more time as the database's schema grows. The *average time for loading a schema triple is about 0.175 sec.*

The respective graphs for the size of the database and the time required for storing resource descriptions are shown in Figure 8. The first graph depicts the size of the database before and after indices are constructed on the tables in comparison with the number of the data triples loaded. We observe that the shape of the graph is not linear. The space allocated for a table in the DBMS doesn't correspond to the total size of records contained in it due to fact that the DBMS does not allocate space every time a record is added. Instead, a specific amount of space is allocated when additional space is necessary to store a record in the table. The additional space allocated influences more the ratio (DBMS size)/(triples size) when the

[13] This is the volume of the pure ODP schema information producing when properties such as symbolic(1), related, altlang, newsGroup, editor, lastUpdate, catid, dc:Title and dc:Description attributed to the classes are removed.

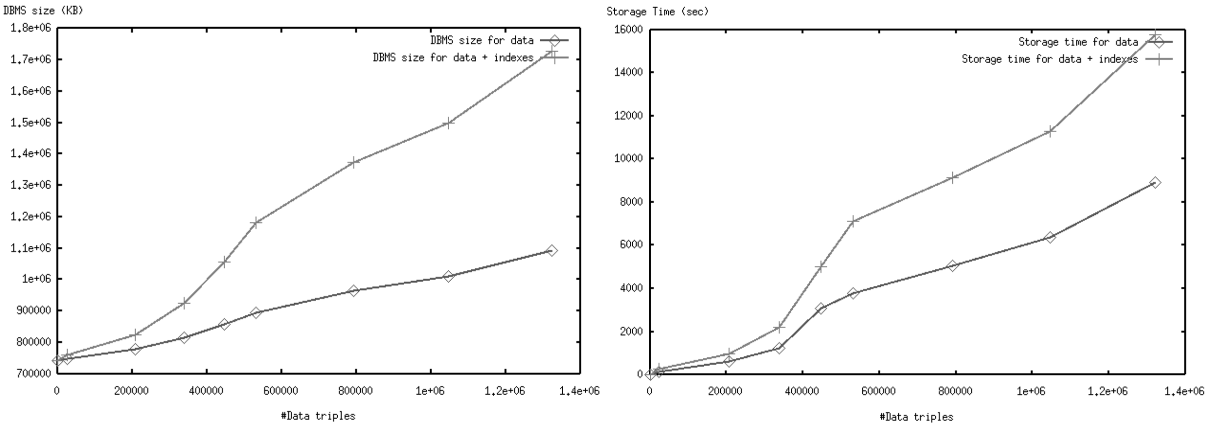


Figure 8: Statistics for loading resource descriptions

volume of the data descriptions stored in the tables is small. The tests show that the *database size is about 2 times bigger* than that of the file containing the descriptions. *When indices are constructed it becomes 6 times bigger.* As far as data triples are concerned, storage time (see the second graph in Figure 8) increases when the number of classes where URIs are classified increases, since more time is required for database's schema querying (i.e., for finding the tables). This additional time influences the total storage time especially when the average number of instances per class is small. The tests show that the *average time for loading a data triple is about 0.006 sec. When indices are constucted, the average time increases to 0.01sec.*

Compared to approaches using a unique table [18, 16, 5] to represent RDF descriptions and schemas in the form of triples, RSSDB requires almost the same database volume for storing (and indexing) RDF resource descriptions (i.e., twice the size of the original file) while RDF schema descriptions clearly consume more space (i.e., 14 times bigger). However, given the proportion of the loaded RDF schema (50 M) and data (700M), the average increasing storage factors are comparable (i.e., 2.8 in RSSDB vs. 2.1 in the triple-based approach). Of course, the definitive advantage of the RDF schema specific representation adopted by RSSDB is the obtained performance gain in query evaluation. As a matter of fact, by restricting the search space to specific class and property tables, we are able to optimize all basic RDF queries (e.g., find the instances of a class, a property, etc.). Due to space limitations extensive query results are not preseted in this paper.

The only limitation of the RSSDB representation is the large number of tables created for large RDF schemata. Given that some commercial DBMS systems (and not PostgreSQL) permit only a limited number of tables, this approach is not always applicable. Furthermore, it also increases the number of joins required to evaluate RQL path expressions (see Section 5).

Several variations are currently studied in order to reduce the number of created tables. In the first alternative representation, a property having as range the class `rdfs:Literal` (attribute-property) is represented as an attribute of the table created for the domain of this property. Consequently, new attributes are added to the class tables. The tables created for properties with range any class other than `rdfs:Literal` remain unchanged. The above representation is applicable to RDF schemas where attribute-properties are single-valued and not specialized.

We conclude this section with a second alternative to the basic RSSDB representation, allowing us to avoid the creation of tables for classes (or properties) with few or no instances at all. The members of those classes would be kept in a unique table `Instances` having the attributes `uri` and `classid` for keeping the uri's of the resources and the id's of the classes in which resources belong. Furthermore, the tables `SubClass` and `SubProperty` can be completely suppressed. For this, we need to replace class (or property) names by *ids* using an appropriate encoding system (e.g., Dewey, postfix, prefix, etc.) for which a convenient total order exists between the elements in the hierarchy (i.e., capturing the subclass or subproperty relationships). We are currently working on the choice of a such linear representation of node or edge labels allowing us to optimize queries that involve different kinds of traversals in a hierarchy (e.g., an entire subtree, a path from the root, etc.). From some preliminary tests, the gains with this representation in storage volumes and time required for recursive query evaluation seems quite promising.

5 Querying RDF Description Bases with RQL

As we have seen in the previous sections, the catalog of Portals like Netscape Open Directory comprises huge hierarchies of classes (248236 Subject topics) and an even bigger number of resource descriptions (2,251,641 indexed sites). It becomes evident that browsing such large *description bases* is a quite cumbersome and time-consuming task. Consider, for instance, that we are looking for information about hotels in Paris, under the topic *Regional* of the ODP (see Figure 2). ODP allows users to navigate through the topic hierarchy; even if one knows the exact path to follow, this would require approximately 10 steps, in order to reach the required topics (i.e. Hotels, Hotel Directories in Figure 2). Then, in order to find the URIs of the sites whose title or description matches the string "*Opera*", users are forced to manually browse the entire collection of resources directly classified under the topic of interest. Note that, in order to locate resources classified under the subtopics (e.g., *Hotel Directories*) of a given topic, browsing should be continued by the users. *RQL* aims to simplify such tasks, by providing powerful path expressions permitting smooth filtering/navigation on both Portal schemas and resource descriptions. Then the previous query can be expressed as follows:

Q1: *Find the resources under the hierarchy Regional, about hotels in Paris whose title matches ”*Opera*”.*

```
select  Z
from    (select  $X
         from    Regional{:$X}
         where   $X like ”*Hotel*” and $X < Paris){Y}.{Z}title{T}
where   T like ”*Opera*”
```

The *schema path expression* in the from clause of the nested query, states that we are interested in classes (variables prefixed by \$ like $\$X$) specializing the root Regional. Then, the filtering condition in the where clause will retain only the classes whose name matches ”*Hotel*” and they are also subclasses of *Paris* (e.g., *Hotel* and Hotel Directories. Here, to get all relevant topics, the only required schema knowledge is that the subtopics of Regional contains geographical information and a topic *Paris* is somewhere in the hierarchy. Thanks to RQL typing, variable Y is of type class name and ranges over the result of the nested query. The *data path expression* in the outer query iterates over the source (variable Z of type resource URI) and target values (variable T of type resource URI) of the *title* property. The “.” implies an implicit join condition between the extend of each class valuating Y and the resources valuating Z. Finally, the result of **Q1** will be a bag of resources whose title value matches ”*Opera*”. Obviously, *RQL schema queries* are far more powerful than the corresponding *topic queries* of common portals, which allow only full-text queries on the names of topics. Furthermore, compositions of *schema and data queries* like **Q1**, are not possible in current portals, since one cannot specify that some query terms should match the topic names and other should be found in the descriptions of the resources.

To make things more complex, relevant information in portals may also be found under different hierarchies, that may be “connected” through *related* links. For example, as we can see in Figure 2, information about hotels in Paris may also be found in the *Recreation* hierarchy. Moreover, such links are not necessarily bi-directional; thus, a user starting from the *Regional* hierarchy may never find out that similar information may be found under *Recreation* e.g., *Vacation-Rentals*. For such cases, *RQL* path expressions allow us to navigate through schemas as for example, $\{:\ \$Z\}related.Regional\{:\ \$X\}$. The above examples illustrate a unique feature of our language, namely that, unlike logic-based RDF query languages (e.g., SiLRI [10], Metalog [17]), *RQL* provides the ability to smoothly switch between schema and data querying while exploiting - in a transparent way - the taxonomies of labels and multiple classification of resources. More examples on *RQL* can be found in [13].

5.1 RQL Interpreter

The *RQL interpreter* consists of (a) the parser, analyzing the syntax of queries; (b) the graph constructor, capturing the semantics of queries in terms of typing and interdependencies of involved expressions; and (c) the evaluation engine, accessing RDF descriptions from the underlying database [14]. Since our implementation relies on a full-fledged ORDBMS like PostgreSQL, the goal of the *RQL optimizer* is to push as much as possible query evaluation to the underlying SQL3 engine. Then pushing selections or reordering joins to evaluate *RQL* path expressions is left to PostgreSQL while the evaluation of *RQL* functions for traversing class and property hierarchies relies on the existence of appropriate indices (see the last paragraph). The main difficulty in translating an entire *RQL* algebraic expression (expressed in an object algebra a la [9]) to a single SQL3 query is due to the fact that most *RQL* path expressions interleave schema with data querying [8]. This is the case of the query **Q1** presented previously.

The left part of figure 9 illustrates the algebraic translation of **Q1**. The translation of the nested query given in shadow box on bottom is straightforward: the class variable $\$X$ ranges over all the subclasses of *Regional* and its values are filtered according to the conditions in the where clause. The operator `Map` on top is a simple variable renaming for the iterator (Y) defined over the nested query result. Then, the data path expression in the from clause is translated into a semi-join between the source-values of *title* and the proper instances of the class extends (W) returned by the nested query. The connection between the two expressions is captured by a `Djoin` operation (i.e., a dependent join in which the evaluation of the right expression depends on the evaluation of the left one). `Djoin` corresponds to a nested loop evaluation with values of variable Y passed from the left-hand side (i.e., nested query) to the right-hand side. Finally, as illustrated in the right part of Figure 9, the selection operation on titles has been also pushed to the left branch. Both branches in the final expression, can be now translated into corresponding SQL3 queries while the nested loop is evaluated by the RQL interpreter (the `*` indicates an extended interpretation of tables, according to the subtable hierarchy):

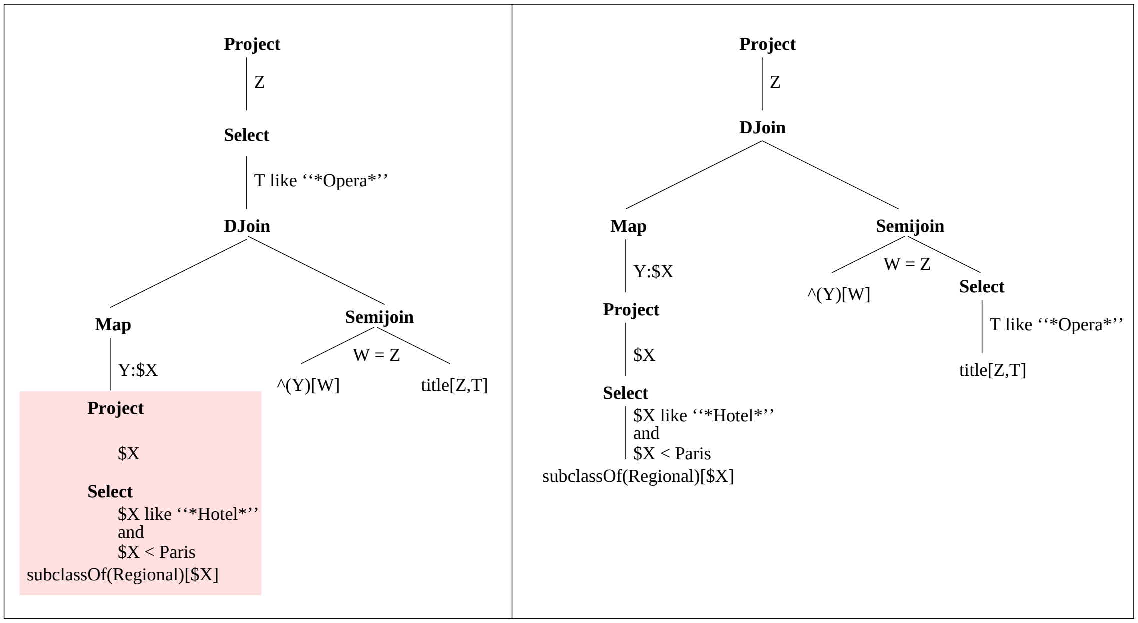


Figure 9: Example of an RQL query optimization

```
select X
from   subclassesof(Regional) X
where  issubclassOf(X, Paris) and X like "*Hotel"


select Z.source
from   title* Z, Y W
where  Z.target like "*Opera*" and Y.source = W
```

Hence, for each class returned by the first query, **Y** will be valuated with the corresponding class name.

6 Summary

In this paper we presented the architecture and functionality of ICS-FORTH RDFSuite, a suite of tools for RDF metadata management. RDFSuite addresses a notable need for RDF processing in Web-based applications (such as Web portals) that aim to provide a rich information content made up of large numbers of heterogeneous resource descriptions. It comprises efficient mechanisms for parsing and validating RDF descriptions, loading into a DBMS as well as query processing and optimization in *RQL*. We also presented a formal data model for RDF metadata and defined a set of constraints that enforce consistency of RDF schemas, thus enabling the incremental validation and loading of voluminous description bases. We argue that, given the immense volumes of information content in Web Portals, this is a viable approach to providing persistent storage for Web metadata. By the same token, efficient access to information in such Portal applications is only feasible using a declarative language providing the ability to query schema and data and to exploit schema organization for the purpose of optimization. We also reported on the results of preliminary tests conducted for assessing the performance of the loading component of RDFSuite. These results illustrate that the approach followed is not only feasible, but also promising for yielding considerable performance gains in query processing, as compared to monolithic approaches. More extensive performance tests are currently being carried out.

Current research and development efforts focus on studying the transactional aspects of loading RDF descriptions in a DBMS, as well as, the problem of updating or revising description bases. A detailed analysis of loading, storage and query evaluation and optimization using alternative representation schemes is underway. Furthermore, appropriate index structures for reducing the cost of recursive querying of deep hierarchies need to be devised as well. Specifically, an implementation of hierarchy linearization is underway, exploring alternative node encodings. Last, but not least, we intend to extend our formal data model to capture higher-order aspects such as statement reification and provide a formal notion of context.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[3] Sofia Alexaki. Storage of RDF Metadata for Community Web Portals. Master's thesis, University of Crete, 2000.

[4] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation. Technical Report CR-rdf-schema-20000327, W3C, Available at http://www.w3.org/TR/rdf-schema, March 27, 2000.

[5] D. Brickley and L. Miller. Rdf, sql and the semantic web - a case study. Available at http://ilrt.org/discovery/2000/10/swsql/.

[6] L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76(2/3):138–164, 1988.

[7] R.G.G. Cattell and D. Barry. *The Object Database Standard ODMG 2.0.* Morgan Kaufmann, 1997.

[8] V. Christophides, S. Cluet, and G. Moerkotte. Evaluating Queries with Generalized Path Expressions. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 413–422, Montreal, Canada, June 1996.

[9] S. Cluet and G. Moerkotte. Nested Queries in Object Bases. In *DBPL'93*, pages 226–242, 1993.

[10] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for rdf. In *W3C Query Languages Workshop*, Cambridge, Mass., 1998.

[11] C. Finkelstein and P. Aiken. *Building Corporate Portals using XML*. McGraw-Hill, 1999.

[12] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing xml data in a relational database. Technical Report 3680, INRIA Rocquencourt, France, May 1999. Available at http://www-caravel.inria.fr/dataFiles/GFSS00.ps.

[13] G. Karvounarakis, V. Christophides, D. Plexousakis, and S. Alexaki. Querying Community Web Portals. Available at http://www.ics.forth.gr/proj/isst/RDF/RQL/, 2001. Submitted for publication.

[14] Greg Karvounarakis. A Declarative RDF Metadata Query Language for Community Web Portals. Master's thesis, University of Crete, 2000.

[15] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, World Wide Web Consortium, February 1999. Available at http://www.w3.org/TR/REC-rdf-syntax.

[16] J. Liljegren. Description of an rdf database implementation. Available at http://WWW-DB.stanford.edu/~melnik/rdf/db-jonas.html.

[17] M. Marchiori and J. Saarela. Query + metadata + logic = metalog. In *W3C Query Languages Workshop*, Cambridge, Mass., 1998.

[18] S. Melnik. Storing rdf in a relational database. Available at http://WWW-DB.stanford.edu/~melnik/rdf/db.html.

[19] Some proposed RDF APIs.
GINF: http://www-db.stanford.edu/~melnik/rdf/api.html,
RADIX: http://www.mailbase.ac.uk/lists/rdf-dev/1999-06/0002.html,
Netscape Communicator: http://lxr.mozilla.org/seamonkey/source/rdf/base/idl/,
RDF for Java: http://www.alphaworks.ibm.com/formula/rdfxml/.

[20] F. Tian, D. DeWitt, J. Chen, and C. Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. Technical report, CS Dept., Universiy of Wisconsin, 2000. Available at http://www.cs.wisc.edu/niagara/papers/vldb00XML.pdf.

[21] Karsten Tolle. ICS-Validating RDF Parser: A Tool for Parsing and Validating RDF Metadata and Schemas. Master's thesis, University of Hannover, 2000.

[22] S. Weibel, J. Miller, and R. Daniel. Dublin Core. In *OCLC/NCSA metadata workshop report*, 1995.