The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases[*]

Sofia Alexaki
Vassilis Christophides
Gregory 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. Next, we present the design of a persistent RDF Store (RSSDB) for loading resource descriptions in an 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 outperforms, both in storage volumes and query execution time, other approaches using a monolithic table to represent resource descriptions and schemas under the form of triples. Last, we briefly present RQL, a declarative language for querying both RDF descriptions and schemas, and sketch query evaluation on top of RSSDB.

1. INTRODUCTION

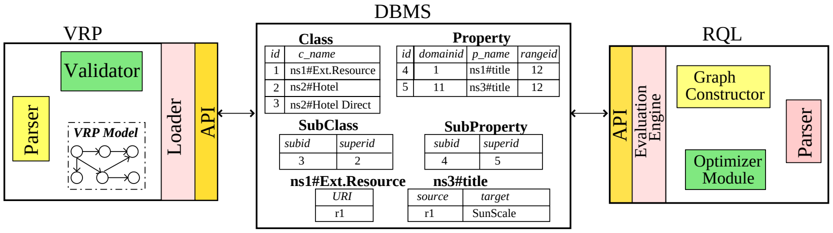
Metadata are widely used in order to fully exploit information resources (e.g., sites, documents, data, images, etc.) available on corporate intranets or the Internet. The Resource Description Framework (RDF) [17] aims at facilitating the creation and exchange of metadata as any other Web data. More 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 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 different target communities (corporate, inter-enterprise, e-marketplace, etc.). The most distinctive RDF feature is its ability to suport superimposed descriptions for the same Web resources, enabling content syndication - and hence, automated processing - in a variety of application contexts. To interpret resource descriptions within or across communities, RDF allows for the definition of schemas [6] i.e., vocabularies of labels for graph nodes (i.e., *classes*) and edges (i.e., *properties*). Furthermore, these vocabularies can be easily extended to meet the description needs of specific (sub-)communities 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, as well as, emerging application standards for Web data and services syndication (e.g., the RDF Site Summary [5], the Dublin Core [25], or the Web Service Description Language [26]). 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

[*] This work was partially supported by the EC project C-Web (IST-1999-13479) and Mesmuses (IST-2000-26074).

Semantic Web Workshop 2001 Hongkong, China

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



Class

id	c_name
1	ns1#Ext.Resource
2	ns2#Hotel
3	ns2#Hotel Direct

Property

id	domainid	p_name	rangeid
4	1	ns1#title	12
5	11	ns3#title	12

SubClass

subid	superid
3	2

SubProperty

subid	superid
4	5

ns1#Ext.Resource

URI
r1

ns3#title

source	target
r1	SunScale

Figure 1: Overview of the ICS-FORTH RDFSuite

Portal of Netscape exports 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 [22] does not ensure fast deployment and easy maintenance of real-scale RDF applications. Still, we want to 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.

In this paper we present `ICS-FORTH RDFSuite` [14], 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). The paper makes the following contributions:

- Section 3 introduces a formal data model for *description bases* created according to the RDF Model & Syntax and Schema specifications [17, 6] (without reification). 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.
- Section 4 presents our persistent RDF Store (RSSDB) for loading resource descriptions in an object-relational DBMS (ORDBMS) by exploiting the available RDF schema knowledge. Our approach preserves the flexibility of RDF in refining schemas and/or enriching descriptions at any time whilst it can be customized in several ways according to the specificities of both the manipulated desription bases and the underlying RDF application scenario.
- Section 5 sketches the functionality of a declarative language, called *RQL*, for querying both RDF descriptions and related schemas. The implementation of *RQL* on top of RSSDB is described with emphasis on how the *RQL* interpreter pushes query evaluation as much as possible to the underlying ORDBMS.
- Section 6 illustrates the performance of RSSDB for storing and querying voluminous RDF descriptions, such as the ODP catalog. In particular, RSSDB outperforms both in storage volumes and query execution time other approaches using a monolithic table [20, 18, 7] to represent resource descriptions and schemas under the form of triples.

Finally, Section 7 presents our conclusions and draws directions for further research.

2. THE OPEN DIRECTORY CATALOG

Portals are nowadays becoming increasingly popular by enabling the development and maintenance of specific communities of interest (e.g., enterprise, professional, trading) [12] on corporate intranets or the Internet. Such *Community Web Portals* essentially provide the means to select, classify and access, in a semantically meaningful way, various information resources. Portals may be distinguished according to the breadth of the target community (e.g., targeting horizontal or vertical markets), and the complexity of information resources (e.g., sites, documents, data). In all cases, the key Portal component is the *Knowledge Catalog* holding descriptive information, i.e., *metadata*, about the community resources.

For instance, the catalogs of Internet (or horizontal) Portals, like Yahoo! or Open Directory (ODP), use huge hierarchies of topics to semantically classify Web resources. Specifically, Table 1 lists statistics of 15 (out of 16) ODP hierarchies (version of 16-01-2001), comprising 252840 topics under which 1770781 sites are classified. We can observe that ODP hierarchies are not deep (the maximum depth is 13) while the number of direct subclasses of a class varies greatly (the maximum number of subclasses is 314 but the average is around 4.02). Additionally, various administrative metadata (e.g., titles, mime-types, modification dates) of resources are usually created using an OCLC Dublin-Core like schema [25]. 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 declarative access to the catalog content. In the sequel, we 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. Figure 2 depicts 2 (out of the 16) hierarchies in the ODP topics schema, namely, *Regional* and *Recreation*, whose topics are represented as RDF classes (see the `rdf` and `rdfs` 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

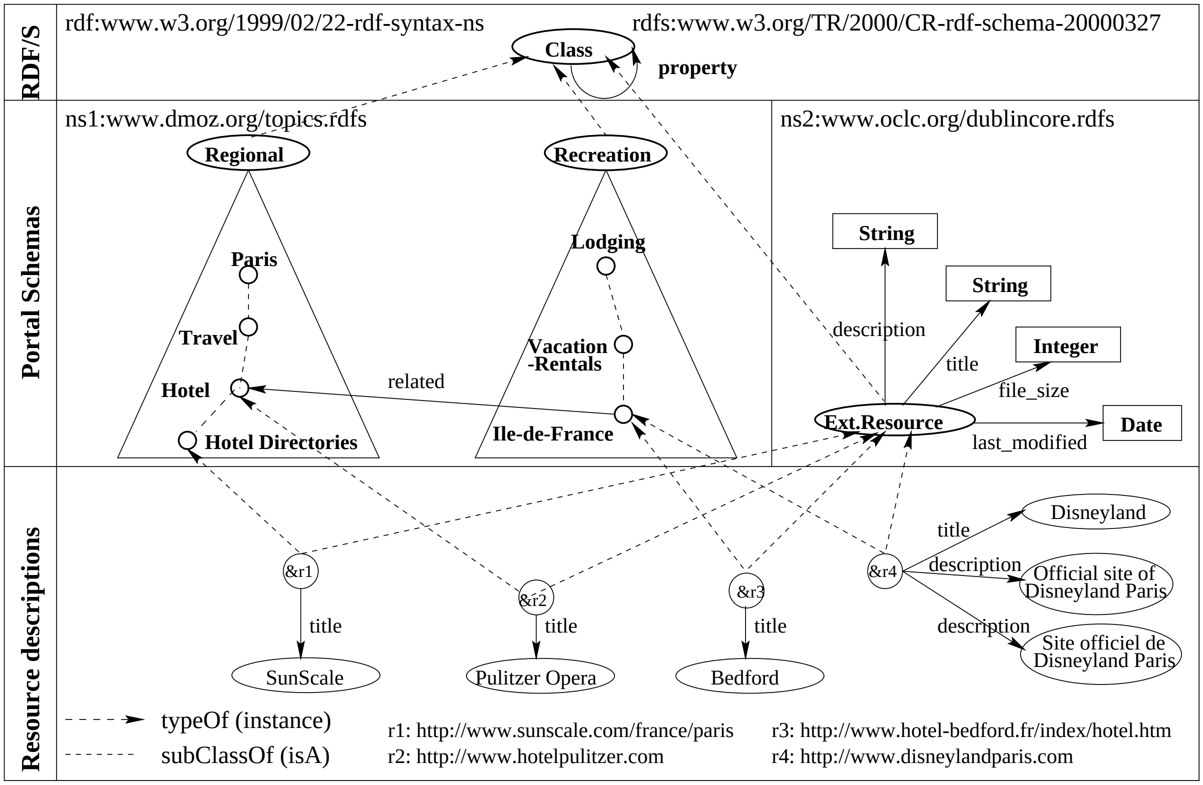

RDF/S
rdf:www.w3.org/1999/02/22-rdf-syntax-ns
rdfs:www.w3.org/TR/2000/CR-rdf-schema-20000327
Class
property
Portal Schemas
ns1:www.dmoz.org/topics.rdfs
Regional
Recreation
ns2:www.oclc.org/dublincore.rdfs
Paris
Travel
Hotel
Hotel Directories
Lodging
Vacation
-Rentals
Ile-de-France
related
String
String
Integer
Date
description
title
file_size
Ext.Resource
last_modified
Resource descriptions
&r1
&r2
&r3
&r4
title
title
title
title
description
description
SunScale
Pulitzer Opera
Bedford
Disneyland
Official site of Disneyland Paris
Site officiel de Disneyland Paris
typeOf (instance)
subClassOf (isA)
r1: http://www.sunscale.com/france/paris
r2: http://www.hotelpulitzer.com
r3: http://www.hotel-bedford.fr/index/hotel.htm
r4: http://www.disneylandparis.com

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 captures various descriptive elements of Dublin-Core [25] (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 class 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* (e.g., Bags) for grouping statements, as well as, higher-order statements (i.e., *reification*[2]). 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).

Hence, RDF properties are *unordered* (e.g., the property *title* can appear 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 superimposed descriptions of community resources, while preserving a conceptually unified view of the 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 [3] or the recently proposed models for semistructured or XML databases [1]:

- *Classes do not define object or relation types*: an instance of a class is just a resource URI;
- *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;
- *Properties may also be refined* by respecting a minimal set of constraints (domain and range compatibility).

Note also that due to multiple classification, resources may have quite irregular descriptions modeled only through an exception mechanism a la SGML [15]. Last but not least, semistructured or XML models can't distinguish between entity (e.g., `ExtResource`) and property labels (e.g., `title`). Therefore existing proposals cannot be used, as such, to manipulate 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 can be used by the underlying DBMS to improve storage volumes and optimize queries on these graphs.

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

Hierarchy	Max.Depth	Avg.Depth	Max.Subclass	Avg.Subclass	#Topics	#Resources
news.rdf	7	5	51	5.625	721	47735
kat.rdf	7	4.8	46	6	761	7730
home.rdf	8	5	53	5	1722	26688
shopping.rdf	9	4.8	61	4.5	3349	88821
health.rdf	9	5.3	52	5.3	3202	45519
games.rdf	10	5.4	125	5.3	4857	36181
computers.rdf	10	5.3	147	5	6010	91597
recreation.rdf	11	5.87	85	5.47	7269	93929
business.rdf	11	5.9	52	4.69	6833	161877
reference.rdf	13	7.13	154	3.92	6483	75105
sports.rdf	9	5.56	178	7.02	10625	66280
science.rdf	10	6.53	314	5.05	8667	65939
society.rdf	12	6.3	157	6.27	16250	161433
arts.rdf	11	5.5	267	6.5	25314	214795
regional.rdf	13	7.7	254	3.37	150762	587152
Total	13	6.86	314	4.02	252825	1770781

Table 1: ODP Topic Hierarchy Statistics

3. A FORMAL DATA MODEL FOR RDF

Since the notion of **resource** is somehow overloaded in the RDF M&S and RDFS specifications [17, 6], 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*).

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 [17, 6]. The set $\mathcal{P}$ also contains integer labels ($\{1, 2, \dots\}$) 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 $range(p_1) \preceq range(p_2)$[3].

3.1 Additional RDF Constraints

Three remarks are noteworthy. First, unlike the current RDF M&S and RDFS specifications [17, 6] 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 domain and range. More precisely, 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 a property 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-). In this context, the `rdfs:Class` definition should be used, in order to define a property with domain or range all the token resources. Conforming to RDF/S, `rdf:Resource` should be used as the domain or range, when we need to define properties viewing schema classes also as tokens (e.g., `rdfs:seeAlso`, `rdfs:isDefinedBy`, `rdfs:comment`, `rdfs:label`).

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

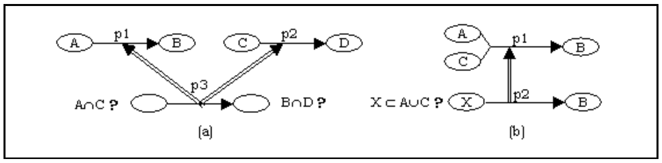


Figure 3: Inconsistencies in Property Specialization

Last, we need to consider an additional constraint of a syntactic nature, imposing that *class and property definitions be complete*. This means that, on the one hand, superclass and superproperty declarations should accompany the class and property definitions respectively and, on the other hand, the domain and range of a property should be given along with the definition 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 used 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 [17, 6] this constraint ensures that the *union of two valid RDF hierarchies of names is always valid*. The imposed constraints (**C3**-**C5**) are summarized in Table 2 using the notation introduced in this section.

3.2 RDF Typing System

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). 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[4]. Alternatives capture the semantics of union (or variant) types [8], 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.

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[5] of type $[(\tau_1 + \tau_2 + \ldots + \tau_n)]$. Unlike traditional object data models, RDF classes are then represented as unary relations of type $\{\tau_U\}$ while properties as binary relations of type $\{[\tau_U, \tau_U]\}$ (for relationships) or $\{[\tau_U, \tau_L]\}$ (for attributes). Furthermore, RDF containers can also be 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[6] is captured by a *population function* $\pi : C \rightarrow 2^U$. The set of all values forseen by our model is denoted by V.

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* $p \in P$*,* $[\![p]\!] = \{[v_1, v_2] \mid v_1 \in [\![domain(p)]\!],$ $v_2 \in [\![range(p)]\!]\} \cup \bigcup_{p' \prec p} [\![p']\!]$.

As usual, the interpretation of classes and properties in our model is set-based. This implies that a resource URI appears only once in the extent of a class even when it is classified several times under its subclasses (i.e., it belongs to the direct extent of the most specific class). The notation $\hat{}[\![.]\!]$ is used in Table 2 to distinguish between strict and extended interpretation of classes and properties.

3.3 RDF Description Bases and Schemas

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* $\lambda : V_S \cup E_S \rightarrow N \cup T$*, and* ψ *is an incidence function* $\psi : E_S \rightarrow V_S \times V_S$*.*

The *incidence function* captures the `rdfs:domain` and `rdfs:range` declarations of properties[7]. 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. Additionally, we impose a *unique name assumption* on the labels of RS nodes and edges.

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* $\psi : E_D \rightarrow V_D \times V_D$*,* ν *is a value function* $\nu : V_D \rightarrow V$*, and* λ *is a labeling function* $\lambda : V_D \cup E_D \rightarrow 2^{N \cup T}$ *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* $p \in P$*.*

Note that the *labeling function* captures the `rdf:type` declaration that associates each RDF node with one or more class names (opposite to traditional object models) which may be defined in several well-formed hierarchies of names. Additionally, integer labels ($\{1, 2, \ldots\}$) are used as property names by the members of RDF container values. The imposed constraints (**C6**-**C7**) are summarized in Table 2 using the notation introduced in this section.

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

[5] 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 $[j_1 : v_1, \ldots, j_n : v_n]$.

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

[7] Constraint **C4** of Table 2 ensures that `rdfs:domain` and `rdfs:range` are not any more relations as in the current RDF M&S and RDFS specifications [17, 6].

Alphabets:	**C1**	Class, Property and Type names are mutually exclusive $\mathcal{C} \cap \mathcal{P} \cap T = \emptyset$
	C2	Literal, Resources and Container values are mutually exclusive $\mathcal{L} \cap \mathcal{U} \cap \mathcal{V} \setminus (\mathcal{L} \cup \mathcal{U}) = \emptyset$
Schema:	**C3**	$\forall c, c', c'' \in \mathcal{C}$:
	C3.1	• Class is the root of the class hierarchy: $c \prec Class$
	C3.2	• subClassOf relation is transitive: $c \prec c', c' \prec c'' \Rightarrow c \prec c''$
	C3.3	• subClassOf relation is antisymmetric: $c \prec c' \Rightarrow c \neq c'$
	C4	Domain and range of properties should be defined and they should be unique: $\forall p \in P, \exists! c_1 \in \mathcal{C}\ (c_1 = domain(p)) \wedge \exists! c_2 \in \mathcal{C} \cup T_L\ (c_2 = range(p))$
	C5	$\forall p, p', p'' \in \mathcal{P}$:
	C5.1	• Property is the root of the property hierarchy: $p \prec Property$
	C5.2	• subPropertyOf relation is transitive: $p \prec p', p' \prec p'' \Rightarrow p \prec p''$
	C5.3	• subPropertyOf relation is antisymmetric: $p \prec p' \Rightarrow p \neq p''$
	C5.4	• if p is subPropertyOf p' then the domain (range) of p is a subset of the domain (range) of p': $p \prec p' \Rightarrow domain(p) \preceq domain(p') \wedge range(p) \preceq range(p')$
Data:	**C6**	$\forall v \in \mathcal{V}$:
	C6.1	• if v is a URI then it is an instance of one or more classes not related by $\preceq$: $v \in \mathcal{U} \Rightarrow \lambda(v) \subseteq \mathcal{C}$ $\forall c, c' \in \mathcal{C}, v \in \hat{}\llbracket c \rrbracket, c \preceq c' \Rightarrow v \notin \hat{}\llbracket c' \rrbracket$
	C6.2	• if v is a literal value then it is an instance of one (and only one) Literal type: $v \in \mathcal{L} \Rightarrow \lambda(v) \subseteq T_L$ and $\lambda(v)$ is a singleton
	C6.3	• if v is a container value then it is an instance of one (and only one) Container type: $v \in \mathcal{V} \setminus (\mathcal{L} \cup \mathcal{U}) \Rightarrow \lambda(v) \subseteq T_{B \vert S \vert A}$ and $\lambda(v)$ is a singleton
	C7	$\forall p \in \mathcal{P}, [v_1, v_2] \in \llbracket p \rrbracket$:
	C7.1	• if p belongs to the set $\{1, 2, \ldots\}$ then v_1 is an instance of either `rdf:Bag` or `rdf:Seq` or `rdf:Alt`: $p \in \{1, 2, \ldots\} \Rightarrow \lambda(v_1) \subseteq T_{B \vert S \vert A}$
	C7.2	• if p doesn't belong to the set $\{1, 2, \ldots\}$ then v_1 (v_2) belongs to the domain (range) of p: $p \in \mathcal{P} \setminus \{1, 2, \ldots\} \Rightarrow \exists n \in \lambda(v_1), n \preceq domain(p) \wedge \exists m \in \lambda(v_2), m \preceq range(p)$
	C7.3	• Additionally, the direct extends of properties must be unique $p \in \mathcal{P} \setminus \{1, 2, \ldots\} \Rightarrow (\forall p' \in \mathcal{P} \setminus \{1, 2, \ldots\}, p \preceq p' \Rightarrow [v_1, v_2] \notin \hat{}\llbracket p' \rrbracket)$
Reification:	**C8**	A reified statement should have exactly one rdf:predicate, rdf:subject and rdf:object property

Table 2: Formal Definition of Imposed RDF Constraints

3.4 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[8]), 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 [17, 6] as well as the additional constraints we introduced previously (see Table 2). 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 [24] 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 enables a fine-grained processing of RDF statements w.r.t. their type, which is crucial in order to implement an *incremental loading* of RDF descriptions and schemas.

4. THE RDF SCHEMA SPECIFIC DATABASE

This section describes the persistent RDF store (RSSDB) for loading resource descriptions in an object-relational DBMS (ORDBMS). We begin by presenting our schema generation strategy in comparison to monolithic approaches [20, 18, 7] representing resource descriptions and schemas as triples.

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

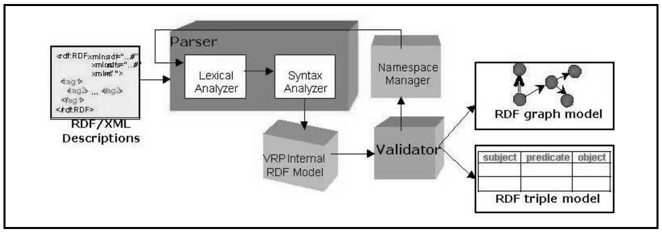


Figure 4: The Validating RDF Parser (VRP)

The core RDF/S model is represented by four tables (see Figure 6-A), namely, Class, Property, SubClass and SubProperty which capture the class and property hierarchies defined in an RDF schema. To save space when storing class and properties names, we also consider a table NameSpace keeping the namespaces of the RDF Schemas stored in the ORDBMS. Furthermore, a table Type is used to hold the names of RDF/S built-in types (e.g., rdf:Property, rdfs:Class), as well as, those of Container (e.g., rdf:Bag, rdf:Seq, rdf:Alt) and Literal types (e.g., string, integer, date). The main novelty of our representation is the separation of the RDF schema from data information, as well, as the distinction between unary and binary relations holding the instances of classes and properties. More precisely, 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, nsid and id of the tables Class and Property and on the attribute subid of the tables SubClass and SubProperty.

It should be stressed that the taxonomic relationships between schema labels is captured by the SubClass and SubProperty tables, while the corresponding instance tables are also connected through the *subtable* relationship, supported today by all ORDBMSs[9]. In other words, RSSDB relies on a schema specific representation of resource descriptions, called *SpecRepr*, similar to the *attribute-based* approach proposed for storing XML data [13, 23]. *SpecRepr* preserves the flexibility of RDF in refining schemas and/or enriching descriptions at any time, and as we will see in the sequel, it can be customized in several ways according to the specificities of both the manipulated description bases and the RDF application scenario (i.e., querying functionality).

This is not the case of other proposals employing a monolithic table [20, 18, 7] to represent RDF metadata under the form of triples. These approaches provide a generic representation (i.e., for all RDF schemas), called *GenRepr*, of both RDF schemas and resource descriptions using two tables (see Figure 6-B), namely, Resources and Triples. The former represents each resource (including schema classes and properties) by a unique *id* whereas the latter represents the statements made about the resources (including classes and properties) under the form of predicate-subject-object triples (captured by *predid*, *subid* and *objid* respectively). Note that in the Triples table we distinguish between properties representing attributes (i.e., objvalue with literal values) and those relationships between resources (i.e., objid with URI object values). Indices (i.e., B-trees) are constructed for all table attributes.

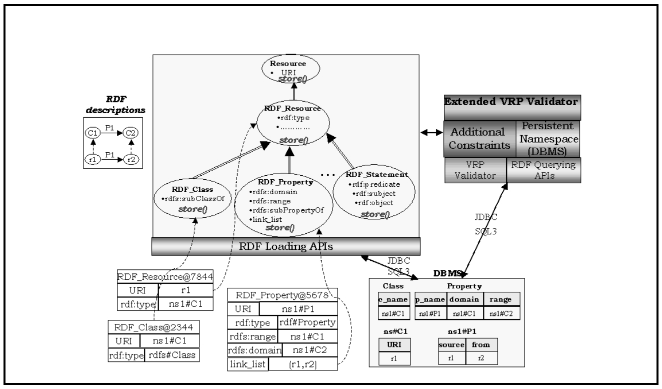


Figure 5: RDF Schema Specific DataBase & Loader

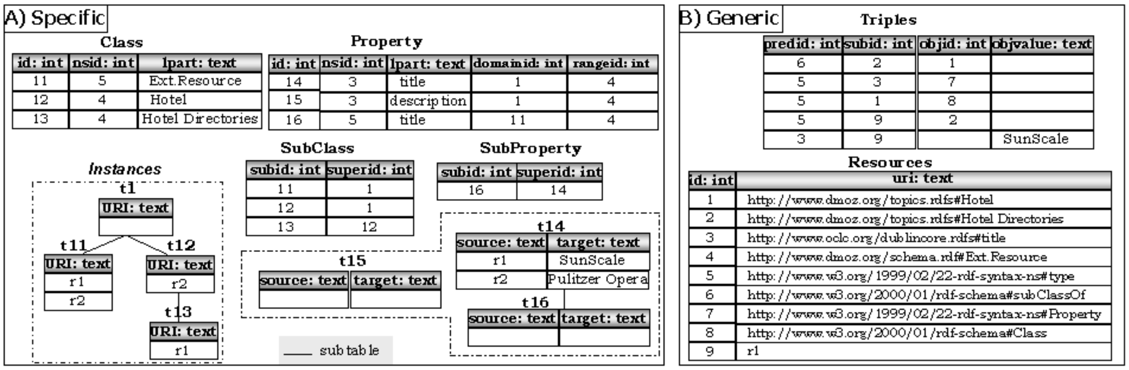
Compared to *GenRepr*, our *SpecRepr* is flexible enough to allow the customization of the physical representation of RDF metadata in the underlying ORDBMS. This is important since no representation is good for all purposes and in most real-scale RDF applications variations of a basic representation are required to take into account the specific characteristics of employed schema classes and properties. Our main goal here is to reduce the total number of created instance tables. This is justified by the fact that some commercial ORDBMSs (and not PostgreSQL) permit only a limited number of tables. Furthermore, numerous tables (e.g., the ODP catalog implies the creation of 252840 tables, i.e. one for each topic) have a significant overhead on the response time of all queries (i.e., to find and open a table, its attributes, etc.). One of the possible variations we have experimented for the ODP catalog is the representation of all class instances by a unique table Instances (see dashed rectangular in Figure 6-A). This table has two attributes, namely uri and classid, for keeping the uri's of the resources and the id's of the classes in which resources belong. The benefits of this *SpecRepr* variation are illustrated in Section 6 given that most ODP classes (i.e. topics) have few or no instances at all (more than 90% of the ODP topics contain less than 30 URIs). For other RDF schemas it could be also interesting to represent in a similar way all the instances of properties, but in general real-scale RDF schemas have more classes than properties. Another alternative to our basic *SpecRepr* could be the representation of properties with range a literal type, as attributes of the tables created for the domain of this property. Consequently, new attributes will be added to the created class tables. The tables created for properties with range a class will remain unchanged. The above representation is applicable to RDF schemas where attribute-properties are single-valued and they are not specialized. Finally, our *SpecRepr* opens the way for a more cleaver representation of class and property names using appropriate encoding systems that preserve the taxonomic relationships of schema labels and enable to optimize recursive traversals on subclass (or subproperty) hierarchies.

4.1 The RDF Description Loader

Figure 5 depicts the architecture of our system for loading RDF metadata in an ORDBMS, namely PostgreSQL[10]. The loader has been implemented [4] in Java and communication

[9] Note that the syntactic constraint (see Section 3) imposing complete class and property definitions, ensures that the table hierarchy created in RSSDB can be only extended through specialization

[10] www.postgresql.org



A) Specific

Class

id: int	nsid: int	lpart: text
11	5	Ext.Resource
12	4	Hotel
13	4	Hotel Directories

Property

id: int	nsid: int	lpart: text	domainid: int	rangeid: int
14	3	title	1	4
15	3	description	1	4
16	5	title	11	4

SubClass

subid: int	superid: int
11	1
12	1
13	12

SubProperty

subid: int	superid: int
16	14

Instances

t1

URI: text

t11

URI: text
r1
r2

t12

URI: text
r2

t13

URI: text
r1

t14

source: text	target: text
r1	SunScale
r2	Pulitzer Opera

t15

source: text	target: text

t16

source: text	target: text

B) Generic

Triples

predid: int	subid: int	objid: int	objvalue: text
6	2	1	
5	3	7	
5	1	8	
5	9	2	
3	9		SunScale

Resources

id: int	uri: text
1	http://www.dmoz.org/topics.rdfs#Hotel
2	http://www.dmoz.org/topics.rdfs#Hotel Directories
3	http://www.oclc.org/dublincore.rdfs#title
4	http://www.dmoz.org/schema.rdf#Ext.Resource
5	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
6	http://www.w3.org/2000/01/rdf-schema#subClassOf
7	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
8	http://www.w3.org/2000/01/rdf-schema#Class
9	r1

Figure 6: Relational Representation of RDF Description Bases

with 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 ORDBMS. 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 ORDBMS. 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.

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 ORDBMS. 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 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.

5. QUERYING RDF DESCRIPTION BASES

As shown in Table 1, the catalog of Portals like Netscape Open Directory comprises huge hierarchies of classes and an even bigger number of resource descriptions. 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 ODP (see Figure 2). ODP allows users to navigate through the topic hierarchy; as shown in Table 1, even if one knows the exact path to follow, this would require approximately 8 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 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:

Q: *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* contain 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 extent of each class valuating Y and the resources valuating Z. Finally, the result of **Q** 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 **Q**, 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*

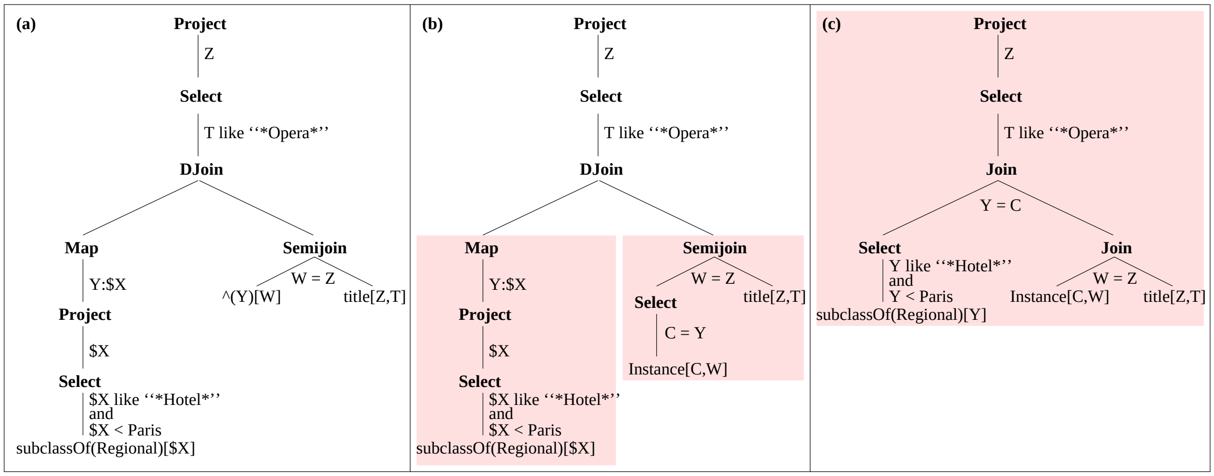


Figure 7: Example of an RQL Query Optimization

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 RQL, namely that 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 (unlike logic-based RDF query languages, e.g., SiLRI [11], Metalog [19]). More examples on RQL can be found in [14].

5.1 RQL Interpreter

The RQL *interpreter*, implemented in C++, 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 [16]. 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 Section 6). The main difficulty in translating an entire RQL algebraic expression (expressed in an object algebra a la [10]) to a single SQL3 query is due to the fact that most RQL path expressions interleave schema with data querying [9]. This is the case of the query **Q** presented previously.

Figure 7 (a) illustrates the algebraic translation of **Q**. The translation of the nested query - given in the left branch - is rather straightforward: the class variable $\$X$ 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 extents (W) returned by the nested query, as shown in the right branch. 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. Using the database representation, we can substitute the scan operation on class instances $(\hat{}(Y)[W])$ with a selection on the `Instances` relation, as shown in Figure 7 (b). Then, each of the subqueries in the shadow boxes can be pushed down to PostgreSQL, leaving the evaluation of the `Djoin` to the interpreter. However, transforming the semijoin on the right into a join and pushing the selection criterium $C = Y$ up to the Djoin operation leads to the evaluation plan shown in Figure 7 (c). As one can see, the whole query is expressible in SQL3 (`subclassof` and `issubclassof` are procedural SQL functions) as follows:

```
select  Z.source
from    subclassesof(Regional) Y, Intances I, title* Z
where   issubclassOf(X, Paris) and X like “*Hotel” and
        I.classid=Y and Z.source=I.uri and
        Z.target like “*Opera*”
```

Note that the `*` indicates an extended interpretation of tables, according to the subtable hierarchy. Thus, the whole query can be pushed down to PostgreSQL, and the PostgreSQL optimizer can perform on further (common) optimizations, as pushing down selections, join reordering etc.

6. PERFORMANCE TESTS

In order to evaluate the performance of the two representations, we used as testbed the RDF dump of the Open Directory Catalog (version of 16-01-2001). Experiments have been carried out on a Sun with two UltraSPARC-II 450MHz processors and 1 GB of main memory, using PostgreSQL (Version 7.0.2) with Unicode configuration. During loading and querying, we have used 1000 and 10000 buffers (of size 8KB) respectively. We have loaded 15 ODP hierarchies (see Table 1) with a total number of 252825 topics (i.e., classes) contained in 51MB of RDF/XML files[11]. For these hierarchies, we have also loaded the corresponding descriptions of 1770781 resource URIs (672MB).

6.1 Loading

The leftmost graph of Figure 8 depicts the database size required to load the ODP schema and resource descriptions measured in terms of triples in the schema-specific (*SpecRepr*) and in the generic (*GenRepr*) representation schemes. Note that to each class definition correspond two

[11] This is the volume of the pure ODP schema, produced when properties attributed to the classes are removed.

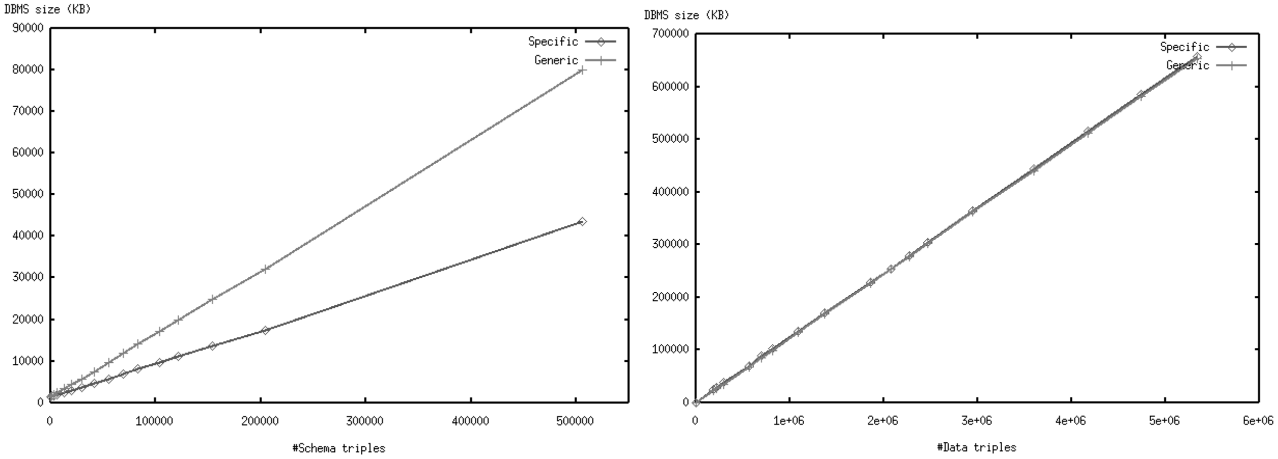


Figure 8: Statistics for Loading Schema and Resource Descriptions

triples: one for the class itself and one for its unique superclass (multiple class specialization is not used in the ODP Catalog). We can observe that in both representations the size of the DBMS scales linearly with the number of schema triples. The tests show that, in *SpecRepr*, each schema triple requires on average *0.086KB* versus *0.1582KB* in the *GenRepr*. This is mainly attributed to the space saved in *SpecRepr* due to the `Namespace` table, as well as to the fact that for each class definition in *GenRepr* three tuples are stored: one in table `Resources` and two tuples in table `Triples` (i.e., 1770781 extra tuples). Although not illustrated here, the average time for loading a schema triple is about *0.0021 sec* and *0.0025 sec* respectively in the two representations. The time required to store a schema triple is larger in *GenRepr* because of extra (252825) tuples stored. When indices are constructed, the average storage volumes per schema triple become *0.1734KB* (*SpecRepr*) and *0.3062KB* (*GenRepr*) and the average loading times become *0.0025 sec* and *0.00317 sec* respectively. The total validation time for the ODP schema is 1539 sec.

The database size required for the resource descriptions is shown in the rightmost graph of Figure 8. As expected, the DBMS size in both representations scales linearly with the number of data triples. The average space required to store a data triple is *0.123KB* in both representations. This should not appear as a surprise since the extra space required in *SpecRepr* for storing the URIs of resources in the property tables compensates for the extra tuples stored for each resource description in *GenRepr*. Note that we could obtain better storage volumes in the *SpecRepr* by encoding the resource URIs as integers, as we did in the *GenRepr*, but this solution comes with extra loading and join costs (between the class and property tables) for the retrieval of the URIs. The tests also show that the average time for loading a data triple is about *0.0033 sec* and *0.0039 sec* respectively in the two representations. When indices are constructed, the average storage volumes per data triple become *0.2566KB* (*SpecRepr*) and *0.2706KB* (*GenRepr*) while the average loading time become *0.0043 sec* and *0.00457 sec* respectively. Despite the use of ids, the indexes in *GenRepr* take up more space because of: a) the extra tuples stored b) the index constructed on the `predid` attribute for which there is no corresponding index in *SpecRepr*.

To summarize, after loading the entire ODP catalog, the size of tables in *GenRepr* is 545MB for `Triples` (5835756 tuples), 202MB for `Resources` (2022869 tuples) and the total size of indexes on these tables is 900MB. In *SpecRepr*, the size is 32MB for `Class` (252825 tuples), 8KB for `Property` (5 tuples), 11MB for `SubClass` (252825 tuples) and 0MB for `SubProperty`, and the total size of indexes on these tables is 44MB. The size of the `Instances` table is 150MB (1770781 tuples) whereas the size of indexes created on it is 140 MB.

6.2 Querying

Table 3 describes the RDF query templates that we used for our experiments, as well as their respective algebraic expressions in the two representations (capital letters abreviate the table names of Figure 6). This benchmark illustrates the desired querying functionality for RDF description bases, namely: a) pure schema queries on class and property definitions (e.g., `Q1`-`Q4`); b) queries on resource descriptions using available schema knowledge (e.g., `Q5`-`Q9`); and c) schema queries for specific resource descriptions (e.g., `Q10`, `Q11`). As a matter of fact, these query templates depict the core functionality of *RQL* presented in the previous section.

To get significant experimental results, we carried out all benchmark queries several times: one initially to warm up the database buffers and then nine times to get the average execution time of a query. Table 4 illustrates the obtained execution time (in sec) for both representations in up to three different result cases per query. The main observation is that *SpecRepr* outperforms *GenRepr* for all types of queries considered. The deviation in performance is more apparent in the cases where self-join computations on the large `Triples` table are required.

GenRepr and *SpecRepr* exhibit comparable performance in queries `Q1, Q2, Q5, Q7, Q10` and `Q11`, with *SpecRepr* outperforming *GenRepr* by a factor of up to 3.73 approximately. This is clearly illustrated in `Q1`, where one tuple is selected from both table `Triples` (selectivity 1,7e-5%) and `Property` (selectivity 20%) using index and sequential scans respectively. As expected, in `Q2`, we can see that the time required to filter a table in both representations depends on the number of tuples in the query results: we have experimented with classes having 1, 30, 314 subclasses which represent in *GenRepr* (*SpecRepr*) selectivities of 1.7e-5% (3.955e-4%), 5.14e-4% (1,19e-2%) and 5.38e-3% (0.124%) for table `Triples` (`SubClass`) in the three cases respectively. The (semi-)joins involved in the evaluation of queries `Q5`, `Q7` and `Q11` incur an additional cost for *GenRepr*,

Query	Description	Algebraic Expression in **GenRepr**	Algebraic Expression in **SpecRepr**
Q1	Find the range (or domain) of a property	$\sigma_{predid=9 \wedge subjid=\texttt{propid}}(T)$	$\sigma_{id=\texttt{propid}}(P)$
Q2	Find the direct subclasses of a class	$\sigma_{predid=6 \wedge objid=\texttt{clsid}}(T)$	$\sigma_{superid=\texttt{clsid}}(SC)$
Q3	Find the transitive subclasses of a class	$\textit{repeat} \quad W_i \leftarrow (W_{i-1} \ltimes_{id=subjid}(\sigma_{predid=6}(T))) - W_{i-1}$ $\textit{until} \quad W_i = W_{i-1}$	$\textit{repeat} \quad W_i \leftarrow (W_{i-1} \ltimes_{id=superid} SC) - W_{i-1}$ $\textit{until} \quad W_i = W_{i-1}$
Q4	Check if a class is a subclass of another class	$\textit{repeat} \quad W_i \leftarrow (W_{i-1} \ltimes_{id=objid}(\sigma_{predid=6}(T))) - W_{i-1}$ $\textit{until} \quad W_i = W_{i-1} \vee \texttt{clsid} \in W_i$	$\textit{repeat} \quad W_i \leftarrow (W_{i-1} \ltimes_{id=subid} SC) - W_{i-1}$ $\textit{until} \quad W_i = W_{i-1} \vee \texttt{clsid} \in W_i$
Q5	Find the direct extent of a class (or property)	$(\sigma_{predid=5 \wedge objid=\texttt{clsid}}(T)) \ltimes_{subjid=id} R$	$\sigma_{id=\texttt{clsid}}(I)$
Q6	Find the transitive extent of a class (or property)	$\cup_{clsid \in \texttt{Q3}}((\sigma_{predid=5 \wedge objid=clsid}(T)) \ltimes_{subjid=id} R)$	$\cup_{clsid \in \texttt{Q3}}(\sigma_{id=clsid}(I))$
Q7	Find if a resource is an instance of a class	$(\sigma_{predid=5 \wedge objid=\texttt{clsid}}(T)) \bowtie_{subjid=id} (\sigma_{URI=\texttt{r}}(R))$	$\sigma_{URI=\texttt{r} \wedge id=\texttt{clsid}}(I)$
Q8	Find the resources having a property with a specific (or range of) value(s)	$(\sigma_{predid=\texttt{propid} \wedge objvalue=\texttt{val}}(T)) \bowtie_{subjid=id} R$	$\sigma_{target=\texttt{val}}(t_{\texttt{propid}})$
Q9	Find the instances of a class that have a given property	$(\sigma_{predid=5 \wedge objid=\texttt{clsid}}(T)) \ltimes_{subjid=subjid}(\sigma_{predid=\texttt{propid}}(T)) \ltimes_{subjid=id}(R)$	$(\sigma_{id=\texttt{clsid}}(I)) \ltimes_{source=URI} (t_{\texttt{propid}})$
Q10	Find the properties of a resource and their values	$(\sigma_{URI=\texttt{r}}(R)) \bowtie_{id=subjid} (\sigma_{predid \neq 5}(T)) \bowtie_{predid=id} (R)$	$\cup_{propid \in P}(\sigma_{source=\texttt{r}}(t_{\texttt{propid}}))$
Q11	Find the classes under which a resource is classified	$(\sigma_{URI=\texttt{r}}(R)) \ltimes_{id=subjid} \sigma_{predid=6}(T)$	$\sigma_{URI=\texttt{r}}(I)$

Table 3: Benchmark Query Templates for RDF Description Bases

whereas in Q10 the join cost (for *GenRepr*) is comparable to the cost of evaluating set union (for *SpecRepr*).

Queries Q3, Q4 and Q6 involve a transitive closure computation (using a variation of the *δ-wavefront* algorithm [21]) over the subclass hierarchy. *SpecRepr* outperforms *GenRepr* by a factor of up to 2.8. In Q3 and Q6, we use the same three classes having 3, 30 and 3879 subclasses and a total of 2, 20 and 9049 instances respectively. The execution times in these three cases depend on the sizes of intermediate results (i.e., the costs of joins involving the tables Triples or SubClass) as well as, the number of iteration steps of the algorithm (i.e., the length of the longest path from the given class to its leaves, called *depth*). In Q4, for the same root class, we have checked for subclasses residing at depth 3, 5 and 7 respectively. The difference in the obtained times between Q3, Q6 and Q4 is due to the different evaluation method used: "top-down" for the former (i.e., from the subtree root to the leaves) and "bottom-up" for the latter.

In the case of queries Q8 and Q9 *SpecRepr* exhibits a much better performance than *GenRepr*. *GenRepr* reaches its limits when table Triples needs to be self-joined (here PostgreSQL uses a hash join algorithm), whereas in *SpecRepr*, a join between two small tables suffices to be evaluated as expected. Specifically, *SpecRepr* outperforms *GenRepr* by a factor ranging from 1753 up to 95538. Note that *GenRep* will suffer similar performance limitations in the evaluation of queries involving complex path expressions (e.g., in more sophisticated schemas than in the case of the ODP catalog) which will essentially result in a number of self-joins of table Triples. Query Q8 has been tested for value ranges returning 1, 10 and 40 resources respectively. In *SpecRepr*, its evaluation involves index scans on the property table, whereas in *GenRepr* different evaluation plans are executed in each case. Q9 has been tested for three properties with 6292, 52029 and 1770584 instances respectively.

To summarize, *SpecRepr* outperforms *GenRepr*, which - in the case of complex path expressions - pays a severe performance penalty for maintaining large tables. We argue that the performance of *SpecRepr* can be further improved by employing an appropriate encoding system (e.g., Dewey, postfix, prefix, etc.) that preserves the taxonomic relationships of schema labels. In this way, checking subclass relationships can be done in constant time. We believe that this approach will prove to be quite useful, not only for RDF, but for tree-structured data, such as XML [2].

7. 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 an ORDBMS 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

Query	Generic			Specific		
	Case 1	Case 2	Case 3	Case 1	Case 2	Case 3
Q1	0.0015			0.0012		
Q2	0.0017	0.0028	0.02	0.0012	0.0022	0.0124
Q3	0.0460	0.082	344.91	0.0463	0.0612	341.98
Q4	0.033	0.0415	0.0662	0.0333	0.0415	0.0662
Q5	0.0043	0.008	0.04	0.0015	0.0028	0.027
Q6	0.0573	0.315	627.43	0.0508	0.1118	482.45
Q7	0.0034	0.0034	0.0034	0.0016	0.0016	0.00174
Q8	124.20	365.73	675.42	0.0013	0.0069	0.0466
Q9	110.58	117.68	185.7	0.031	0.0338	0.1059
Q10	0.0072	0.0072	0.0072	0.0071	0.0071	0.0076
Q11	0.0035	0.0043	0.0056	0.0013	0.0015	0.0015

Table 4: Execution Time of RDF Benchmark Queries

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.

Current research and development efforts focus on studying the transactional aspects of loading RDF descriptions in an ORDBMS, as well as, the problem of updating or revising description bases. A detailed analysis of query 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.

8. REFERENCES

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

[2] S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *12th Symposium on Discrete Algorithms*, 2001.

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

[4] S. Alexaki. Storage of RDF Metadata for Community Web Portals. Master's thesis, Univ. of Crete, 2000.

[5] G. Beged-Dov et.al. RSS 1.0 Specification Protocol. Draft, August 2000.

[6] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation. Technical report, , 2000.

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

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

[9] V. Christophides, S. Cluet, and G. Moerkotte. Evaluating Queries with Generalized Path Expressions. In *Proc. of ACM SIGMOD*, pages 413–422, 1996.

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

[11] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for rdf. In *W3C QL Workshop*, 1998.

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

[13] 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, 1999.

[14] ICS-FORTH. The ICS-FORTH RDFSuite web site. http://139.91.183.30:9090/RDF, 2001.

[15] ISO. Information Processing-Text and Office Systems-Standard Generalized Markup Language (SGML). ISO 8879, 1986.

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

[17] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation. Technical report, , 1999.

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

[19] M. Marchiori and J. Saarela. Query + metadata + logic = metalog. In *W3C QL Workshop*, 1998.

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

[21] G. Qadah, L. Henschen, and J. Kim. Efficient Algorithms for the Instantiated Transitive Closure Queries. *IEEE Transactions on Software Engineering*, 17(3):296–309, 1991.

[22] Some proposed RDF APIs. GINF, RADIX, Mozzila, Jena, Redland: www.w3.org/RDF/Interest.

[23] F. Tian, D. DeWitt, J. Chen, and C. Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. Technical report, Univ. of Wisconsin, 2000.

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

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

[26] Web service description language www106.ibm.com. /developerworks/library/ws-rdf, 2000.