Querying the Semantic Web with RQL*†

G. Karvounarakis A. Magganaraki S. Alexaki V. Christophides D. Plexousakis [a]
Michel Scholl [b] Karsten Tolle [c]

[a]Institute of Computer Science, FORTH,
Vassilika Vouton, P.O.Box 1385, GR 711 10, Heraklion, Greece
{gregkar, aimilia, alexaki, christop, dp}@ics.forth.gr

[b]CEDRIC/CNAM
292 Rue St Martin, 75141 Paris, Cedex 03, France
scholl@cnam.fr

[c]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

Real-scale Semantic Web applications, such as Knowledge Portals and E-Marketplaces, require the management of voluminous repositories of resource metadata. The Resource Description Framework (RDF) enables the creation and exchange of metadata as any other Web data. Although large volumes of RDF descriptions are already appearing, sufficiently expressive declarative query languages for RDF are still missing. We propose *RQL*, a new query language adapting the functionality of semistructured or XML query languages to the peculiarities of RDF but also extending this functionality in order to *uniformly query* both RDF descriptions and schemas. *RQL* s a typed language, following a functional approach a la *OQL* and relies on formal graph model that permits the interpretation of superimposed resource descriptions created using one or more RDF schemas. We illustrate the syntax, semantics and type system of RQL and report on the performance of RSSDB, our persistent RDF Store, for storing and querying voluminous RDF metadata.
Keywords: RDF Description Bases, RDF Query Languages, RDF Stores, Knowledge Portals, E-Marketplaces

1. Introduction

In the next evolution step of the Web, termed the *Semantic Web* [9], vast amounts of information resources (data, documents, programs) will be made available along with various kinds of descriptive information, i.e., *metadata*. Better knowledge about the meaning, usage, accessibility or quality of web resources will considerably facilitate automated processing of available Web content/services. The Resource Description Framework (RDF) [38,11] enables the creation and exchange of resource 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*; ii) a *Schema Definition Language* (RDFS) [11], for creating vocabularies of labels for these graph nodes (called *classes*) and edges (called *property types*); and iii) an *XML syntax* for expressing metadata and schemas in a form that is both humanly readable and machine understandable. The most distinctive feature of the RDF/S data model is its ability to superimpose several descriptions for the same Web resources in a variety of application contexts (e.g., advertisement, recommendation, copyrights, content rating, push channels, etc.). Yet, declarative languages for smoothly querying both RDF resource descriptions and related schemas, are still missing.

*This work was supported in part by the European projects C-Web (IST-1999-13479) and Mesmuses (IST-2000-26074).
†An earlier version of this paper appears in the proceedings of WWW2002.

This ability is particularly useful for real-scale Semantic Web applications such as *Knowledge Portals* and *E-Marketplaces* that require the management of voluminous RDF description bases. For instance, in Knowledge Portals such as Open Directory Project (ODP), CNET, XMLTree[3], various information resources such as sites, articles, etc. are aggregated and classified under large hierarchies of thematic categories or topics. The entire catalog of Portals is exported in RDF, as in the case of Open Directory, comprising around 170M of Subject Topics and 700M of indexed URIs. Unfortunately, searching Portal catalogs is still limited to keyword-based retrieval or theme navigation. The same is true for white (or yellow) pages of emerging E-Marketplaces, where descriptions involve not only information about potential buyers and sellers, but also about provided/requested Web services (i.e., programs). Standards like UDDI [21] and ebXML [26] intend to support registries with service advertisements using keywords for categorization under geographical (e.g., ISO 19119), industry (e.g., NAICS) or product (e.g., UNSPSC) classification taxonomies. There is an ongoing effort to express service descriptions and schemas in RDF (e.g., see the RDF version of WSDL [54]) and take benefit from existing RDF support (e.g., query engines) in service matchmaking (i.e., matching service offers with service requests).

It becomes evident that managing voluminous *RDF description bases* and *schemas* with existing low-level APIs [49] (mostly file-based) does not ensure fast deployment and easy maintenance of real-scale Semantic Web applications. Still, we want to benefit from database technology in order to support *declarative access* and *logical and physical RDF data independence*. In this way, Semantic Web 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 their descriptions to the underlying *database engine*.

Motivated by the above issues, we propose a new query language for RDF descriptions and schemas. Our language, called *RQL*, relies on a formal graph model that captures the RDF modeling primitives (i.e., labels on both graph nodes and edges, taxonomies of labels) and permits the interpretation of superimposed resource descriptions. In this context, *RQL* adapts the functionality of semistructured or XML query languages [1] to the peculiarities of RDF but also extends this functionality in order to *uniformly query* both RDF descriptions and schemas. Thus, users are able to query resources described according to their preferred schema, while discovering, in the sequel, how the same resources are also described using another classification schema. To illustrate our claims, we are using as a running example a cultural community Web Portal (see Section 2). Then, we make the following contributions:

- In Section 3, we introduce a formal data model and type system for *description bases* created according to the RDF Model & Syntax and Schema specifications [38,11]. In order to support superimposed RDF descriptions, the main modeling challenge is to represent properties as *self-existent* individuals, as well as to introduce a graph instantiation mechanism permitting multiple classification of resources.
- In Section 4, we propose *RQL*, the first declarative language for querying RDF description bases. *RQL* is a typed language following a functional approach (a la OQL [14]). Its functionality is illustrated by means of numerous useful RDF queries. The novelty of *RQL* lies in its ability to smoothly combine schema and data querying while exploiting the RDF features.
- In Section 5, we describe our persistent RDF Store (RSSDB) for loading resource descriptions in an object-relational DBMS by exploiting the available RDF schema knowledge. In particular, we illustrate the performance of RSSDB for storing and querying voluminous RDF descriptions, such as the ODP catalog. For this purpose, we rely on a benchmark of RDF query templates depicting the core *RQL* functionality.

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

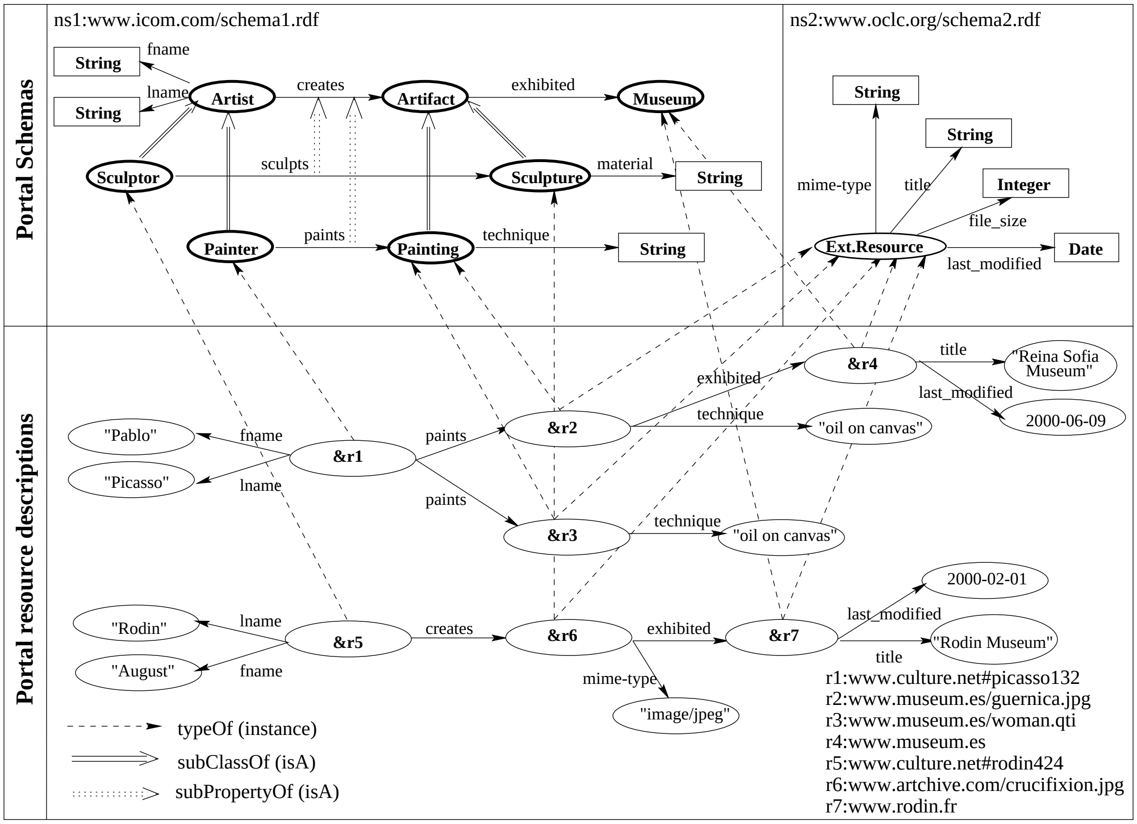

Figure 1. An example of RDF resource descriptions for a Cultural Portal

Finally, in Section 6 we summarize our contribution and draw directions for further research.

2. Motivating example

In this section, we briefly recall the main modeling primitives proposed in the Resource Description Framework (RDF) Model & Syntax and Schema (RDFS) specifications [38,11] using as a running example a cultural Portal catalog. To build this catalog, we need to describe cultural resources (e.g., Museum Web sites, Web pages with exhibited artifacts) both from a Portal administrator and a museum specialist perspective. The former is essentially interested in administrative metadata (e.g., mime-types, file sizes, modification dates) of resources on the Web, whereas the latter needs to focus more on their semantic description using notions such as Artist, Artifact, Museum and their possible relationships. These semantic descriptions[4] can be constructed using existing ontologies (e.g., the International Council of Museums CIDOC Conceptual Reference Model[5]) or vocabularies (e.g., the Open Directory Topics[6]) and cannot always be extracted automatically from resource content or links.

The lower part of Figure 1 depicts the descriptions created for two Museum Web sites (resources `&r4` and `&r7`) and three images of artifacts available on the Web (resources `&r2`, `&r3` and `&r6`). We hereforth use the prefix `&` to denote the involved resource URIs (i.e., resource identity). Let us first consider resource `&r4`. On the one hand, it is described as an `ExtResource`

[4] Note that the complexity of semantic descriptions depends on the nature of resources (e.g., sites, documents, data, programs) and the breadth of the community domains of discourse (e.g., targeting horizontal or vertical markets).

[5] www.ics.forth.gr/proj/isst/Activities/CIS/cidoc

[6] www.dmoz.org

having two properties: `title` with value the string "Reina Sofia Museum" and `last_modified` with value the date 2000/06/09. On the other, `&r4` is also classified under `Museum`, in order to capture its semantic relationships with other Web resources such as artifact images. For instance, we can state that `&r2` is an instance of class `Painting` and has a property `exhibited` with value the resource `&r4` and a property `technique` with string value "oil on canvas". Resources `&r2`, `&r3` and `&r6` are *multiply classified*: under `ExtResource` and under `Painting` and `Sculpture` respectively. Finally, in order to interrelate artifact resources, some intermediate resources for artists (i.e., which are not on the Web) need to be generated, as for instance, `&r1` and `&r5`. More precisely, `&r1` is a resource instance of class `Painter` and its URI is given internally by the Portal description base. Associated with `&r1` are: a) two `paints` properties with values the resources `&r2` and `&r3`; and b) a `fname` property with value "Pablo" and a `lname` property with value "Picasso". Hence, diverse descriptions of the same Web resources (e.g., `&r2` as `ExtResource` and `Museum`) are easily and naturally represented in RDF as *directed labeled graphs*. The labels for graph nodes (i.e., classes or literal types) and edges (i.e., properties) are defined in RDF schemas.

The upper part of Figure 1 depicts two such schemas, intended for museum specialists and Portal administrators respectively. The scope of the declarations is determined by the corresponding *namespace* definition of each schema, e.g., **ns1** (`www.icom.com/schema1.rdf`) and **ns2** (`www.oclc.com/schema2.rdf`). The uniqueness of schema labels is ensured by using namespaces as prefixes of the corresponding class and property names (for simplicity, we will hereforth omit namespaces). In the former schema, the property `creates`, is defined with domain the class `Artist` and range the class `Artifact`. Note that properties serve to represent *attributes* (or *characteristics*) of resources as well as *relationships* (or *roles*) between resources. Furthermore, both classes and properties can be organized into taxonomies carrying inclusion semantics (multiple specialization is also supported). For example, the class `Painter` is a subclass of `Artist` while the property `paints` (or `sculpts`) refines `creates`. In a nutshell, *RDF properties are self-existent individuals* (i.e., decoupled from class definitions) and are by default *unordered* (e.g., there is no order between the properties `fname` and `lname`), *optional* (e.g., the property `material` is not used), *multi-valued* (e.g., we have two `paints` properties), and they can be *inherited* (e.g., `creates`). Note that, although multiple resource classification can be expressed by multiple class specialization, it is an unrealistic alternative, since it implies that, for each class C in our cultural schema, a common subclass of C and `ExtResource` has to be created. However, in a Web setting, resources are usually described by various communities using their independently developed schemas.

2.1. RDF/S vs. Well-Known Data Models

The RDF modeling primitives are reminiscent of knowledge representation languages like Telos [46,48] as well as of data models proposed for net-based applications such as Superimposed Information Systems [24,40] and LDAP Directory Services [33,8]. It becomes clear that the RDF modeling primitives are substantially different from those defined in object or relational database models [3]:

- *Classes do not define object or relation types*: an instance of a class is just a resource URI without any value/state (e.g., the URI `&r2` is an instance of `Painting` regardless of any property associated to it);
- *Resources (URIs) may belong to different classes* not necessarily pairwise related by specialization: the instances of a class may have associated quite different properties, while there is no other class on which the union of these properties is defined (e.g., the different properties of `&r2` and `&r4` which both are instances of `ExtResource`);
- *Properties may also be refined* by respecting a minimal set of constraints i.e., domain and range compatibilities (e.g., the property `creates`).

In addition, less rigid models, such as those proposed for semistructured or XML databases [1], also fail to capture the semantics of RDF description bases. Clearly, most semistructured formalisms, such as OEM [47] or UnQL [12], are totally schemaless (allowing arbitrary labels on edges or nodes but not both). Moreover, semistructured systems offering typing features (e.g., pattern instantiation) like YAT [19,20], cannot exploit the RDF class (or property) hierarchies. Finally, RDF schemas have substantial differences from XML DTDs [10] or the more recent XML Schema proposal [52,41]: due to multiple classification, resources may have quite irregular structures (e.g., the different descriptions of `&r2` and `&r4`) modeled only through an exception mechanism a la SGML [32] in the XML proposals. Last but not least, they can't distinguish between *entity labels* (e.g., `Artist`) and *relationship labels* (e.g., `creates`). On the other hand, XML element content models (i.e., regular expressions) cannot be expressed in RDF since properties are - by default - *unordered*, *optional* and *multi-valued*. As a consequence, query languages proposed for semistructured or XML data (e.g., LOREL [4], StruQL [27], XML-QL [25], XML-GL [15], Quilt [22] or the recent XQuery language [16]) fail to interpret the semantics of RDF node or edge labels. The same is true for the languages proposed to query standard database schemas (e.g., SchemaSQL [37], XSQL [35], Noodle [45]).

Similar difficulties are encountered in logic-based frameworks, which have been proposed for RDF manipulation. For instance, SiLRI [23] proposes some RDF reasoning mechanisms using F-logic [36]. Although powerful, this approach does not capture the peculiarities of RDF: refinement of properties is not allowed (since slots are locally defined within classes), container values are not supported (since it relies on a pure object model), while resource descriptions having heterogeneous types cannot be accommodated (due to strict typing). Metalog [42] uses Datalog to model RDF properties as binary predicates and suggests an extension of the RDFS specification with variables and logical connectors (and, or, not, implies). However, storing and querying RDF descriptions with Metalog almost totally disregards RDF schemas. Furthermore, the recently proposed query language for DAML+OIL [53,28] (a Description Logic extension of RDF/S) has substantially limited expressive power compared to RQL: only existential quantification is supported, disjunction is expressible only through the implicit existential quantification while (safe) negation, nested queries and aggregate functions are not supported.

Finally, a number of languages [44,50,51] have been proposed for querying RDF descriptions and schemas under the form of triples (i.e., atomic statements). These languages consider a flat relational representation of RDF statements (i.e., a SQL table with attributes subject, predicate, and object), as a logical model for issuing queries on RDF graphs. Simple RQL queries (i.e., without transitive closure on class/property hierarchies) can be easily rewritten into these languages, leaving to the users the arduous task of expressing path navigation with explicit join conditions.

3. A Formal Model for RDF

In this section we introduce a graph data model bridging and reconciling W3C RDF Model & Syntax with Schema specifications [38,11]. Compared to the RDF/S specifications, the main contribution of the RQL formal model is the introduction of a semistructured type system for RDF schemas, as well as the representation of RDF descriptions as atomic or complex data values. The connection between the two worlds, is ensured by a type interpretation function which (a) does not impose a strict typing on the descriptions (e.g., a resource may be liberally described using optional and repeated properties which are loosely-coupled with classes); (b) permits superimposed descriptions of the same resources (e.g., by classifying resources under multiple classes which are not necessarily related by subsumption relationships); and (c) allows for a flexible schema refinement (e.g., through specialization of both entity classes and properties).

RDF resource descriptions [38] are represented as *directed labeled graphs* whose nodes are called *resources* (or *literal* and *container values*) and edges are called *properties*. RDFS schemas [11] are also represented as directed acyclic labelled graphs and essentially define vocabularies of labels for graph nodes, called *classes* (or *literal* and *container types*) and edges called *property types*. Labels of classes and properties can be organized into taxonomies carrying inclusion semantics (i.e., class or property subsumption).

More formally, we initially assume the existence of the following countably infinite and disjoint sets of symbols:

- $\mathcal{M} = \{m_1, m_2 \ldots\}$: metaclass names
- $\mathcal{C} = \{c_1, c_2 \ldots\}$: class names
- $\mathcal{P} = \{p_1, p_2 \ldots\}$: property names
- $\mathcal{U} = \{u_1, u_2 \ldots\}$: resource URIs
- $\mathcal{O} = \{o_1, o_2 \ldots\}$: container values
- $\mathcal{L}$: literals, strings, integers, dates, etc.

These sets of symbols are used to label the nodes and edges of an RDF/S graph (at the data, schema and metaschema abstraction layers). More specifically, $\mathcal{M}$ represents the set of metaclass names. Metaclasses can be distinguished into metaclasses of classes, denoted by $\mathcal{M}_c$, and metaclasses of properties, denoted by $\mathcal{M}_p$ ($\mathcal{M} = \mathcal{M}_c \cup \mathcal{M}_p$). The former includes also the default RDF/S name *rdfs:Class* and the latter the default RDF/S name *rdf:Property*, representing respectively the root of the subsumption hierarchy of metaclasses of classes and properties. The set $\mathcal{C}$ contains also the default RDF/S name *rdfs:Resource* representing the root of the user-defined class hierarchy (schema layer).

Although not illustrated in Figure 1, RDF/S also supports structured values called *containers* (data layer). The set $\mathcal{P}$, apart from property names, includes also the arithmetic labels {1,2,3 ... } used as property names for the members of container values. The set of all container values is denoted as $\mathcal{O}$. Each RDF/S container value can be uniquely identified by a URI and can be an instance of one and only one bulk type in Bt, namely *rdf:Bag*, *rdf:Seq* and *rdf:Alt*. Furthermore, the domain of every literal type t in Lt (e.g., string, integer, date, etc.) is denoted as `dom(t)`, while $\mathcal{L}$ represents the set $\cup_{t \in Lt} dom(t)$, i.e., the definition of the default RDF/S name *rdfs:Literal*. As a matter of fact, Lt represents the set of all XML basic datatypes which can be used by an RDF/S schema [41].

Each RDF schema uses a finite number of metaclass names $M \subseteq \mathcal{M}$, class names $C \subseteq \mathcal{C}$, and property names $P \subseteq \mathcal{P}$ as well as the sets of type names Lt and Bt. The property names are defined using metaclass, class, literal or container type names, such that for every $p \in P$, $domain(p) \in M \cup C$ and $range(p) \in M \cup C \cup Bt \cup Lt$ (1.1).

Definition 1 *An* ***RDF/S schema graph*** RS *is a six-tuple* $RS = (V_S, E_S, \psi, \lambda, \prec, N)$, *where* $N = M \cup C \cup P \cup Bt \cup Lt$ *and*

- *V_S is a set of nodes and E_S is a set of edges, where $V_S = M \cup C \cup Bt \cup Lt$ and $E_S = P$ (1.1)*
- *ψ is an incidence function $\psi : E_S \rightarrow V_S \times V_S$ (1.2)*
- *λ is a labeling function $\lambda : V_S \cup E_S \rightarrow 2^M$ (1.3)*
- *$\prec$ is a subsumption relation, such that:*
  - ◇ *`rdfs:Class` is the root of the hierarchy of metaclasses of classes*
  - ◇ *`rdf:Property` is the root of the hierarchy of metaclasses of properties*
  - ◇ *`rdfs:Resource` is the root of the class hierarchy (1.4)* □

The *incidence function* ψ represents the domain (*rdfs:domain*) and range (*rdfs:range*) of properties and imposes the restriction that the domain and range of a property must be unique (`1.2`). Using the example of Figure 1, the property edge *creates* connects the class nodes *Artist* and *Artifact*. The *labeling function* λ captures the *rdf:type* edges connecting the names of the schema layer with those of the metaschema (`1.3`). In particular, applied to the nodes of an RDF/S schema graph, λ returns the names of one or more metaclasses of classes, while applied on edges, it returns the names of one or more metaclasses of properties. For instance, the schema class *Artist* has an *rdf:type* edge to the metaclass *rdfs:Class* and the schema property *creates* to the metaclass *rdf:Property*.

The incidence function ψ and the labeling function λ are *total* on the sets E_S and $V_S \cup E_S$ respectively. This fact does not exclude the case of nodes not related with a schema property edge. Furthermore, we assume that the node and edge labels of an RDF/S schema graph are unique, i.e., we adopt a *unique name assumption* in N (possibly using namespace URIs for disambiguation). (Meta)schema names are finally organized in subsumption hierarchies through the relation "$\prec$" (1.4) capturing the *rdfs:subClassOf* and *rdfs:subPropertyOf* edges.

In order to provide a clear crystal definition of the RDF/S data model semantics we introduce the notion of *valid RDF/S schema graph* imposing adequate restrictions on the formed RDF/S schema graphs.

Definition 2 *An RDF/S schema graph* $RS = (V_S, E_S, \psi, \lambda, \prec, N)$ *is* ***valid*** *if and only if:*

- *For the labeling function* λ*, it holds that:*
  - ⋄ *if* $c \in C$ *and* $\lambda(c) = \{m_1, \ldots, m_n\}$*, for every* $i \in [1 \ldots n]$*,* $m_i \in M_c$ *(2.1)*
  - ⋄ *if* $p \in P$ *and* $\lambda(p) = \{m_1, \ldots, m_n\}$*, for every* $i \in [1 \ldots n]$*,* $m_i \in M_p$ *(2.2)*
- *For the subsumption relation* $\prec$*, it holds that:*
  - ⋄ *the relation* $\prec$ *is a strict partial order (irreflexive, antisymmetric and transitive relation)* [7] *(2.3)*
  - ⋄ *if* $p_1, p_2 \in P$ *and* $p_1 \prec p_2$*, then* $domain(p_1) \preceq domain(p_2)$ *and* $range(p_1) \preceq range(p_2)$ *(2.4)*
  - ⋄ *for every* $c \prec c', c, c' \in C$ *(2.5)*
  - ⋄ *for every* $p \prec p', p, p' \in P$ *(2.6)*
  - ⋄ *for every* $m \prec m'$ *either* $m, m' \in M_c$ *or* $m, m' \in M_p$ *(2.7)* □

Condition 2.1 states that every class must be an instance only of metaclasses of classes. Respectively, condition 2.2 states that a property must be an instance only of metaclasses of properties. Condition 2.3 imposes that the subsumption relation is essentially an *acyclic, binary order relation* so that RDF/S schema or metaschema hierarchies form a *directed acyclic graph* (DAG). As a matter of fact, RDF/S subsumption hierarchies are essentially semi-lattices. Condition 2.4 imposes that the domain and range of a subproperty must be subsumed by the domain and range of its super-properties. Conditions 2.5–2.7 restrict the application of the subsumption relation between (meta)schema names of the same kind (e.g., hierarchies between metaclasses and classes are not allowed). The above constraints guarantee that the *union of two valid RDF schema graphs* is *always valid* w.r.t. the inclusion semantics of (meta)class and property subsumption.

Resource descriptions are defined using a finite set of resource URIs $U \subseteq \mathcal{U}$, literal $L \subseteq \mathcal{L}$ or container values $O \subseteq \mathcal{O}$ and property names P, such that every $p \in P$ emanates from a node in U and ends to a node in $U \cup L \cup O$ (3.1).

Definition 3 *An* ***RDF resource description*** *RD, instance of an RDF/S schema graph* $RS = (V_S, E_S, \psi, \lambda, \prec, N)$*, is a quintuple* $RD = (RS, V_D, E_D, \psi, \lambda)$*, such that:*

- V_D *is a set of nodes and* E_D *is a set of edges, where* $V_D = U \cup L \cup O$ *and* $E_D = P$ *(3.1)*
- ψ *is an incidence function* $\psi : E_D \to V_D \times V_D$ *(3.2)*
- λ *is a labeling function* $\lambda : V_D \to 2^C \cup Lt \cup Bt$ *(3.3)* □

The *incidence function* ψ represents the set of relationships and attributes attached to resources (3.2). The *labeling function* λ captures essentially *rdf:type* edges, connecting the RDF data graph with an RDF/S schema graph (3.3). In particular, applied to resource nodes λ returns the names of one or more classes while applied to value nodes it returns either a literal or a bulk type name. The incidence function ψ and the labeling function λ are *total* on the sets E_D and V_D respectively.

As in the case of RDF/S schemas, we introduce in the sequel the notion of a *valid RDF resource description*.

[7] A relation R is *irreflexive*, when it does not hold that iRi. In case of class/metaclass and property hierarchies, it means that $i \not\prec i$. A relation R is *antisymmetric*, when: if iRj, then it does not hold that jRi. A relation R is *transitive*, when: if iRj and jRk, then iRk. The symbol $\preceq$ extends $\prec$ with equality (thus, the reflexive property holds).

Definition 4 *An RDF resource description* $RD=(RS, V_D, E_D, \psi, \lambda)$ *is* ***valid*** *if and only if* RS *is a valid RDF/S schema and:*

- *for every node* $n \in V_D$*:*
  - $\diamond$ *if* $n \in U \Rightarrow \lambda(n) \subseteq C$ *(4.1)*
  - $\diamond$ *if* $n \in L \Rightarrow \lambda(n) \in Lt$ *(4.2)*
  - $\diamond$ *if* $n \in O \Rightarrow \lambda(n) \in Bt$ *(4.3)*

- *for every* $p \in E_D$ *from node* n *to node* n'*:*
  - $\diamond$ $\exists c \in \lambda(n), c \preceq domain(p) \wedge \exists c' \in \lambda(n'), c' \preceq range(p)$ *(4.4)* $\Box$

Condition `4.1` states that a data resource is instance only of classes. Condition `4.2` (`4.3`) states that a literal (container) value is an instance of one and only one literal (bulk) type. Condition `4.4` imposes that a data resource (or literal, container value) to whom a property is applied (ends), must be an instance of the class (or type) constituting the domain (range) of the property. The above constraints guarantee that the *union of two valid RDF resource descriptions* is *always valid* w.r.t. the inclusion semantics of class and property subsumption.

3.1. A Type System for RDF

Schemas or types have been traditionally exploited in the database world by query languages, such as OQL [14], for several reasons:

Clear data interpretation: a type system provides an unambiguous *understanding of the nature of RDF/S data* returned by a query. For example, we can understand that a URI identifies uniquely a data resource and not a class or property.

Error detection and safety: due to typing rules, we can —on the one hand— *ensure the safety of operations* and —on the other hand— *check the validity of their compositions.* For instance, arithmetic operations on class names are meaningless.

Better Performances: a type system can provide valuable clues for designing a better storage for RDF/S graphs, while it can facilitate the efficient processing of queries (e.g., rewriting path expressions).

In order to introduce a type system for the RDF/S data model, a number of requirements must be taken into consideration. First, contrary to object-oriented schemas, RDF/S properties (i.e., attributes and relationships) are self-existent individuals. For instance, one can query the property *creates* regardless of whether it emanates from resources that are instances of the class *Artist*, or its subclass *Painter*. Second, due to the existence of the data-schema-metaschema abstraction layers, the instances of a type could be data of another type. For example, instances of the metaclass `rdf:Property` are schema properties like *creates*, while instances of this property are (binary) sequences. Last, container values may have heterogeneous contents e.g., literal or other container values, resource URIs or (meta)class and property names. The type system foreseen by RQL is given below:

$\tau = \tau_{M_c} \mid \tau_{M_p} \mid \tau_C \mid \tau_P[\tau, \tau] \mid \tau_U \mid \tau_L \mid \{\tau\} \mid [1 : \tau_1, 2 : \tau_2, \ldots, n : \tau_n] \mid (1 : \tau_1 + 2 : \tau_2 + \ldots + n : \tau_n)$

where τ_{M_c} is a `metaclass of classes` type, τ_{M_p} is a `metaclass of properties` type, τ_C is a `class` type, $\tau_P[\tau, \tau]$ is a `property` type, τ_U is the type of `resource URIs` (including the URIs of namespaces), τ_L is a `literal` type in Lt, $\{.\}$ is a `bag` type, $[.]$ is a `sequence` type and $(.)$ is the `alternative` type. Alternatives in our model capture the semantics of union (or variant) types [13], 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, no subtyping relation is defined in RDF/S. The set of all types one can construct from the (meta)class or property names, the resource URIs and the literal or container types is denoted as `T`.

The RQL type system provides all the arsenal we need to capture collections with homogeneous and heterogeneous contents, as well as to uniformly interpret metaclasses, classes and properties defined at various RDF/S abstraction layers. Thus, classes and metaclasses can be interpreted as unary relations (i.e., bags) of type $\{\tau_U\}$ (data layer) and $\{(\tau_C + \tau_P)\}$ (schema layer) respectively, while properties can be interpreted (data layer) as binary relations (i.e., bag of sequences) of type $\{[\tau_U, \tau_U]\}$ for relationships and $\{[\tau_U, \tau_L]\}$ for atomic attributes. When the property range is a bulk type then the corresponding interpretation involves container values of an appropriate `bag` or `sequence` or `alternative` type. The notation, $\tau_P[\tau, \tau]$, for property types indicates the exact type of its domain and range (first and second position in the sequence). Properties whose domain and range are metaclasses, are interpreted (schema layer) as $\{[(\tau_C + \tau_P), (\tau_C + \tau_P)]\}$, depending on whether the domain (range) of the property is a metaclass of classes or of properties. Generally speaking, (meta)schema names in RDF/S play a dual role both as *labels* and *containers* (disambiguation depends on the operation context). For instance, the name *Artist* refers to the schema graph node with the same label (class label), but also to the resource URIs which are direct and indirect instances of this class. The instantiation of (meta)schema names with a finite set of resource URIs (or schema names) is captured by appropriate *population functions*.

Definition 5 *A* class population function, $\pi_c : C \rightarrow 2^U$*, assigns a finite set of resource URIs to a schema class such that:*

- *for every* $c, c' \in C, c \preceq c', v \in \pi_c \Rightarrow v \notin \pi_{c'}$ *(5.1)* □

In order to capture instantiation of metaclasses of classes and properties we also define the *population functions* $\pi_{M_c} : M_c \rightarrow 2^C$ and $\pi_{M_p} : M_p \rightarrow 2^P$. Note that population functions are the inverse of the labeling functions λ employed at each abstraction layer (definitions 1 and 3). These functions are *partial*, due to the fact that there may be (meta)classes without population. Furthermore, since we can multiply classify resources (or class, property names) under several (meta)classes, π_c (or π_{M_c}, π_{M_p}) is *non-disjoint*. However, condition `5.1` imposes that a resource URI (or class, property name) appears only once in the extension of a (meta)class even though it can be classified more than once in its subclasses (i.e., it belongs to its "closest" class with regards to the defined subsumption hierarchy). It should be stressed that, by default, we use an *extended* (meta)class interpretation, denoted by π^*, including in the proper instances of a (meta)class, also the instances of its subclasses. The set of all values one can construct from the class or property names, the resource URIs and the literal and container values using the *RQL* type system is denoted as `V` and the *interpretation* function $[\![.]\!]$ of *RQL* types is defined as follows:

Definition 6 *Given a population function* π *of (meta)schema names, the* ***interpretation function*** $[\![.]\!]$ *is defined as follows:*

- *literal types:* $[\![\tau_L]\!] = dom(\tau_L)$
- *resource types:* $[\![\tau_u]\!] = u \in U$
- *metaclass types:* $[\![\tau_m]\!] = \{v | v \in \pi^*_M(m)\}$
- *class types:* $[\![\tau_c]\!] = \{v | v \in \pi^*_c(c)\}$
- *property types:* $[\![\tau_p[\tau_1, \tau_2]]\!] = \{[v_1, v_2] | v_1 \in [\![\tau_1]\!], v_2 \in [\![\tau_2]\!]\} \cup \{[\![\tau'_p]\!] | p' < p\}$
- *bag types:* $[\![\{\tau\}]\!] = \{\{v_1, \ldots, v_j\} | j > 0, \forall i \in [1..j], v_i \in [\![\tau]\!]\}$
- *sequence types:* $[\![[\tau]]\!] = \{[1 : v_1, 2 : v_2, \ldots, n : v_n] | n > 0, \forall i \in [1..n], v_i \in [\![\tau_i]\!]\}$
- *alternative types:* $[\![(1 : \tau_1 + 2 : \tau_2 + \ldots + n : \tau_n)]\!] = \{i : v_i | \forall i \in [1..n], v_i \in [\![\tau_i]\!]\}$ □

In order to ensure a *set-based* interpretation of *RQL* types, restriction (5.1.) on (meta)class population is also applied to the interpretation of subproperties. In the rest of the paper, we will use the terms *class* and *property extent* to denote their corresponding interpretations while we will use the notation $\hat{}[\![.]\!]$ to refer only to the proper instances of a (meta)class (or a property).

3.2. RDF Description Bases and Schemas

Taking advantage of the *RQL* types `T` and values `V`, we subsequently introduce formally the notions of a *description schema* and *base*.

Definition 7 *A* ***description schema S*** *is a tuple* $S = (RS, \sigma)$*, where* $RS = (V_S, E_S, \psi, \lambda, \prec, N)$ *is a valid RDF/S schema graph and* σ *is a type function* $\sigma : N \rightarrow T$ □

The *typing function* σ, which is *total* on N, relates (meta)class names with (meta)class types and property names with property types. In addition, it relates the basic XML Schema or RDF/S container type names with the *RQL* literal or bulk types.

Definition 8 *A* ***description base*** D*, instance of a description schema* $S = (RS, \sigma)$*, is a tuple* $D = (RD, \omega)$*, where* $RD = (RS, V_D, E_D, \psi, \lambda)$ *is a set of valid RDF resource descriptions and* ω *is a valuation function* $\omega : V_D \cup E_D \rightarrow V$*, such that:*

- *for every* $n \in V_D$*,* $\omega(n) \in [\![\sigma(\lambda(n))]\!]$ *(8.1)*
- *for every* $p \in E_D$*, from node* n *to node* n'*,* $[\omega(n), \omega(n')] \in [\![\sigma(p)]\!]$ *(8.2)* □

The *valuation function* ω relates the nodes and edges of RDF resource descriptions with one of the values in V. Conditions `8.1` and `8.2` impose that the value of a node (edge) belongs to the interpretation of the type attached to the label of that node (edge). Finally, atomic nodes valuated with literals belong to the interpretation of concrete types like *string*, *integer*, *date*, etc.

In Figure 2 we summarize graphically the formal definitions introduced in this section. An `RDF schema graph RS` consists of a set of names `N`, connected through subsumption ($\prec$) and property edges (ψ). An `RDF resource description RD` is also a graph comprising a set of resource URIs and literal (or container) values which are connected through property edges. Both graphs can be represented using triples of the form `<subject, predicate, object>`. Then, when `RS` and `RD` satisfy appropriate validity constraints we are able to map the schema names `N` to a finite set of *RQL* types `T` (function σ) and the description triples to a finite set of *RQL* values `V` (function ω). `RS` and `T` constitute a `description schema S`, while `V` and `RD` constitute a `description base D` which are connected through a type interpretation $[\![.]\!]$ mapping *RQL* types to values.

3.3. Differences with RDF/S Specs

In RDF/S [38,11] and recently proposed RDF Semantics [30], the distinction of abstraction layers (data, schema, metaschema) is not explicitly stated. Even though these specifications do not assume the existence of abstraction layers, they do not prohibit them. Thus, a class may have instances from different layers, like other classes, properties or resource URIs. Unlike well-known knowledge representation languages like *UML* [56] or *Telos* [46], all RDF/S information (i.e., tokens, classes and metaclasses) is

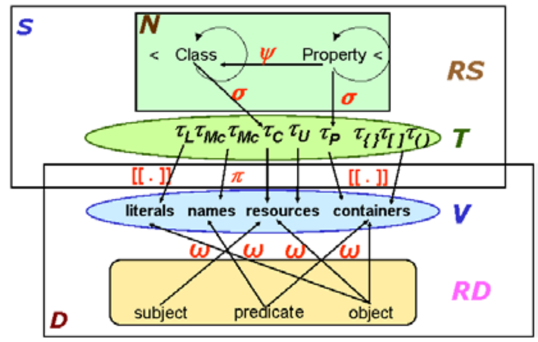

Figure 2. The RQL formal data model

uniformly represented in the form of a graph. On the other hand, in *DAML+OIL* [53] and *OWL* [55], the metaschema is fixed and they do not allow its extension with user-defined metaclasses (although the RDF/S root metaclasses, *rdfs:Class* and *rdf:Property*, are refined to define the *DAML+OIL* metamodel). As indicated in [57], the above flexibility may cause semantic inconsistency problems to application developers. The *RQL* type system makes a clear distinction of the different RDF/S abstraction layers while it provides appropriate interpretation functions to pass from one layer to another.

Furthermore, although the RDF/S specification claims that properties are first-class citizens, properties are not treated as equally as classes. In RDF/S, both a metaclass of classes and a metaclass of properties is a class, in contrast to Telos [46], where a metaclass of individuals is a class, but a metaclass of properties (metaproperty) is a property. Hence, while in Telos a metaproperty can have domain and range, in the RDF/S model it cannot. Furthermore, at the data layer, a property cannot be of type `property`, as in the case of resources and classes. This is attributed to the fact that the *rdf:type* property is applicable only for classes and RDF/S does not provide us with an instantiation mechanism for properties at the data layer. The current version of the *RQL* data model preserves this kind of asymmetry in the manipulation of properties.

Regarding the domain and range of properties, the *RQL* data model enforces the constraint that *the domain and range of a property must always be defined and be unique*. This constraint is opposed to the RDF Semantics [30], which permits an optional declaration of multiple domains and ranges, while considering a conjunctive (i.e., intersection) semantics on endpoint classes of properties. However, a property with undefined domain/range may have as values both resources and literals. For instance, in the example of Figure 1, if the property *technique* has an undefined range, then the token `oil-on-canvas` may be interpreted both as a *string* and a resource URI. Since the intersection of the *rdfs:Class* and *rdfs:Literal* is empty (i.e., the sets of resource URIs $\mathcal{U}$ and literals $\mathcal{L}$), this freedom lead to semantic inconsistencies: resources should be uniquely identified by their URIs while literals by their values. Otherwise, changing the literal value `oil-on-canvas` to `aquarelle` implies that all properties with value `oil-on-canvas` will be updated with `aquarelle` as a new value. Moreover, unlike the RQL data model, RDF Semantics [30] introduce rules to infer that the *source node* of a property is of type `domain` and the *target node* is of type `range`. For instance, if a *title* is attributed to `&r2` then this resource will be automatically classified under all the classes declared in the domain of *title* (i.e., intersection semantics). However, classifying `&r2` as `Painting` and/or `ExtResource` should be under the entire responsibility of application developers. In addition, in the above specifications, specialized properties do not preserve set inclusion semantics of their domain and range. For example, the subproperty *sculpts* of *creates* may have as domain (range) a superclass of *Artist* (*Artifact*). The semantics of properties become completely unclear when subproperties are defined with multiple domains and ranges. We believe that such representation flexibility combined with the previous inference rules (so-called generative interpretation in [30]) impose serious modeling and scalability limitations for real-scale Semantic Web applications especially when numerous namespaces of interconnected schemas are used (e.g., in semantic P2P systems).

Finally, contrary to the new RDFS specification [58] and the RDF Semantics [30], the RQL formal model forbids the existence of cycles in the subsumption hierarchies. Cycles are also allowed in DAML+OIL [53] and OWL [55]. According to these specifications if a resource is declared to be an instance of one classes in a subsumption cycle, then it will also be an instance of all the other classes of the cycle. In other words, all classes participating in the cycle will have the same extent. Thus, cycles in a subsumption hierarchy are mainly used to provide different names for the same (meta)class or property. The rationale behind this modeling choice is the inference of class equivalence. However, the introduction of cycles may considerably affect the semantics of already created RDF/S schemas and resource descriptions, especially when the subclass declarations are provided in many, different namespaces. Once more, declaring versus inferring class equivalences is more preferable for developers mastering the semantics of their applications. Note that DAML+OIL [53] and OWL [55] also support explicit mechanisms for the declaration of equivalent classes or properties with the use of the properties *equivalentTo*, *samePropertyAs* and *sameClassAs*. The RQL type system can easily capture such property types having as domain and range metaclasses.

To conclude, the data model and type system presented in this section makes possible to define a well-founded semantics of a declarative RDF/S query language like RQL involving recursion and functional composition.

4. The RDF Query Language: RQL

RQL is a typed query language relying on a functional approach (a la OQL [14]). It is defined by a set of basic queries and iterators which can be used to build new ones through functional composition. RQL supports *generalized path expressions*, [17,18,4] featuring variables on labels for both nodes (i.e., classes) and edges (i.e., properties). The smooth combination of RQL schema and data path expressions is a key feature for satisfying the needs of several Semantic Web applications such as Knowledge Portals and e-Marketplaces. For the complete RQL syntax, formal semantics and type inference rules, readers are referred to the RQL online documentation.[8]

[8] 139.91.183.30:9090/RDF/RQL/

4.1. Basic Queries

The core RQL queries essentially provide the means to access RDF description bases with minimal knowledge of the employed schema(s). These queries can be used to implement a simple browsing interface for RDF description bases. For instance, in Knowledge Portals, for each topic (i.e., class), one can navigate to its subtopics (i.e., subclasses) and eventually discover the resources (or their total number) which are directly classified under them. Similar needs are exhibited for the classification schemas used in E-Marketplace registries.

To traverse class/property hierarchies defined in a schema, RQL provides functions such as `subClassOf` (for transitive subclasses) and `subClassOf^` (for direct subclasses). For example, the query `subClassOf^(`*Artist*`)` returns a bag with the class names *Painter* and *Sculptor*. Similar functions exist for properties (i.e., `subPropertyOf` and `subPropertyOf^`). Then, for a specific property we can find its definition by applying the functions `domain` (of type $(\tau_C + \tau_M)$) and `range` (of type τ_L for attributes and $(\tau_C + \tau_M)$ for relationships). For instance, `domain(`*creates*`)` returns the class name *Artist*.

We can access the interpretation of classes by just writing their name. For instance, the query `Artist` returns a bag containing the URIs `www.culture.net#rodin424` (&r5) and `www.culture.net#picasso132` (&r1), since these resources belong to the extent of *Artist*. It should be stressed that, by default, we use an *extended* class (or property) interpretation, that is, the union of the set of proper instances of a class with those of all its subclasses. Thus, RQL allows to query complex descriptions using only few abstract labels (i.e., the top-layer classes or properties). In order to obtain the proper instances of a class (i.e., only the nodes labeled with the class name), RQL provides the special operator ("^"): e.g,. `^Artist`.

Additionally, RQL uses as entry-points to an RDF description base not only the names of classes but also the names of properties. For instance, by considering properties as binary relations, the basic query `creates` returns the bag of ordered pairs of resources belonging to the extended interpretation of *creates*:

source	*target*
&r5	&r6
&r1	&r2
&r1	&r3

For cases when same names are used in different schemas one can use a `namespace` clause (in the style of XQuery [16]) to explicitly resolve such naming conflicts e.g.,

```
ns:title
Using Namespace ns=&www.olcl.org/schema2.rdf#
```

More generally, the whole schema can be queried as normal data using the names of appropriately defined metaclasses. This is the case of the default RDF metaclasses `Class` and `Property`. Using them as basic RQL queries, we obtain in our example, the names of all the classes (of type τ_C) and properties (of type τ_P) illustrated in the upper part of Figure 1. Moreover, we can use the name of the built-in metaclass `DProperty`, in order to retrieve only data properties (i.e., involving data resources). Since RDF allows for instantiation links between classes, this query functionality can be easily extended to user defined metaschemas (e.g., DAML+OIL [53]). To retrieve the class (or metaclass) name under which a resource (or class) is classified one can use the function `typeof`: e.g., `typeof(www.artchive.com/crucifixion.jpg)` will return a bag with the class names *Sculpture* and *ExtResource* (due to multiple classification).

Common set operators (`union`, `intersect`, `minus`) applied to collections of the same type are also supported. For example, the query "*Sculpture* `intersect` *ExtResource*" returns a bag with the URI `www.artchive.com/crucifixion.jpg` (&r6), since, according to our example, it is the only resource classified under both classes. However, the following query returns a type error since the function `range` is defined on names of properties and not on names of classes:[9]

[9]It should be stressed that XML query languages like XQuery [16] can be extended with RDF-specific function libraries as those provided by RQL (e.g., `range`, `subclassof`). However, due to the XML and RDF model mismatch they are not able to ensure type safety of the supported functions. For instance, the above query ex-

`bag`(`range`(*Artist*)) `union` `subclassof`(*Artifact*)

As we can see from the above query, besides class or property extents, *RQL* also permits the manipulation of RDF container values. More precisely, we can explicitly construct Bags and Sequences using the basic *RQL* queries `bag` and `seq`. For instance, to find both the domain and range of property *creates* one can issue the query:

`seq` (`domain`(*creates*), `range`(*creates*))

To access a member of a Sequence we can use the operator "`[ ]`" with an appropriate position index. If the specified member element does not exist, the query returns a runtime error. The Boolean operator `in` can be used for membership test in Bags.

For data filtering *RQL* relies on standard Boolean predicates as `=`, `<`, `>` and `like` (for string pattern matching). All operators can be applied on literal values (i.e., strings, integers, reals, dates) or resource URIs. For example, "X = &www.artchive.com/crucifixion.jpg" is an equality condition between resource URIs. It should be stressed that this also covers comparisons between class or property names. For example, the condition "*Painter* `<` *Artist*" returns `true` since the first operand is a subclass of the second. This is equivalent to the basic boolean query *Painter* `in subclassof` (*Artist*). Disambiguation is performed in each case by examining the type of operands (e.g., literal value vs. URI equality, lexicographical vs. class ordering, etc.).

Last but not least, *RQL* is equipped with a complete set of aggregate functions (`min`, `max`, `avg`, `sum` and `count`). For instance, we can inspect the cardinality of class extents (or bags) using the `count` function: `count`(*Painting*).

To conclude this subsection, note that basic *RQL* queries allow us to retrieve the contents of any kind of collection with RDF data or schema information. *RQL* provides a `select-from-where` filter to iterate over these collections and introduce variables. Given that the whole description base or related schemas can be viewed as a collection of nodes/edges, *path expressions* can be used in *RQL* filters to traverse RDF graphs at arbitrary depths.

pressed in XQuery will return all the subclasses of *Artifact* and not a type error.

4.2. Schema Queries

In this subsection, we focus on querying RDF schemas, regardless of any underlying instances. More precisely, we show how *RQL* extends the notion of generalized path expressions [17,18,4] to entire class (or property) inheritance paths in order to implement schema browsing or filtering using appropriate conditions. We believe that declarative query support for navigating through taxonomies of classes and properties is quite useful for real-scale Portal catalogs and E-Marketplace registries, which employ large description schemas. Consider, for instance the following query, where, given a specific schema property we want to find all related schema classes:

Q1: *Which classes can appear as domain and range of the property creates?*

`select` $\$C1$, $\$C2$ `from` $\{\$C1\}$creates$\{\$C2\}$

$\$C1$	$\$C2$
Artist	Artifact
Artist	Painting
Artist	Sculpture
Painter	Artifact
Painter	Painting
Painter	Sculpture
Sculptor	Artifact
Sculptor	Painting
Sculptor	Sculpture

In the `from` clause of the filter, we use a basic *schema path expression* composed of the property name *creates* (i.e., an edge label) and two class variables $\$C1$ and $\$C2$ (i.e., variables over node labels). The {} notation is used in *RQL* path expressions to introduce appropriate schema or data variables (see also next Subsection). In general, class variables are prefixed by $ and - by default - range over the extent of the RDF metaclass *Class*. The type of these variables is τ_C, i.e., names of available schema classes. Since RDF properties can be applied to any subclass of their domain and range (due to polymorphism), the expression $\{\$C1\}$*creates*$\{\$C2\}$ simply denotes that $\$C1$ and $\$C2$ iterate over `subclassof` (`domain`(*creates*)) and `subclassof` (`range`(*creates*)), respectively (including the hierarchy roots). In other words, it is equivalent to the filtering condition "$C1 <=$ `domain`(*creates*) `and` $C2 <=$ `range`(*creates*)" evaluated over *Class*

$\times$ *Class* (i.e., *Class*$\{C1\}$, *Class*$\{C2\}$). We can observe that the above path expression essentially traverses the `rdf:SubClassOf` links in the schema graph. It should be stressed that such a kind of RQL path expressions can be composed not only of edge labels like *creates*, but also of node labels like *Artist*. *Artist*$\{\$C\}$ is a shortcut for `subclassof`$(Artist)\{C\}$ (including the root *Artist*).

The `select` clause defines a projection over the variables of interest (e.g., $\$C1$, $\$C2$). Moreover, we can use "`select *`" to include in the result the values of all variables introduced in the `from` clause. This projection will construct an ordered tuple (i.e., a sequence), whose arity depends on the number of used variables. The result of the filter is a bag. In **Q1** the type of the result is $\{[\tau_C, \tau_C]\}$. It should be stressed that RDF container values are not strictly typed: their members can be any name, URI, literal or other container value. The union types provided by the RQL type system permit the representation of heterogeneous query results. The closure property of RQL is ensured by the supported basic queries for container values (see previous subsection). For simplicity, we will present query results in this paper using an internal relational representation (e.g., as $\neg$1NF relations), instead of RDF containers. Readers can execute all example queries with the RQL online demo[10] to see the results under the RDF/XML syntax for container values or an HTML form produced after XSLT processing.

Let us now see how we can retrieve all related schema properties for a specific class:

Q2: *Find all properties (and their range) that are applicable on class Painter.*

`select` $@P$, `range`$(@P)$
`from` $\{\$C\}@P$
`where` $\$C$ = Painter

$@P$	`range`$(@P)$
creates	Artifact
paints	Painting
lname	string
fname	string

In the `from` clause of **Q2**, we use another *schema path expression* composed of a class variable $\$C$ (i.e., over node labels) and a property variable $@P$ (i.e., over edge labels). In general, property variables are prefixed by @ and by default they range over the extent of the built-in metaclass *DProperty*, containing all data properties. The type of these variables is τ_P, i.e., names of available schema properties. Then, for each possible valuation p of $@P$, the class variable $\$C$ ranges over `subclassof(domain(`p`))`. The condition in the `where` clause will filter $@P$ valuations to keep only those properties for which class *Painter* is equal to their domain (e.g., *paints*) or is a valid subclass of their domain (e.g., *creates*, *lname*, *fname*). In other terms, **Q2** is equivalent to the filtering condition `domain`(P)$>=$*Painter* evaluated over *DProperty* (i.e., *DProperty*{P}). We can observe that the above path expression traverses the `rdfs:domain` and `rdfs:range` links in conjunction with the `rdfs:SubClassOf` links in the schema graph. Note that in the result of **Q2**, `range` is of type union $(\tau_C + \tau_L)$ since data properties may range to classes (i.e., they represent relationships) and literal types (i.e., they represent attributes).

We introduce in path expressions the notation $\{x; C\}$ that filters data nodes x (i.e., resources) which are labeled with a class name C (i.e., the `rdf:type` links). In other terms, it is equivalent to the filtering condition " C `in typeof`(x)". By extension, $\{;C\}$ simply denotes a filtering condition of schema nodes (i.e., classes) identified by a name C and taking into account the `rdfs:SubClassOf` links. For instance, in the expression $\{;Painter\}@P$ the domain of $@P$ is denoted to be *Painter* or any of its superclasses and it implies the filtering condition "Painter `>= domain`(@P)". It is essentially, a shorthand notation for **Q2** by avoiding to introduce an iterator $\$C$ (i.e., class variable) over the `subclassof(domain`(@P))).

To illustrate the expressive power of the RQL schema querying capabilities combined with its functional semantics, consider the following query:

Q3: *Find all information related to class Painter (i.e., its superclasses as well as direct or inherited properties).*

`seq`(Painter, `superclassof^`(Painter)),

[10] http://139.91.183.30:9090/RDF/RQL/

```
(select @P, domain(@P), range(@P)
 from   {;Painter}@P))
```

To collect all relevant information we explicitly construct in **Q3** a sequence with three elements. The first element is a constant (*Painter*) interpreted by the *RQL* type system as a class name (i.e., of type τ_C). The second element is a bag containing the names of the direct superclasses of *Painter* (i.e., of type $\{\tau_C\}$). The third element is a bag of sequences with three elements: the first of type property names (τ_P) and the other two of type union (i.e., Alternative) of class and literal type names (as in **Q2**).

We conclude this subsection, with a query illustrating how *RQL* schema paths can be composed to perform more complex schema navigation. It should be stressed that this kind of query cannot be expressed in existing languages with schema querying capabilities (e.g., XSQL [35]).

Q4: *What properties can be reached (in one step) from the range classes of creates?*

```
select  $Y, @P, range(@P)
from    creates{$Y}.@P
```

$Y	@P	range(@P)
Artifact	exhibited	Museum
Painting	exhibited	Museum
Sculpture	exhibited	Museum
Painting	technique	string
Sculpture	material	string

In **Q4**, the "." notation implies a join condition between the range classes of the property *creates* and the domain of @P valuations: for each class name Y in the range of creates, we look for all properties whose domain is \$$Y$ or a superclass: \$$Y <= domain(@P)$ and \$$Y <= range(creates)$. In other words, this join condition will enable us to follow properties which can be applied to range classes of *creates* (i.e., either because they are directly defined or because they are inherited) to any subclass of the range of *creates*. Schema path expressions may also be exclusively composed of property variables (with or without variables on domains and ranges). For instance, @P.@Q will retrieve all two-step schema paths emanating from the subclasses of the domain of @P and whose second part is either inherited from / defined on superclasses / subclasses of the domain of @Q. The complete set of *RQL* schema path expressions is given in Figure 3, where for each kind of expression, we give the part of the schema graph over which the involved variables range.

4.3. Data Queries

In this subsection, we illustrate how *RQL* generalized path expressions can be used to navigate/filter RDF description bases without taking into account the (domain and range) restrictions implied by the properties defined in an RDF/S schema. This is quite useful since, in most real-scale Knowledge Portals or E-Marketplaces, resources can be multiply classified and several properties coming from different class hierarchies may be used to describe the same resources. In this context, *RQL* generalized path expressions may be liberally composed from node and edge labels featuring both data or schema variables. As explained in the following, the "." notation is used to introduce appropriate join conditions between the left and the right part of the expression depending on the type of each path component (i.e., node vs. edge labels, data vs. schema variables). Consider, for instance, the following query:

Q5: *Find the Museum resources that have been modified after year 2000.*

```
select  X, Y
from    Museum{X}.last_modified{Y}
where   Y >= 2000-01-01
```

In the `from` clause we use a *data path expression* with a class name *Museum* and a property name *last_modified*. The introduced data variables X and Y range respectively over the extent of the class *Museum* (i.e., traversing the `rdf:type` links connecting schema and data graphs) and the *target* values of the extent of the *last_modified* property (i.e., traversing properties in the RDF data graph). The "." used to concatenate the two path components, implies a join condition between the *source* values of the extent of *last_modified* and X. Hence, **Q5** is equivalent to the query $Museum\{X\}, \{Z\}last\_modified\{Y\}$ where $X = Z$. As we can see in Figure 1, the *last_modified* property has been defined with *domain* the class *ExtResource* but, due to multiple classification, X may be valuated with resources

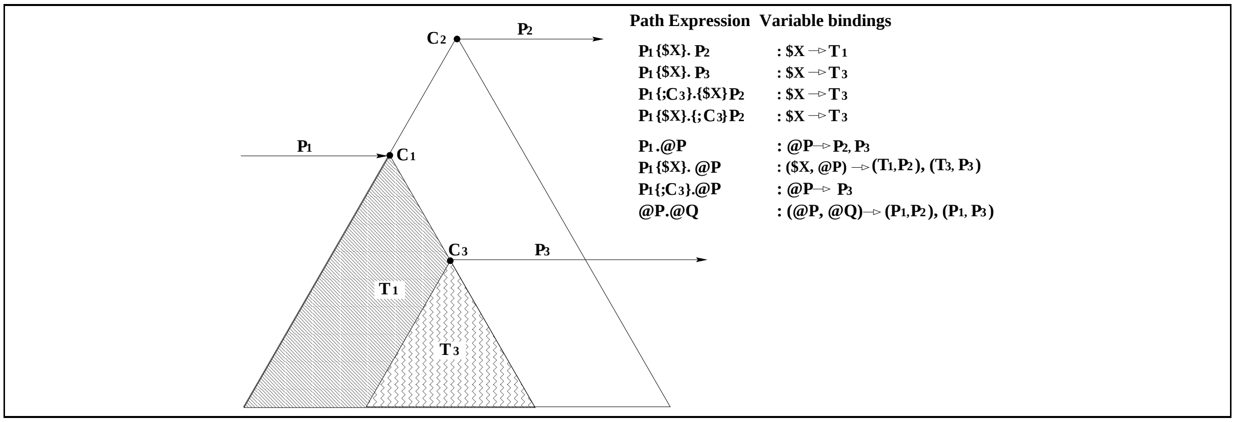

Figure 3. *RQL* Schema Path Compositions

also labeled with any other class name (e.g., *Museum*, *Artifact*, etc.). Yet, in our model X has the unique type τ_U, Y has type the literal type *date*, and the result of **Q5** is of type $\{[\tau_U, date]\}$. According to our example, **Q5** returns the sites `www.museum.es` (&r4) with last modification date 2000-06-09 and `www.rodin.fr` (&r7) with date 2000-02-01.

More complex forms of navigation through RDF description bases are possible, using several data path expressions.

Q6: *Find the names of Artists whose Artifacts are exhibited in museums, along with the related Museum titles.*

`select` V, R, Y, Z
`from` $\{X\}$creates.exhibited$\{Y\}$.title$\{Z\}$,
$\{X\}$fname$\{V\}$, $\{X\}$lname$\{R\}$

In the `from` clause we use three data path expressions. Variable X (Y) ranges over the *source* (*target*) values of the *creates* (*exhibited*) property. Then, the reuse of variable X in the other two path expressions simply introduces implicit (equi-)joins between the extents of the properties *fname*/*lname* and *creates*, on their *source* values. Since the *range* of property *exhibited* is the class *Museum* we don't need to further restrict the labels for the Y values in this query.

Note that due to multiple classification of nodes (e.g., `www.museum.es` (&r4) is both a *Museum* and *ExtResource*) we can query paths in a data graph that are not explicitly declared in the schema. For instance, *creates.exhibited.title* is not a valid schema path since the *domain* of the title property is the class *ExtResource* and not *Museum*. Still, we can query the corresponding data paths by ignoring the schema classes labeling the endpoint instances of the properties (in the style of LOREL [4], or XQuery [16]). This is achieved by using only data variables on path nodes like X, Y and Z. However, the flexibility of *RQL* path expressions enables us to *turn on* or *off* schema information during data filtering with the use of appropriate class and property variables. This functionality is illustrated in the following query:

Q7: *Find the source and target values of properties emanating from ExtResources.*

`select` X, Y
`from` $\{X;$ExtResource$\}@P\{Y\}$

X	Y
&r6	"image/jpg"
&r7	"Rodin Museum"
&r4	"Reina Sofia Museum"
&r7	2000-06-09
&r4	2000-02-01

The *mixed path expression* of **Q7**, features both data (X, Y) and schema variables on graph edges ($@P$). The notation $X; ExtResource$ denotes a restriction of X to the resources that are (transitive) instances of (i.e., labeled by) class *ExtResource*. $@P$ is of type τ_P and is valuated to all properties having as a domain *ExtResource* or one of its superclasses (see **Q2**). Finally, Y is range-restricted, for each successful binding of $@P$, to the corresponding target values. X is of type τ_U while Y type is a union of all the range types of `ExtResource` properties. According to the schema of Figure 1, $@P$ is valuated to *file_size*, *title*, *mime-type*, and *last_modified*, while Y will be of type (*integer*+*string*+*date*).[11] It should be stressed that the data path expression $ExtResource\{X\}.@P\{Y\}$ returns as result not only the values of the properties having as a domain *ExtResource* but also those with domain any class under which instances of *ExtResource* are multiply classified (e.g., *exhibited*, *technique*).

4.4. Combining Schema with Data Queries

In the previous subsections, we have presented the main RQL path expressions allowing us to browse and filter description bases with or without schema knowledge, or, alternatively to query exclusively the schemas. Additionally, RQL filters admit arbitrary mixtures of different kinds of path expressions. In this way, one can start querying resources according to one schema, while discovering in the sequel how the same resources are described using another schema. To our knowledge, none of the existing query languages has the power of RQL path expressions. This functionality is illustrated by the following examples.

Q8: *Find the descriptions of resources whose URI matches "www.museum.es".*

select X, (select $\$W$, (select $@P$, Y
from $\{X;\$W\}@P\{Y\}$)
from $\$W\{X\}$)
from Resource$\{X\}$
where X like "www.museum.es"

In **Q8** we are interested to discover for each matching resource (*Resource* is considered as the top class of all schema classes) the classes under which it is classified and then for each class the properties which are used along with their respective values. This grouping functionality is captured by the two nested queries in the `select` clause of the external query. Note the use of string predicates such as `like` on resource URIs. Then for each successful valuation of X, in the outer query, variable $\$W$ iterates over the classes having X in their extent. Finally, for each successful valuation of X and $\$W$, in the inner query, variable $@P$ iterates over the properties which may have $\$W$ as domain and X as source value in their extent. According to the example of Figure 1 the type of Y is the union (τ_U + *string* + *date*). The final result of **Q8** is given in Figure 4. In cases where a grouped form of RQL results is not desirable, we can easily generate a flat triple-based representation (i.e., subject, predicate, object) of resource descriptions, as in the following query:

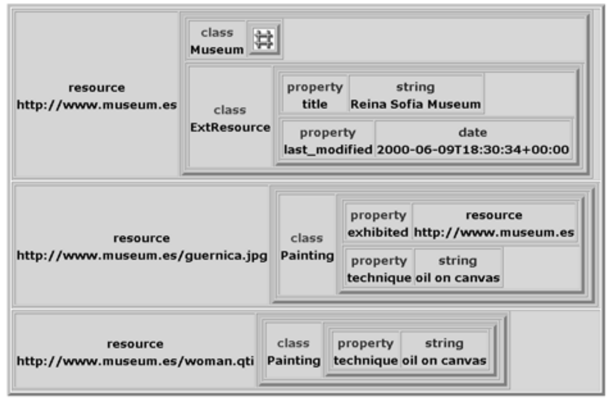

Figure 4. The result of Q8 in HTML form

Q9: *Find the description, under the form of triples, of resources excluding properties related to the class ExtResource.*

((select X, $@P$, Y from $\{X\}@P\{Y\}$)
union
(select X, type, $\$W$ from $\$W\{X\}$))
minus
((select X, $@P$, Y from $\{X$;ExtResource$\}@P\{Y\}$)
union
(select X, type, ExtResource from ExtResource$\{X\}$))

In **Q9** we essentially perform a set difference between the entire set of resource descriptions (i.e., the attributed properties and their values, as well as, the class instantiation properties) and the descriptions of resources which are instances of class *ExtResource*. The only subtle issue in **Q9** is the typing of the two union query results. First, the inferred type for the constants `type` and *ExtResource* (in the select clause of the two union subqueries) is τ_P (i.e., a property name) and τ_C (i.e., class name). Second, variables Y and $\$W$ (in the select clause of the first union) is

[11]In case we want to filter Y values in the `where` clause, RQL supports appropriate coercions of union types in the style of POQL [2] or Lorel [4].

of type (τ_U + *string* + *float* + *integer* + *date*) and τ_C. In this case, the union operation is performed between subqueries of different types. The RQL type system is equipped with rules allowing us to infer appropriate union types whenever it is required for query evaluation, as for example, ($\tau_U + string + float + integer + date + \tau_C$). Note that set-based queries as **Q9** are not supported by the so-called triple-based query languages [44,50,51].

5. The RDF Schema-Specific Database

We have implemented RDF storage and querying on top of the PostgreSQL object-relational DBMS (ORDBMS).[12] The architecture of our persistent RDF Store (RSSDB) is illustrated in Figure 5. It comprises three main components: the RDF validator and loader (**VRP**), the RDF description database (**DBMS**) and the query language interpreter (**RQL**). In the following, we elaborate on the database representation employed by RSSDB, as well as, the performance results in storing and querying voluminous RDF description bases. Readers are referred to [6] for a detailed presentation of the system architecture and components.

5.1. Database Representation

In order to load RDF metadata in a ORDBMS, we consider a database representation depending on the employed RDF schemas (similar to the *attribute-based* approach for storing XML data [29]). Many proposals [43,39] use a single table to represent RDF metadata under the form of triples. These approaches provide a generic representation applicable to all RDF schemas, where both RDF schemas and resource descriptions are stored in two tables called *Resources* and *Triples*. The former represents each resource, whereas the latter represents statements about the resources identified by a unique id. Compared to this representation, our scheme is more flexible as it takes into account the specificity of the schemas (see [7] for a performance analysis).

In our approach, the core RDF/S model is represented by four tables (see Figure 5), namely,

[12] www.postgresql.org

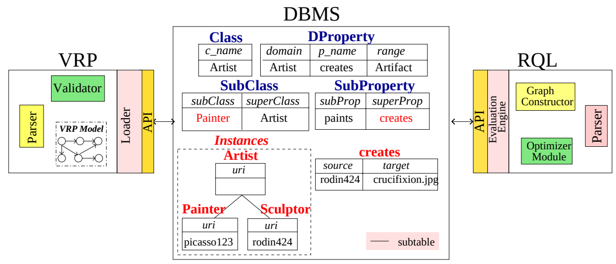

Figure 5. Overview of the ICS-FORTH RSSDB

`Class`, `Property`, `SubClass` and `SubProperty` which capture the class and property hierarchies defined in an RDF schema. The main goal is the separation of RDF schema information 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 (i.e., B-trees) are constructed on the attributes `URI`, `source` and `target` of the above tables, as well as on all the attributes of the tables `Class`, `Property`, `SubClass` and `SubProperty`.

Since no representation is good for all purposes, variations of a basic representation are required to take into account the specific characteristics of the employed schema classes and properties, as well as those of the intended query functionality. Our aim 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.). A variant we have experimented with for storing the ODP catalog, is the representation of all class instances by a unique table `Instances`. This table has two attributes, namely `uri` and `classid`, for storing the uri's of the resources and the id's of the classes which the resources belong to. The benefits of this variant are illustrated in the following section. These benefits arise as a consequence of the fact that most ODP classes (i.e., topics) have few or no instances at all (more than 90%

Query	Description	Algebraic Expression	Case 1	Case 2	Case 3
QB1	Find the range (or domain) of a property	$\sigma_{id=\texttt{propid}}(P)$	0.0012		
QB2	Find the direct subclasses of a class	$\sigma_{superid=\texttt{clsid}}(SC)$	0.0012	0.0022	0.0124
QB3	Find the transitive sub-classes of a class	$repeat \quad W_i \leftarrow (W_{i-1} \bowtie_{id=superid} SC) - W_{i-1}$ $until \quad W_i = W_{i-1}$	0.0463	0.0612	341.98
QB4	Check if a class is a subclass of another class	$repeat \quad W_i \leftarrow (W_{i-1} \bowtie_{id=subid} SC) - W_{i-1}$ $until \quad W_i = W_{i-1} \vee \texttt{clsid} \in W_i$	0.0333	0.0415	0.0662
QB5	Find the direct extent of a class (or property)	$\sigma_{id=\texttt{clsid}}(I)$	0.0015	0.0028	0.027
QB6	Find the transitive extent of a class (or property)	$\cup_{clsid \in \texttt{Q3}}(\sigma_{id=clsid}(I))$	0.0508	0.1118	482.45
QB7	Find if a resource is an instance of a class	$\sigma_{URI=\texttt{r} \wedge id=\texttt{clsid}}(I)$	0.0016	0.0016	0.00174
QB8	Find the resources having a property with a specific (or range of) value(s)	$\sigma_{target=\texttt{val}}(t_{\texttt{propid}})$	0.0013	0.0069	0.0466
QB9	Find the instances of a class that have a given property	$(\sigma_{id=\texttt{clsid}}(I)) \bowtie_{source=URI} (t_{\texttt{propid}}) \bowtie_{subjid=id}(R)$	0.031	0.0338	0.1059
QB10	Find the properties of a resource and their values	$\cup_{propid \in P}(\sigma_{source=\texttt{r}}(t_{\texttt{propid}}))$	0.0071	0.0071	0.0076
QB11	Find the classes under which a resource is classified	$\sigma_{URI=\texttt{r}}(I)$	0.0013	0.0015	0.0015

Table 1
Benchmark Query Templates for RDF Description Bases

of the ODP topics contain less than 30 URIs). Another variant 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 whose range is a class will remain unchanged. The above representation is applicable to RDF schemas where attribute-properties are single-valued and they are not specialized. Multi-valued attributes can always be represented in a pure relational schema by separate tables but this implies an extra translation cost by the *RQL* interpreter. More on *RQL* query evaluations plans can be found in [34].

5.2. Performance Tests

For our performance study we used as a testbed the RDF dump of the Open Directory Catalog (01-16-2001 version). Experiments have been carried out on a Sun with two UltraSPARC-II 450MHz processors and 1 GB of main memory, using PostgreSQL (7.0.2). We have loaded 15 ODP hierarchies with a total number of 252825 topics stored in 51MB of RDF/XML files as well as the corresponding descriptions of 1770781 resources (672MB). Note that only 82744 resources were actually classified under multiple ODP classes/topics.

We have measured the database size required to load the ODP schema and resource descriptions in terms of triples. As expected, the size of the DBMS scales linearly with the number of schema and data triples. The tests show that each schema triple requires on average *0.086KB*. The average time for loading a schema triple is about *0.0021 sec*. When indices are constructed, the average storage volume per schema triple becomes *0.1734KB* and the average loading time becomes *0.0025 sec*. The average space required to store a data triple is *0.123KB*. Note that we could obtain better storage volumes by encoding the resource URIs as integers, 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* without indices and *0.2566KB* with indices while the average loading time becomes *0.0043 sec*.

To summarize, after loading the entire ODP catalog, the size of tables is 32MB for `Class` (252825 tuples), 8KB for `Property` (5 tuples), 11MB for `SubClass` (252825 tuples) and the total size of indices on these tables is 44MB. The size of table `Instances` is 150MB (1770781 tuples) whereas that of the indices created on it is 140 MB.

The left part of Table 1 describes the RDF query templates that we used for our experiments, as well as their algebraic expressions using the first variation of our core representation scheme of section 5.1, i.e., employing a unique table for representing all class instances (capital letters abbreviate the table names of Figure 5). This benchmark illustrates the core functionality of RQL: a) pure schema queries on class and property definitions (`QB1`-`QB4`); b) queries on resource descriptions using available schema knowledge (`QB5`-`QB9`); and c) schema queries for specific resource descriptions (`QB10`, `QB11`). In this context, the most frequently asked queries for Portals like ODP are: `QB2`,`QB3`,`QB5`,`QB8` and `QB9`. The right part of Table 1 displays the resulting execution time (in sec) in up to three different result cases per query. Depending on the particular query templates, the different cases refer to different characteristics of the class or property in question, such as number of subclasses, length of path from a class to its leaves, etc. For the sake of accuracy, we carried out all benchmark queries several times: one initially to warm up the database buffers and then nine times to obtain the average execution time of a query.

Queries `QB3` and `QB6`, as expected are expensive, because they involve a transitive closure computation over the subclass hierarchy. The execution time depends on the size of the intermediate join results, as well as on the number of iterations. The advantage of this representation over the generic representation in terms of query evaluation performance is drastic in the presence of complex path expressions. Indeed, the latter representation implies expensive self joins of a large table, namely *Triples*. In [31] we compared the performance of queries `QB8` and `QB9` with the two representations. Our specific representation outperformed the generic representation by a factor of almost 10^5.

We conclude this section with one remark concerning the encoding of class and property names. Recall that schema or mixed RQL path expressions need to recursively traverse a given class (or property) hierarchy. We can transform such traversal queries into interval queries on a linear domain, that can be answered efficiently by standard DBMS index structures (e.g., B-trees). This can be done by replacing class (or property) names by *ids* using an appropriate encoding, such as the one used in [5]. We are currently working on the choice of a such a linear representation of node or edge labels allowing us to optimize queries that involve different kinds of traversals in a hierarchy.

6. Summary and Future work

In this paper, we presented a data model capturing the most salient features of RDF and a declarative query language, RQL, for uniformly querying both RDF schema and resource descriptions. We reported on the design and implementation of a system for storing and querying voluminous RDF description bases, called RSSDB, and gave some performance results using the ORDBMS PostgreSql. There currently exist two distinct implementations of RQL, one by ICS-FORTH (*139.91.183.30:9090/RDF/RQL*) and the other by Aidministrator (*sesame.aidministrator.nl/rql/*). The latter implementation however does not support the type system presented in this paper. As a matter of fact, RQL is a generic tool actually used by several EU projects (i.e., C-Web, MesMuses, Arion and OntoKnowledge[13]) aiming at building, accessing and personalizing Community Knowl-

[13] See cweb.inria.fr, cweb.inria.fr/Projects/Mesmuses, dlforum.external.forth.gr:8080, www.ontoknowledge.org, respectively.

edge Portals. The optimization of *RQL* query evaluation is a challenging issue and a topic of our current research. In particular, we study the translation of *RQL* into SQL3 queries in the presence of path expressions interleaving schema with data querying, as well as appropriate encoding schemes for class and property taxonomies in order to optimize transitive closure queries over deep hierarchies of names.

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, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Siméon. Querying Documents in Object Databases. *Inter. Journal on Digital Libraries*, 1(1):5–18, April 1997.
3. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
4. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured Data. *Inter. Journal on Digital Libraries*, 1(1):68–88, April 1997.
5. R. Agrawal, A. Borgida, and H.V. Jagadish. Efficient Management of Transitive Relationships in Large Data Bases. In *SIGMOD'89*, pages 253–262, Portland, Oregon, USA, 1989.
6. S. Alexaki, G. Karvounarakis, V. Christophides, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *2nd Inter. Workshop on the Semantic Web*, pages 1–13, Hong Kong, 2001.
7. S. Alexaki, G. Karvounarakis, V. Christophides, D. Plexousakis, and K. Tolle. On Storing Voluminous RDF descriptions: The case of Web Portal Catalogs. In *4th Inter. Workshop on the Web and Databases (WebDB)*, Santa Barbara, CA, 2001.
8. S. Amer-Yahia, H. Jagadish, L. Lakshmanan, and D. Srivastava. On bounding-schemas for LDAP directories. In *Proceedings of the Inter. Conference on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, pages 287–301, Konstanz, Germany, March 2000. Springer.
9. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.
10. T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible markup language (XML) 1.0. W3C Recommendation, February 1998. Available at www.w3.org/TR/-REC-xml/.
11. D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation. Technical report, , 2000.
12. P. Buneman, S.B. Davidson, and D. Suciu. Programming Constructs for Unstructured Data. In *Proceedings of Inter. Workshop on Database Programming Languages*, Gubbio, Italy, 1995.
13. L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76(2/3):138–164, 1988.
14. R.G.G. Cattell, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez. *The Object Database Standard ODMG 3.0*. Morgan Kaufmann, January 2000.
15. S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a Graphical Language for Querying and Restructuring XML Documents. In *Proceedings of Inter. World Wide Web Conference*, Toronto, Canada, 1999.
16. D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: A Query Language for XML. W3C Working Draft 15 November 2002 Available at www.w3.org/TR/xquery/.
17. V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 313–324, Minneapolis, Minnesota, May 1994.
18. V. Christophides, S. Cluet, and G. Moerkotte. Evaluating Queries with Generalized Path Expressions. In *Proc. of ACM SIGMOD*,

pages 413–422, 1996.

19. V. Christophides, S. Cluet, and J. Siméon. On Wrapping Query Languages and Efficient XML Integration. In *Proceedings of ACM SIGMOD Conf. on Management of Data*, Dallas, TX., May 2000.
20. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your Mediators Need Data Conversion! In *Proceedings of ACM SIGMOD Conf. on Management of Data*, pages 177–188, Seattle, WA., June 1998.
21. The UDDI community. Universal description, discovery, and integration (uddi v2.0). Available at www.uddi.org/, October 2001.
22. D. Florescu D. Chamberlin, J. Robie. Quilt: An xml query language for heterogeneous data sources. In *WebDB'2000*, pages 53–62, Dallas, US., May 2000.
23. S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *W3C QL Workshop*, 1998.
24. L. Delcambre and D. Maier. Models for superimposed information. In *ER '99 Workshop on the World Wide Web and Conceptual Modeling*, volume 1727 of *Lecture Notes in Computer Science*, pages 264–280, Paris, France, November 1999. Springer.
25. A. Deutsch, M.F. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *Proceedings of the 8th Inter. World Wide Web Conference*, Toronto, 1999.
26. The ebXML community. Enabling a global electronic market (ebxml v.1.4). Available at www.ebxml.org/, February 2001.
27. M.F. Fernandez, D. Florescu, J. Kang, A.Y. Levy, and D. Suciu. System Demonstration - Strudel: A Web-site Management System. In *Proceedings of ACM SIGMOD Conf. on Management of Data*, Tucson, AZ., May 1997. Exhibition Program.
28. R. Fikes. DAML+OIL query language proposal, August 2001. Available at www.daml.org/listarchive/joint-committee/-0572.html.
29. 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.
30. P. Hayes. RDF Semantics. W3C Working Draft, 12 November 2002.
31. ICS-FORTH. The ICS-FORTH RDFSuite web site. Available at -139.91.183.30:9090/RDF, March 2002.
32. ISO. Information Processing-Text and Office Systems- Standard Generalized Markup Language (SGML). ISO 8879, 1986.
33. H. Jagadish, L. Lakshmanan, T. Milo, D. Srivastava, and D. Vista. Querying network directories. In *Proceedings of ACM SIGMOD Conf. on Management of Data*, pages 133–144, Philadelphia, USA, 1999. ACM Press.
34. G. Karvounarakis, V. Christophides, D. Plexousakis, and S. Alexaki. Querying RDF Descriptions for Community Web Portals. In *BDA'2001 (17iemes Journees Bases de Donnees Avances - French Conference on Databases)*, pages 133–144, Agadir, Morocco, 2001.
35. M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proceedings of the ACM SIGMOD Inter. Conference on Management of Data*, pages 393–402, 1992.
36. M. Kifer and G. Lausen. F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *Proceedings of ACM SIGMOD Conf. on Management of Data*, volume 18, pages 134–146, Portland, Oregon, June 1989.
37. L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. SchemaSQL - a language for interoperability in relational multi-database systems. In *Proceedings of Inter. Conference on Very Large Databases (VLDB)*, pages 239–250, Bombay, India, September 1996.
38. O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation. Technical report, , 1999.
39. J. Liljegren. Description of an RDF database implementation. Available at www-db.stanford.edu/~melnik/rdf/db-jonas.html.
40. D. Maier and L. Delcambre. Superimposed information for the internet. In *ACM SIGMOD Workshop on The Web and Databases Philadelphia, Pennsylvania, June 3-4*, pages

1–9, 1999.
41. M. Maloney and A. Malhotra. XML schema part 2: Datatypes. W3C Candidate Recommendation, October 2000.
42. M. Marchiori and J. Saarela. Query + metadata + logic = metalog. In *W3C QL Workshop*, 1998.
43. S. Melnik. Storing RDF in a relational database. Available at www-db.stanford.edu/~melnik/rdf/db.html.
44. L. Miller. RDF Query using SquishQL. Available at swordfish.rdfweb.org/rdfquery/, 2001.
45. I.S. Mumick and K.A. Ross. Noodle: A Language for Declarative Querying in an Object-Oriented Database. In *Proceedings of Inter. Conference on Deductive and Object-Oriented Databases (DOOD)*, pages 360–378, Phoenix, Arizona, December 1993.
46. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing Knowledge about Information Systems. *ACM TOIS*, 8(4):325–362, 1990.
47. Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. In *Proceedings of IEEE Inter. Conference on Data Engineering (ICDE)*, pages 132–141, New Orleans, LA., February 1996.
48. D. Plexousakis. Semantical and Ontological Considerations in Telos: a Language for Knowledge Representation. *Computational Intelligence*, 9(1):41–72, 1993.
49. Some proposed RDF APIs.
GINF: www-db.stanford.edu/~melnik/rdf/-api.html,
RADIX: www.mailbase.ac.uk/lists/rdf-dev/-1999-06/0002.html,
Netscape/Mozilla: lxr.mozilla.org/-seamonkey/source/rdf/base/idl/,
RDF4J: www.alphaworks.ibm.com/formula/-rdfxml/,
Jena: www-uk.hpl.hp.com/people/bwm/-RDF/jena,
Redland: www.redland.opensource.ac.uk/-docs/api.
50. A. Seaborne. RDQL: A Data Oriented Query Language for RDF Models. Available at www-uk.hpl.hp.com/people/afs/RDQL/, 2001.
51. M. Sintek and S. Decker. RDF Query and Transformation Language. Available at www.dfki.uni-kl.de/frodo/triple/, August 2001.
52. H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures. W3C Candidate Recommendation, October 2000. Available at www.w3.org/-TR/xmlschema-1/.
53. F. van Harmelen, P. Patel-Schneider, and I. Horrocks. Reference description of the DAML+OIL ontology markup language. March 2001.
54. Web service description language (WSDL). Available at www106.ibm.com/developer-works/library/ws-rdf, 2000.
55. M. Dean, D. Connolly, F. Van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L.A. Stein. OWL Web Ontology Language Reference Version 1.0 W3C Working Draft 12 November 2002.
56. OMG Unified Modeling Language Specification. Version 1.4 (2001) Available at: cgi.omg.org/docs/formal/01-09-67.pdf
57. A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis. Benchmarking RDF Schemas for the Semantic Web. In Proceedings of the First Inter. Semantic Web Conference (ISWC'02), Sardinia, Italy. June 9-12, 2002.
58. D. Brickley, R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft 12 November2002.

Greg Karvounarakis studied Computer Science at the Computer Science Department of Crete University, Heraklion, Greece and received his M.SC. in Information Systems & Software Technology in 2001. Since 2001 he is a research and development engineer in the Information Systems Laboratory of the Institute of Computer Science, Foundation for Research and Technology - Hellas (ICS-FORTH). His main research interests include Semantic Web and Digital Libraries, RDF/XML data models and query languages as well as Web services description and composition.

Aimilia Magkanaraki studied Informatics at the department of Applied Informatics at the University of Macedonia, Greece. She is a graduate student at the department of Computer Science at the University of Crete, Greece. Since 2000, she is a research assistant in the Information Systems and Software Technology Laboratory of the Institute of Computer Science, Foundation for Research and Technology, Hellas (ICS-FORTH). The research work for her master thesis is on the topic of view mechanisms for RDF metadata. Her main research interests include Semantic Web technologies, knowledge representation and information modeling, ontology storage and query languages.

Sofia Alexaki studied Computer Engineering & Informatics at the University of Patras, Greece (1998). She received her M.SC. in Information Systems & Software Technology from the University of Crete, Greece (2001). From 2001 till now, she is a research and development engineer in the Information Systems Laboratory of the Institute of Computer Science, Foundation for Research and Technology - Hellas (ICS-FORTH). Her main research interests include Semantic Web, RDF/XML data models and Web-based Information Systems.

Vassilis Christophides studied Electrical engineering at the National Technical University of Athens, Greece (NTUA). He received his DEA in computer science from the University PARIS VI and his Ph.D. from the Conservatoire National des Arts et Metiers (CNAM) of Paris, France. From 1992 until 1996 he did his research work at the French National Institute for Research in Computer Science and Control (INRIA) in the topics of Object Database and Documents Management Systems. Since 1997 he is researcher at the Information Systems & Software Technology Group of the Institute of Computer Science, Foundation for Research and Technology - Hellas (ICS-FORTH), leading several EU projects related to the integration of heterogeneous and distributed information sources. Since 2001 he is assistant professor at the Computer Science Department of Crete University, Heraklion, Greece. His main research interests include Semantic Web and P2P information management systems, semistructured and XML/RDF data models and query languages as well as description and composition languages for e-services. He has served as program committee member of many conferences, including ACM SIGMOD, VLDB and WWW and the newly established conference series on the Semantic Web.

Dimitris Plexousakis is an Associate Professor of Computer Science at the Computer Science Department of the University of Crete, Greece and a Researcher at the Information Systems Lab of the Institute of Computer Science of the Foundation of Research and Technology - Hellas. He holds a B.Sc. in Computer Science from the University of Crete and M.Sc. and Ph.D. degrees in Computer Science from the University of Toronto. Prior to joining the faculty of the University of Crete he was an Assistant Professor at Kansas State University and the University of South Florida. He is a member of ACM and IEEE. His research interests include data and knowledge base management, workflow and business process management systems, e-services, peer-to-peer and grid based systems, applications of AI in data management, metadata management and query languages for the Semantic Web. He is an active participant in the OntoWeb and Planet Networks of Excellence and in national and European projects on service integration in web-based information systems. He has served as program committee member of many conferences, including WWW and CAiSE and the newly established conference series on the Semantic Web.

Michel O. Scholl received his Ph.D. in computer science from UCLA, Los Angeles in 1976 and his French Thèse détat from the University of Grenoble in 1985. He was with INRIA for twelve years, where he headed the Verso database group, prior to joining the faculty of CNAM. Since 1989, he has been a full professor of computer science at CNAM. He manages the database group in the research laboratory CEDRIC of CNAM and had a joint position at INRIA in the Verso database group from 1989 until 2002. He has published in the fields of computer-assisted instruction, packet-switched radio networks, data structures and databases. His current research interest include spatial databases, image databases and the semantic web. He has served as program committee member of many conferences, including ACM SIGMOD, VLDB and EDBT.

Karsten Tolle studied Mathematics and Computer Science at the University of Hannover, Germany. He conducted his Master's Thesis on RDF Parsing and Validation in cooperation with the Information Systems & Software Technology Group of the Institute of Computer Science, Foundation of Research and Technology - Hellas (ICS-FORTH). Since 2000 he is a research assistant at the Databases and Information Systems group (DBIS) of the Johann Wolfgang Goethe-University of Frankfurt am Main, Germany. His main research interests include Semantic Web, RDF data validation and agent technologies.