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.

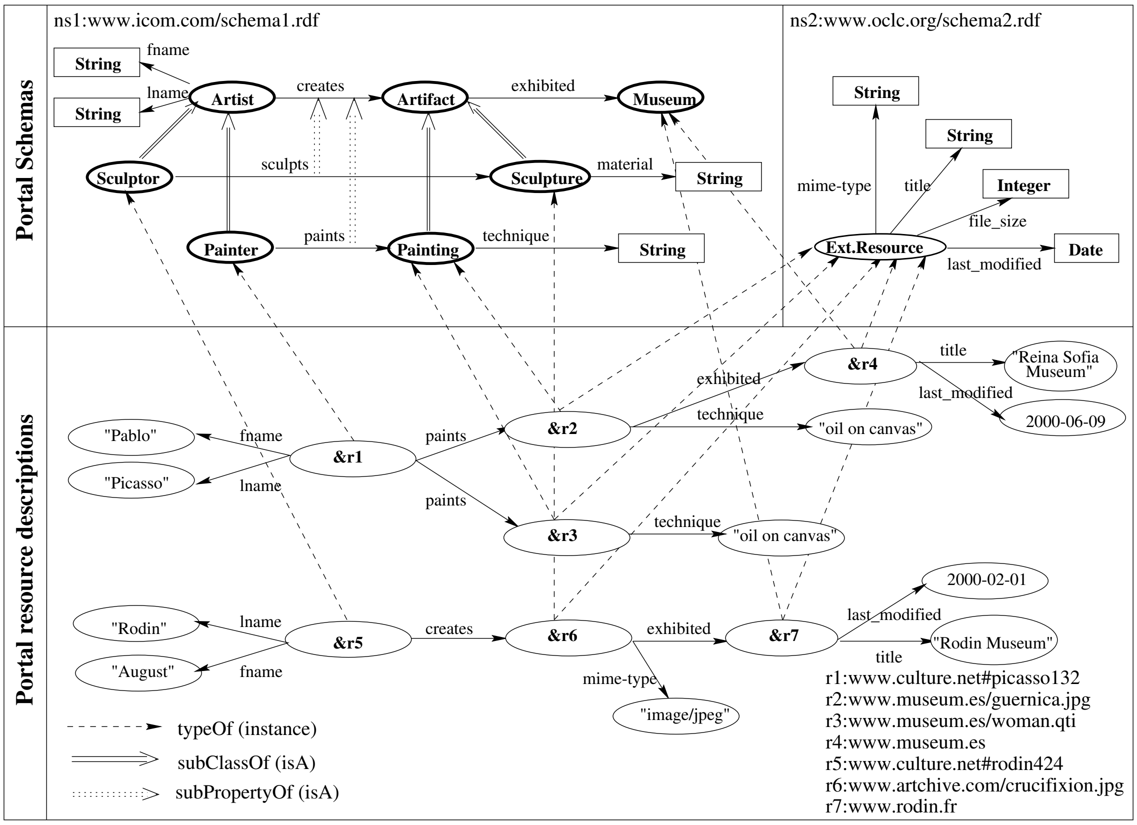


Figure 1. An example of RDF resource descriptions for a Cultural Portal

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] 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:

- $\mathcal{M} = \{m_1, m_2 \dots\}$: metaclass names
- $\mathcal{C} = \{c_1, c_2 \dots\}$: class names
- $\mathcal{P} = \{p_1, p_2 \dots\}$: property names
- $\mathcal{U} = \{u_1, u_2 \dots\}$: resource URIs
- $\mathcal{O} = \{o_1, o_2 \dots\}$: 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*.

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:*

---

[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).

- *for every node* $n \in V_D$*:*
  - ⋄ *if* $n \in U \Rightarrow \lambda(n) \subseteq C$ *(4.1)*
  - ⋄ *if* $n \in L \Rightarrow \lambda(n) \in Lt$ *(4.2)*
  - ⋄ *if* $n \in O \Rightarrow \lambda(n) \in Bt$ *(4.3)*
- *for every* $p \in E_D$ *from node* n *to node* n'*:*
  - ⋄ $\exists c \in \lambda(n), c \preceq domain(p) \wedge \exists c' \in \lambda(n'), c' \preceq range(p)$ *(4.4)* □

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 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, $\pi_c : C \to 2^U$, *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* $\pi_{M_c} : M_c \to 2^C$ and $\pi_{M_p} : M_p \to 2^P$. 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

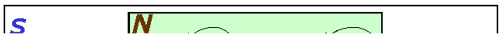
Taking advantage of the *RQL* types `T` and values `V`, we subsequently introduce formally the notions of a *description schema* and *base.*

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* $\sigma : N \to T$ □

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* $\omega : V_D \cup E_D \to V$, *such that:*

- *for every* $n \in V_D$, $\omega(n) \in [\![\sigma(\lambda(n))]\!]$ *(8.1)*


S
N