Querying RDF Descriptions for Community Web Portals[1]

Gregory Karvounarakis Vassilis Christophides Dimitris Plexousakis Sofia Alexaki

Institute of Computer Science,
FORTH, Vassilika Vouton, P.O.Box 1385, GR 711 10, Heraklion, Greece

E-mail : {gregkar, christop, dp, alexaki}@ics.forth.gr

Résumé

Les portails offrent à des communautés d'utilisateurs sur le Web ou à l'intérieur des entreprises, l'accès à des services de bibliothèque numérique, de syndication de contenu, ou aux services de fournisseurs intermédiaires d'information. Un portail communautaire repose sur un catalogue qui organise et décrit un ensemble de ressources d'information (services, documents ou données) s'adressant à une audience particulière : personnel d'une entreprise, personnes intéressées par un secteur d'activité, clientèle d'un marché électronique, etc. Un seul catalogue permet de décrire la même ressource d'information de plusieurs façons, offrant ainsi une souplesse de représentation bien plus importante que les bases de données traditionnelles. Dans cet article, nous proposons un langage déclaratif pour l'interrogation des catalogues des portails créés en utilisant RDF, une norme de description de ressources du Web proposée par le W3C. Ce langage, appelé RQL, se fonde formellement sur un modèle de graphes qui capture les primitives de modélisation de RDF et permet l'interprétation des descriptions des ressources à travers un ou plusieurs schémas superposés. RQL adapte les fonctionnalités des langages de requêtes semi-structurés aux particularités de RDF, mais il étend également ces fonctionnalités afin de permettre l'interrogation uniforme des descriptions de ressources et de leur(s) schéma(s) associé(s). RQL est utilisé dans plusieurs projets visant à la construction, l'accès et la personnalisation de portails communautaires.

Mots clés : *Web Communautaires, Portails, Reseaux Semantique, Données Semistructurées, Langages de Requêtes, RDF*

Abstract

Community Web Portals (e.g., digital libraries, vertical aggregators, infomediaries) have become quite popular nowadays in supporting specific communities of interest on corporate intranets or the Web. Portal Catalogs, organize and describe various information resources (e.g., sites, documents, data) for diverse target audiences (corporate, inter-enterprise, e-marketplace, etc.), in a multitude of ways, which are far more flexible than those provided by standard databases. In this paper, we propose a declarative language suitable for querying Portal Catalogs created according to the Resource Description Framework (RDF) W3C standard. Our language, called RQL, relies on a formal graph model, that captures the RDF modeling primitives and permits the interpretation of superimposed descriptions by means of one or more schemas. In this context, RQL adapts the functionality of semistructured query languages to the peculiarities of RDF but also extends this functionality in order to uniformly query both resource descriptions and related schemas. RQL is used in several projects aiming at building, accessing and personalizing Community Web Portals.

Keywords : *Community Webs, Portals, Semantic Networks, Semistructured Data, Query Languages, RDF*

1 Introduction

Information systems such as digital libraries, vertical aggregators, infomediaries, etc. are expected to play a central role in the 21st-century economy by enabling the development and maintenance of specific communities of interest (e.g., enterprise, professional, trading) on corporate intranets or the Web [28]. Such *Community Web Portals* essentially provide the means to select, classify and access various information resources (e.g., sites, documents, data) for diverse target audiences (corporate, inter-enterprise, e-marketplace, etc.). The core Portal component is a *Catalog* holding descriptions, i.e., *metadata*, about the available resources. In order to effectively disseminate community knowledge, Portal Catalogs assimilate and organize information in a *multitude of ways*, which are far more flexible than those provided by standard (relational or object) databases. Yet, in commercial software for deploying Community Portals (e.g., Epicentric, Plumtree, Glyphica[2]), querying is still limited to full-text (or attribute-value) retrieval and more advanced information-seeking needs is realized by navigational access. Furthermore, recent Web standards for describing resources are completely ignored.

The Resource Description Framework (RDF) standard [36, 10] proposed by W3C intends to facilitate the creation and exchange of resource descriptions between Community Webs. By considering community information as a collection of resources identified by URIs and by modeling resource descriptions using named properties, RDF enables the provision of various kinds of metadata (for administration, recommendation, content rating, intellectual property rights, site maps, push channels, etc.) about resources of quite diverse nature (ranging from PDF or Word documents, e-mail or audio/video files to HTML pages or XML data). The most distinctive feature of the RDF model is its ability to superimpose several descriptions for the same Web resources, enabling content syndication - and hence, automated processing - in a variety of application areas. To interpret these descriptions within or across communities, RDF al-

[2] www.epicentric.com, www.plumtree.com, www.glyphica.com respectively.

lows the definition of appropriate schema vocabularies [10]. Many content providers (e.g., ABCNews, CNN, Time Inc.), Web Portals (e.g., Open Directory, CNET, XMLTree[3]) or browsers (e.g., Netscape 6.0, W3C Amaya) already have adopted RDF, as well as, emerging application standards for Web data and services syndication (e.g., the RDF Site Summary [8], the Dublin Core [48], the Web Service Description Language [49], the Composite Capabilities/ Preference Profiles [14] or the Publishing Requirements for Industry Standard Metadata [44]). In a nutshell, the growing number of Web resources and the proliferation of description services, lead nowadays to large volumes of RDF metadata (e.g., the Open Directory Portal of Netscape comprises around 170M of Subject Topics and 700M of indexed URIs). It becomes evident that browsing such large *description bases* is a quite cumbersome and time-consuming task. Unfortunately, this is the only support provided by existing RDF systems [45].

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 query languages to the peculiarities of RDF but also extends this functionality in order to *uniformly query* both RDF descriptions and schemas. Thus, community members are able to query resources described according to their preferred schema, while discover, in the sequel, how the same resources are also described using another (sub-)community schema. This is quite useful when different Community Web Portals need to exchange descriptions about their resources. To illustrate our claims, we are using as a running example a Portal created for a cultural community (see Section 2). Then, we make the following contributions :

- In Section 3, we introduce a formal data model for *description bases* created according to the RDF Model & Syntax and Schema specifications [36, 10]. 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 (i.e., nodes).
- 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 [13]). Its functionality is illustrated for several categories of useful queries required by Community Web Portals. The novelty of RQL lies in its ability to smoothly switch between schema and data querying while exploiting - in a transparent way - the taxonomies of classes and properties, as well as, the multiple classification of resources.
- In Section 5, we describe the implementation of RQL on top of an object-relational DBMS (ORDBMS). We illustrate how RDF descriptions can be represented in an ORDBMS taking into account the related schemas and sketch how RQL queries are translated into SQL3. More precisely, we focus on the algebraic rewriting performed by the RQL optimizer to push the maximum of path expressions evaluation (involving both schema and data querying) to the underlying ORDBMS.

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

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

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 [36, 10] using as example a Portal Catalog created for a cultural community. 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, 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 Reference Conceptual Model[5]) or vocabularies (e.g., the Open Directory Topics[6]) and cannot always be extracted automatically from resource content or hyperlinks.

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). For example, `&r4` is first described as an `ExtResource` having two properties : `title` with value the string "Reina Sofia Museum" and `last_modified` with value the date 2000/06/09. Then, `&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 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`. 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 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

[4]Note that the complexity of semantic descriptions depends on the kind of resources (e.g., sites, documents, data) 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

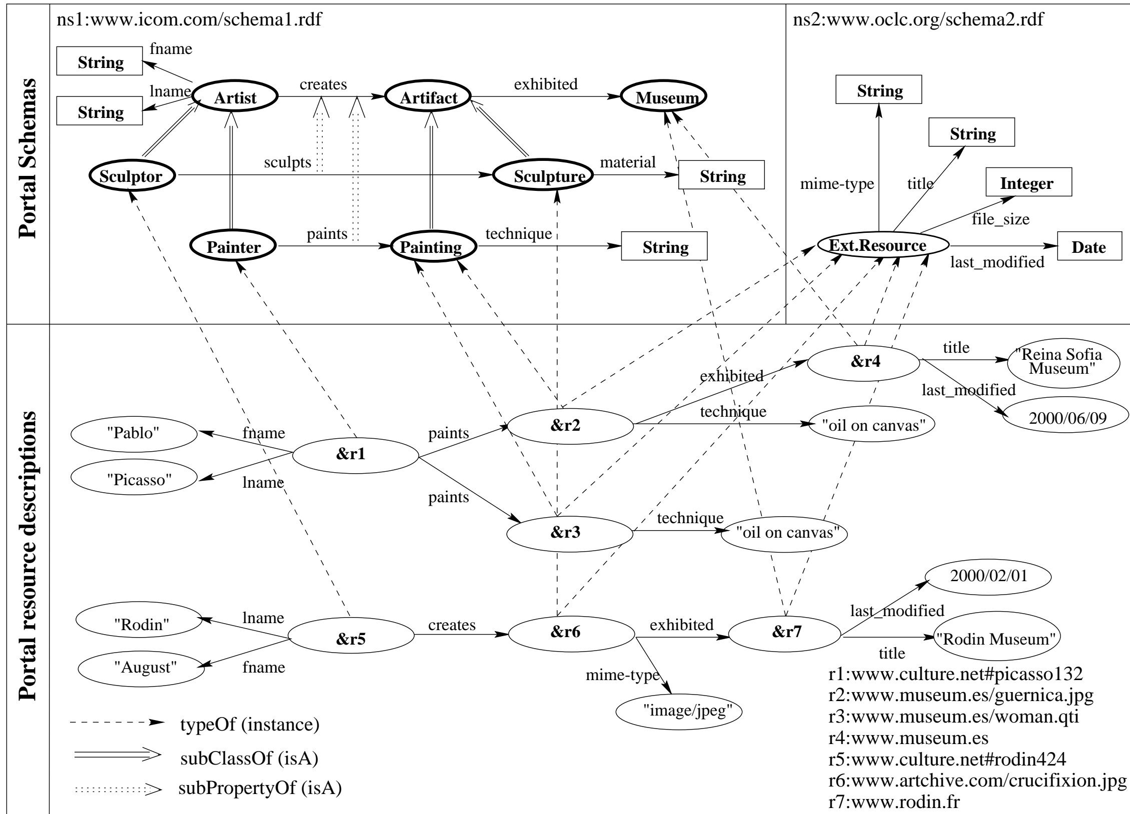

FIG. 1: An example of RDF resource descriptions for a Cultural Portal

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*[7] (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` needs to be created. However, in a Web setting, resources are usually described by various communities using their independently developed schemas.

These modeling primitives give us the flexibility we need to refine Portal schemas (e.g., by adding new taxonomies of names) and/or enrich descriptions (e.g., by adding new descriptions to the same resources) at any time, whilst they ensure the autonomy of description bases for different (sub-) communities. In this context, a Portal may comprise superimposed resources descriptions while preserving a conceptually unified view of its description base (i.e., its Catalog) through one or the union of all related RDF schemas.

2.1 RDF/S vs. Well-Known Data Models

The RDF modeling primitives are reminiscent to knowledge representation languages like TELOS [41, 43] as well as, to data models proposed for net-based applications such as Superimposed Information Systems [24, 37] and LDAP Directory Services [32, 7]. It becomes clear that RDF modeling primitives are substantially different from those defined in object or relational database models [4]. More precisely :

- *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 may belong to different classes* not necessarily pairwise related by specialization : the instances

[7] Conflicts between inherited properties are excluded due to the unique name assumption.

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 [42] or UnQL [11], 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 [9] or the more recent XML Schema proposal [46, 38] : 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 [31]. Last but not least, they can't distinguish between *entity labels* (e.g., `Artist`) and *relationship labels* (e.g., `creates`) and represent *unordered*, *optional*and *multi-valued* properties. As a consequence, query languages proposed for semistructured or XML data (e.g., LOREL [3], StruQL [27], XML-QL [26], XML-GL [15], Quilt [22] or recent XQuery [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 [35], XSQL [34], Noodle [40]).

Similar difficulties are also encountered in logic-based frameworks, which have been proposed for RDF manipulation. For instance, SiLRI [23] proposes some RDF reasoning mechanisms using F-logic. Although powerful, this approach does not capture the peculiarities of RDF : refinement of properties is not allowed (since slots are locally defined within the classes), container values are not supported (since it relies in a pure object model), while resource descriptions having heterogeneous types cannot be accommodated (due to strict typing). Moreover, Metalog [39] uses Datalog to model mainly RDF properties as binary predicates, while it 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.

3 A Formal Model for RDF

In most Community Web Portals, resources may have several descriptions for different syndication contexts, that can be easily and naturally represented in RDF [36] as *directed labeled graphs* whose nodes are called *resources* (or *literals*) and edges are called *properties*. Hence, RDF schemas [10] essentially define vocabularies of labels for graph nodes (called *classes* or *literal types*) and edges (i.e., properties). Both kinds of labels can be organized into taxonomies carrying inclusion semantics (i.e., class or property subsumption). More formally, each RDF schema uses a finite set of class C and property names P. Properties are then defined using class names or literal types so that : for each $p \in P$, $domain(p) \in C$ and $range(p) \in C \cup \mathcal{L}$, where $\mathcal{L}$ is a set of *Literal type names* like $string$, $integer$, $date$, etc. We denote by $H = (N, \prec)$ a hierarchy of class and property names, where $N = C \cup P$. H is *well-formed* if $\prec$ is a smallest partial ordering such that : if $p_1, p_2 \in P$ and $p_1 \prec p_2$, then $domain(p_1) \preceq domain(p_2)$ and $range(p_1) \preceq range(p_2)$. Our model guarantees that the *union of two well-formed hierarchies of names* used in an RDF schema *is always well-formed*.

A specific resource (i.e., node) together with a named property (i.e., edge) and its value (i.e., node) form a *statement* in the RDF jargon. Each statement can be represented by a *triple* having a *subject* (e.g., `&r1`), a *predicate* (e.g., `fname`), and an *object* (e.g., "Pablo"). The subject and object should be of a type compatible (under class specialization) with the domain and range of the predicate (e.g., `&r1` is of class `Painter`). Although not illustrated in Figure 1, RDF also supports structured values called *containers* for grouping statements, namely `rdf :Bag` (i.e., multi-sets) and `rdf :Sequence` (i.e., lists), as well as, higher-order statements (i.e., *reification*) which are not treated here. In the rest of the paper, the term *description base* will be used to denote a set of RDF statements and the term *description schema* to denote one or more *well-formed* hierarchies of RDF names used to label RDF descriptions.

3.1 RDF Typing System

As we have previously seen, RDF schemas (a) do not impose a strict typing on the descriptions (e.g., a resource may be liberally described using properties which are loosely-coupled with entity classes) ; (b) permit superimposed descriptions of the same resources (e.g., by classifying resources to multiple classes which are not necessarily related by subclass relationships) ; (c) can be easily extended to meet the description needs of specific (sub-)communities (e.g., through specialization of both entity classes and properties).

In this context, RDF data can be literals, resource URIs, and container values and the typing system foreseen by our model[8] is :

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

where τ_L is a literal type in $\mathcal{L}$, τ_U is the type for resource URIs[9], $\{.\}$ is the Bag type, $[.]$ is the Sequence type, and $(.)$ is the Alternative type. Alternatives capture the semantics of union (or variant) types [12], 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 type names is denoted by T.

This typing system allows us to capture containers with both homogeneous and heterogeneous member types (e.g., rep-

[8] Compared to the current status of the W3C RDF/S recommendation, our model provides a richer type system including several basic types as well as union types.

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

resenting n-ary relations returned by queries), as well as, to manipulate RDF schema classes and properties as self-existent individuals. For instance, *unnamed ordered tuples* denoted by $[v_1, v_2, \ldots, v_n]$ (where v_i is of some type τ_i) can be defined as heterogeneous sequences of type $[(\tau_1 + \tau_2 + \ldots + \tau_n)]$. Unlike traditional object data models, RDF classes are then represented as unary relations of the type $\{\tau_U\}$ while properties as binary relations of type $\{[\tau_U, \tau_U]\}$ (for relationships) or $\{[\tau_U, \tau_L]\}$ (for attributes). Finally, assignment of a finite set of URIs (of type τ_U) to each class name[10] is captured by a *population function* $\pi : C \rightarrow 2^U$. The set of all values foreseen by our model is denoted by V and the *interpretation* function $[\![.]\!]$ of types is defined in a straightforward manner.

3.2 RDF Description Bases and Schemas

Definition 1 *An RDF schema is a 5-tuple* $RS = (V_S, E_S, \psi, \lambda, H)$ *where :* V_S *is the set of nodes and* E_S *is the set of edges,* H *is a well-formed hierarchy of class and property names* $H = (N, \prec)$, λ *is a labeling function* $\lambda : V_S \cup E_S \rightarrow N \cup T$, *and* ψ *is an incidence function* $\psi : E_S \rightarrow V_S \times V_S$.

Note that the incidence and labeling functions are *total* in $V_S \cup E_S$ and E_S respectively. This does not exclude the case of schema nodes which are not connected through an edge. Additionally, we impose a *unique name assumption* on the labels of RS nodes and edges.

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

- *for each node* v *in* V_D*,* λ *returns a set of names* $n \in C \cup T$ *where the value of* v *belongs to the interpretation of each* n *:* $\nu(v) \in [\![n]\!]$*;*
- *for each edge* ϵ *in* E_D *going from node* v *to node* v'*,* λ *returns a property name* $p \in P$*.*

Note that the labeling function returns for atomic data nodes a literal type τ_L while for complex resource nodes returns one or more class names which may be defined in several well-formed hierarchies of names. Opposite to traditional object models all class names annotating resource nodes have a unique type τ_U. Additionally, integer labels (*1, 2, ...*) are used as property names by the members of RDF container values. Readers are referred to [5] for formal definitions of the imposed constraints.

4 The RDF Query Language : RQL

RQL is a typed query language relying on a functional approach (a la OQL [13]). 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*, featuring variables on labels for both nodes (i.e., classes) and edges (i.e., properties). As we will see in the sequel, the smooth combination of RQL schema and data path expressions is the key issue in order to satisfy the needs of several Community Web Portal applications (e.g., simple browsing, personalization, interactive querying, etc.).

To uniformly query nodes and edges either in RDF descriptions or in schemas, RQL blurs the distinction between schema labels (for classes, properties and types) and resource labels (i.e., URIs and literal values). In the rest of the paper we consider that both kinds of labels can be treated as strings and the interpretation of τ_U is extended as follows : $[\![\tau_U]\!] = T \cup C \cup P \cup U$. Abusing notation, we use τ_{UC} (τ_{UP}) to denote the type of class (property) names in schemas and τ_{UR} to denote only the URIs of resources in description bases. For the complete syntax, formal semantics and typing rules of RQL readers are referred to [33].

4.1 Browsing Portals

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 Community Web Portals. For instance, in Web Portals such as Netscape Open Directory, for each topic (i.e., class), one can navigate to its subtopics (i.e., subclasses) and eventually discover the resources which are directly classified under them. In this subsection, we show how the basic RQL queries can be used to generate such Portal interfaces, either off-line (i.e., by materializing the various query results in HTML/XML files) or online (by computing query answers on the fly).

To warmup readers, we start with queries which can find all the schema classes or properties used in a Portal Catalog as, by using the names `Class` and `Property`. In our example, these basic queries will return the URIs of the classes (of type τ_{UC}) and properties (of type τ_{UP}) illustrated in the upper part of Figure 1. Then, for a specific property we can find its definition by applying the corresponding `domain` (of type τ_{UC}) and `range` (of type τ_{UL} for attributes and τ_{UC} for relationships) functions. For instance, `domain(` *creates* `)` will return the class name *Artist*. To traverse the class/property hierarchies, RQL provides various functions such as `subClassOf` (for transitive subclasses) and `subClassOf^` (for direct subclasses). For example, the query `subClassOf^(` *Artist* `)` will return the class URIs *Painter* and *Sculptor*. More generally, we can access any RDF collection by just writing its name.

This is the case of RDF classes considered as unary relations : `Artist`. This query will return the 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 consider an extended class (or property) interpretation. Thus, RQL allows to query complex descriptions using only few abstract labels (i.e., the top-level classes or properties). This is motivated by the fact that names can be simply viewed as *terms* in an RDF schema vocabulary, and RQL offers a term expansion mechanism similar to that of thesauri-based information retrieval systems [30]. In order to

[10] Due to multiple classification we consider here a non-disjoint object id assignment to classes.

obtain the proper instances of a class (i.e., only the nodes labeled with the class URI), RQL provides the special operator ("`^`"). In our example, the result of `^Artist` will be the empty bag. Additionally, we can inspect the cardinality of class extents (or any other collection) using the `count` function.

It should be stressed that RQL considers as entry-points to an RDF description base not only the names of classes but also the names of properties. This is quite useful in several practical cases where Portal schemas may be composed of a) just property names (e.g., the Dublin Core Metadata Elements [48]) defined on the root class; b) large class hierarchies with only few properties defined between the top classes (e.g., when extending ontology concepts with thesauri terms [6]); or c) large property hierarchies resulting from the interconnection of several RDF schemas (e.g., when integrating different Metadata schemas [25]). In such cases, the labels of nodes may not be available or users may not be aware of them. Still, RQL allows one to formulate queries, as for example, `creates`. By considering properties as binary relations, the above query will return the bag of ordered pairs of resources belonging to the extended interpretation of creates (*source* and *target* are simple position indices) :

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

Common set operators applied to collections of the same type are also supported. For example, the query "Sculpture `intersect` ExtResource" will return 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. Note that the typing system of RQL permits the union of a bag of URIs with a bag of strings, but not between a class and a property extent (unary vs. binary relation). Besides class or property extents, RQL also allows the manipulation of RDF container values, such as Bag and Sequence. More precisely, the Boolean operator `in` can be used for membership test in any kind of collection and the operator `element` is used to extract the unique member of a singleton. Additionally, to access a member of a Sequence we can use the operator "`[ ]`" with an appropriate position index (or index range). If the specified member elements do not exist the query will return an empty sequence.

Finally, RQL supports standard Boolean predicates as =, <, > and `like` (for string pattern matching). All operators can be applied on literal values or resource URIs. It should be stressed that the latter case also covers comparisons between class or property URIs. For example, the condition "Painter < Artist" will return `true` since the first operand is a subclass of the second one. Disambiguation is performed in each case by examining the type of operands (e.g., literal value vs. URI equality, lexicographical vs. class ordering, etc.).

4.2 Personalizing Portal Access Channels

In order to personalize access to Community Web Portals, more complex RQL queries are needed. Portals personalization is actually supported by defining *information channels* to which community members may subscribe. Channels essentially preselect a collection of the Portal resources related to a theme, subject or topic (e.g., Museum Web sites) and they are specified using the recent RDF Site Summary (RSS) schema [8]. An RSS channel is specified by a static RDF/XML document containing the URIs of the resources along with some administrative metadata (e.g., titles, etc.). Not surprisingly, we can use RQL to define channels as views over the Portal Catalog and generate their contents on-demand.

In order to iterate over collections of RDF data (e.g., class or property extents, container values, etc.) and introduce variables, RQL provides a `select-from-where` filter. Given that the whole description base can be viewed as a collection of nodes/edges, *path expressions* can be used in RQL filters for the traversal of RDF graphs at arbitrary depths. Consider, for instance, the following query :

Q1 : *Find the resources having a title along with the property values.*

```
select  X, Y
from    {X}title{Y}
```

In the `from` clause we use a basic *data path expression* with the property name title. The node variables X and Y take their range restrictions from the *source* and *target* values of the title extent. As we can see in Figure 1, the title property has been defined with *domain* the class ExtResource but, due to multiple classification, X may be valuated with resources also labeled with any other class name (e.g., Artifact, Museum, etc.). Yet, in our model X has the unique type τ_{UR}, Y the literal type $string$, and the result of **Q1** is of type $\{[\tau_{UR}, string]\}$. The `select` clause defines, as usual, a projection over the variables of interest. Moreover, we can use "`select *`" to include in the result the values of all variables. This clause will construct an ordered tuple, whose arity depends on the number of variables. The result of the whole filter will be a bag. The closure of RQL is ensured by the basic queries supported for container values.

In order to define a channel with Museum resources available in our Cultural Portal we need to restrict the *source* values of the title extent to resources belonging to the class name Museum or any of its subclasses. To achieve this, we can formulate the following query exploiting superimposed resource descriptions (e.g., as ExtResources and Museums) :

Q2 : *Find the Museum resources and their title.*

```
select  X, Y
from    Museum{X}.title{Y}
```

Here $Museum\{X\}$ is also a basic data path expression where X ranges over the resource URIs in the extent of class Museum. The "`.`" used to concatenate the two path expressions is a *syntactic shortcut* for an implicit join condition between the *source* values of the title extent and X. Hence, **Q2** is equivalent to the query $Museum\{X\}, \{Z\}title\{Y\}$ where $X = Z$. Variables X, Z are of type τ_{UR}, Y is of type $string$ and the result of **Q2** will contain all the resources accessible by our Museum channel along with their titles (e.g., the site `www.museum.es` (&r4) with title "Reina Sofia Museum").

In addition, for each available resource (called *item*), channels usually provide a textual description of their information

content. This description can also be generated automatically by appropriate RQL queries. For instance, we can use the names of artists whose artifacts are exhibited in the Museums as descriptions of our channel items as follows :

```
select  Y, Z, V, R
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, reusing variable X for the source values of the other two path expressions simply denotes implicit joins between the extents of the properties *fname*/*lname* and *creates*, on their *source* values. Since the *range* of the *exhibited* property is the class *Museum* we don't need to further restrict the labels for the Y values in this query example.

Two remarks are noteworthy. First, RQL data path expressions may be liberally composed in the `from` clause of a query using node and edge labels, as for example, $Museum\{X\}.title\{Y\}$ (a class and a property name) or $\{X\}creates.exhibited\{Y\}$ (two property names). The "`.`" syntactic sugaring 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). In the above query, it implies a join between the extents of *creates* (including its subproperties *paints* and *sculpts*) and *exhibited* on their *target* and *source* values respectively. Recall at this point, that RDF classes do not define types on which attribute extractors like "`.`" could be defined and thus the expression "$Y.title$" is meaningless in out setting.

Second, 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 [3], StruQL [27], XML-QL [26], XML-GL [15], Quilt [22], 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.

4.3 Querying Portals with Large Schemas

The browsing interfaces actually supported by Portals like Netscape Open Directory, force users to navigate through the whole hierarchy of topics (i.e., classes) in order to find resources classified under the leaf topics. It is evident that for large Portal schemas this is a cumbersome and time consuming task (e.g., the Art hierarchy of the Open Directory alone contains 25000 subtopics and currently over 200000 indexed resources). Clearly, we also need declarative query support for navigating through the schema taxonomies of classes and properties. Consider, for instance, the following query :

Q3 : *Find the resources classified under a class more specific than Painter and more general than Neo-Impressionist which have created something.*

```
select  X, Y
from    {X :$Z}creates{Y}
where   $Z <= Painter and
        $Z >= Neo-Impressionist
```

In the from clause of **Q3** we can see a *mixed path expression* featuring both data (e.g., X) and schema variables on graph nodes (e.g., $\$Z$). More precisely, class variables prefixed by the symbol $ are implicitly range-restricted to `Class` (i.e., $Class\{Z\}$). Then, $\$Z$ is of type τ_{UC} and it will be valuated to the domain class of the property *creates* (i.e., `Artist`) and recursively to all of its subclasses (i.e., `Painter`, `Sculptor`, or `Neo-Impressionist`). The conditions in the `where` clause will in turn restrict $\$Z$ to the classes in the hierarchy having as superclass `Painter` and as subclass `Neo-Impressionist`. Naturally, without any restriction to $\$Z$ the whole extent of *creates* will be returned and $\$Z$ will be valuated to the actual classes of its *source* values. Note that, if the class in the `where` clause is not a valid subclass of the domain of *creates*, then the query will return an empty bag without accessing the *creates* extent. To make this kind of path expressions more compact for class equality (e.g., $\$Z = Painter$), shortcuts as "`{X :Painter}creates{Y}`" are also supported.

In other words, RQL extends the notion of generalized path expressions [17, 18, 3] to entire class (or property) inheritance paths. This is quite useful since resources can be multiply classified and several properties coming from different class hierarchies may be used to describe the same resources. Still RQL allows one to query properties emanating from resources according to a specific class hierarchy only, as we will see in the next example.

Q4 : *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	"R.Sofia Mus"
&r7	2000/06/09
&r4	2000/02/01

The mixed path expression of **Q4**, features a schema variable on graph edges (e.g., $@P$). More precisely, property variables, prefixed by the symbol @, are implicitly range-restricted to `Property` (i.e., $Property\{P\}$). Then, $@P$ is of type τ_{UP} and it will be valuated to all properties having as domain `ExtResource` or one of its superclasses. Finally, X, Y will be range-restricted for each successful binding of $@P$. The type of X is τ_{UR} while that of Y is the union of all the range types of `ExtResource` properties. According to the schema of Figure 1, $@P$ will valuated to *file_size*, *title*, *mime-type*, and *last_modified*, while Y will be of type $(integer + string + date)$. 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 [3].

4.4 Querying Portal Schemas

In this subsection, we focus our attention on querying RDF schemas, regardless of any underlying instances. The main motivation for this is to use RQL as a high-level language to implement schema browsing. This is quite useful when Portal Catalogs use large schemas (e.g., the Open Directory Topics hierarchy). Thus, to describe resources we first need to discover what classes and properties could be used. Consider for instance the query :

Q5 : *Find all the properties which specialize the property creates and may have as domain the class Painter along with their corresponding range classes.*

```
select   @P, $Y
from     { :Painter}@P{ :$Y}
where    @P <= creates
```

$@P$	$\$Y$
creates	Artifact
creates	Painting
creates	Sculpture
paints	Painting

The *schema path expression* in the from clause of **Q5** introduces two variables : $@P$ (of type τ_{UP}) ranging over `Property`, and $\$Y$ (of type τ_{UC}) ranging over the range class (and its subclasses) of each $@P$ valuation ($\$Y <= range(@P)$). Furthermore, $@P$ should be a subproperty of *creates* for which the domain is *Painter* or one of its superclasses. This expression is a shortcut for $\{: \$X\}@P\{: \$Y\}$ where $\$X = Painter$ and $\$X <= domain(@P)$. Given the schema of Figure 1, $@P$ will be valuated to the properties *creates* and *paints*. Due to class inheritance, *creates* may have as range any subclass of *Artifact*.

In cases where an automatic expansion of class hierarchies is not desired, RQL allows one to obtain only the classes which are directly involved in the definition of properties by using the functions `domain()` and `range()`. We conclude this subsection, with a query example illustrating how RQL schema paths can be composed to perform more complex schema navigations. It should be stressed that this kind of queries cannot be expressed in existing languages with schema querying capabilities (e.g., SchemaSQL [35], XSQL [34]).

Q6 : *What properties can be reached (at one step) from the range classes of creates.*

```
select   $Y, @P, $$Z
from     creates{ :$Y}.@P{ :$$Z}
```

$\$Y$	$@P$	$\$\$Z$
Artifact	exhibited	Museum
Painting	exhibited	Museum
Sculpture	exhibited	Museum
Painting	technique	string
Sculpture	material	string

In **Q6** the "`.`" notation implies a join condition between the range classes of the property *creates* and the domain of $@P$ valuations : $\$Y <= domain(@P)$. As we can see from the query result, this join condition will enable us to follow properties which can be applied (i.e., either directly defined of inherited) to any subclass of the *creates* range. Variable $\$\$Z$ will be valuated to the range classes (and their subclasses) or literal types of the previously returned properties. The prefix $$ is used to denote schema variables ranging over both class and type names (i.e., of type τ_{UCT}).

4.5 Putting it all Together

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 discover in the sequel, how the same resources are described using another schema. This functionality is required especially when different servers of Portal Catalogs (within or across communities) need to exchange resource descriptions. None of the existing query languages combines all the power of path expressions we provide with RQL. The following two scenarios depict this functionality.

In the first, we assume two servers for Portal Catalogs having different schemas (e.g., identified by two namespaces $ns1$ and $ns2$ shown in Figure 1) while sharing the same description base (e.g., in the case of sub-communities). Then we can combine schema information from the first server with data information from the second one, as illustrated in the next example :

Q7 : *Find the resources modified after 2000/01/01 which can be reached by a property applied to the class Painting and its subclasses.*

```
select   R, Y, Z
from     (select @P
          from { :$X}@P
          where $X <= Painting
         ){R}.{Y}last_modified{Z}
where    Z > 2000/01/01
```

R	Y	Z
exhibited	&r4	2000/06/09
exhibited	&r7	2000/02/01

In **Q7**, the nested query will return all the property names which are used by the first server and satisfy the filtering conditions (e.g., *exhibited*, *technique*). Then, the result of this nested query will serve to query the description base using the properties known only by the second server (e.g., *last_modified*). Here, the variable R (of type τ_{UP}) will iterate over the result of the nested query and the shortcut "`.`" implies the join condition : $target(R) = Y$ for each valuation of R. In other terms, we will obtain those resources modified after 2000/01/01 for which there exists an incoming edge with a (property) label returned by the nested query[11].

In the second scenario, the two Portal servers have both different description bases and schemas. Then one of them can send to the other the following query :

[11] Nested queries featuring existential and universal quantification are also supported.

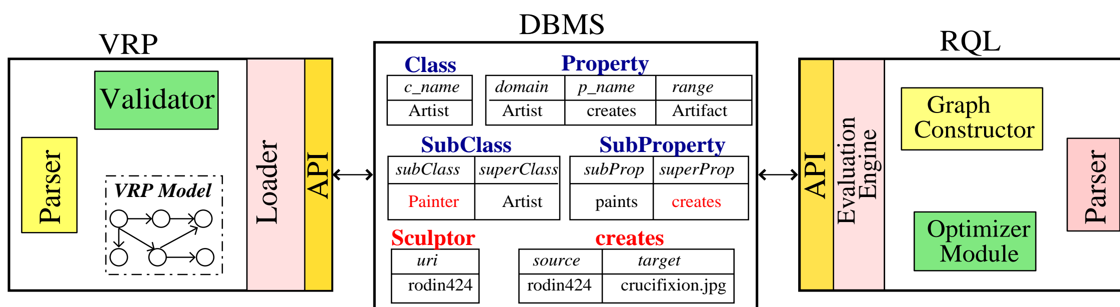

FIG. 2: The RQL Interpreter and Storage System

Q8 : *Tell me everything you know about the resources of the site "www.museum.es".*

```
select  X, $$Z, @P, Y, $$W
from    {X :$$Z}@P{Y :$$W}
where   Y like "www.museum.es*" or
        X like "www.museum.es*"
```

This query will iterate over all property names ($@P$). Then for each property it will iterate over its possible domain ($\$\$Z$) and range ($\$\$W$) classes or types, and, finally over the corresponding extents (X, Y). According to the example of Figure 1 the type of Y is the union ($\tau_{UC}+string+integer+date$) and the predicate `like` will be applied only on class names and strings. The final result of **Q8** is given below :

X	$\$\$Z$	$@P$	Y	$\$\$W$
&r1	Painter	paints	&r2	Painting
&r1	Painter	paints	&r3	Painting
&r2	Painting	exhibited	&r4	Museum
&r2	Painting	technique	"oil on canvas"	string
&r3	Painting	technique	"oil on canvas"	string
&r4	ExtResource	title	"R.Sofia Mus"	string
&r4	ExtResource	last_modified	2000/06/09	date

5 The RDF Schema Specific DataBase

The algebraic interpretation of RQL (see [33] for further details), relies on a relational representation of RDF description and schema graphs. Hence, we have implemented RDF storage and querying on top of an object-relational DBMS (ORDBMS), namely PostgreSQL[12]. The architecture of our RDF-enabled DBMS, called RSSDB, is illustrated in Figure 2. It comprises three main components : the RDF validator and loader (**VRP**), the RDF description database (**DBMS**) and the query language interpreter (**RQL**).

5.1 RDF Loading to an ORDBMS

We have implemented our *loader* as an extension of the *Validating RDF Parser* (VRP[13]) for analyzing, validating and processing RDF schemas and descriptions. Unlike other RDF parsers (e.g., SiRPAC[14]), VRP is based on standard compiler generator tools for Java, namely CUP/JFlex (similar to YACC/LEX). The stream-based parsing support of JFlex and the quick LALR grammar parsing of CUP ensure a good performance, when processing large volumes of RDF descriptions. The *VRP validation module* relies on an internal object representation, separating RDF schemas from their instances. This representation simplifies RDF metadata manipulation while it enables an *incremental loading* of RDF descriptions and schemas which is crucial for large volumes of RDF data (e.g., Netscape Open Directory exports 100M of class hierarchies and 700M of resource descriptions). The various loading methods have been implemented as member functions of the related VRP internal classes and communication with PostgreSQL relies on the JDBC protocol.

The core RDF/S model is represented in PostgreSQL by four tables, namely, `Class`, `Property`, `SubClass` and `SubProperty` which capture the class and property-type hierarchies defined in an RDF schema. Although not illustrated in Figure 2, we also consider a table `NameSpace` holding the namespaces of the RDF Schemas stored in the ORDBMS. The main goal of RSSDB is the separation of the RDF schema from data information, as well, as the distinction between unary and binary relations holding the instances of classes and properties. More precisely, class tables store the URIs of resources, while property tables store the URIs of the source and target nodes of the property (recall that all names are unique). Indices (i.e., B-trees) are constructed on all the attributes of the above tables. In other words, RSSDB relies on a schema specific representation of resource descriptions similar to the *attribute-based* approach proposed for storing XML data [29, 47].

Furthermore, RSSDB is flexible enough to allow the customization of the representation of RDF metadata in the underlying ORDBMS. This is important since no representation is good for all purposes and in most real-scale RDF applications variations of a basic representation are required to take into account the specific characteristics of 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 have a significant overhead on the response time of all queries (i.e., to find and open a table, its attributes, etc.).

One of the possible variations we have successfully experimented for the ODP Catalog [5] is the representation of all class instances by a unique table `Instances`. This table has two attributes, namely `uri` and `classid`, for keeping the uri's of the resources and the id's of the classes in which resources belong. Finally, an alternative variation could be the representation of properties with range a literal type, as

[12] www.postgresql.org
[13] www.ics.forth.gr/proj/isst/RDF
[14] www.w3.org/RDF/Implementations/SiRPAC

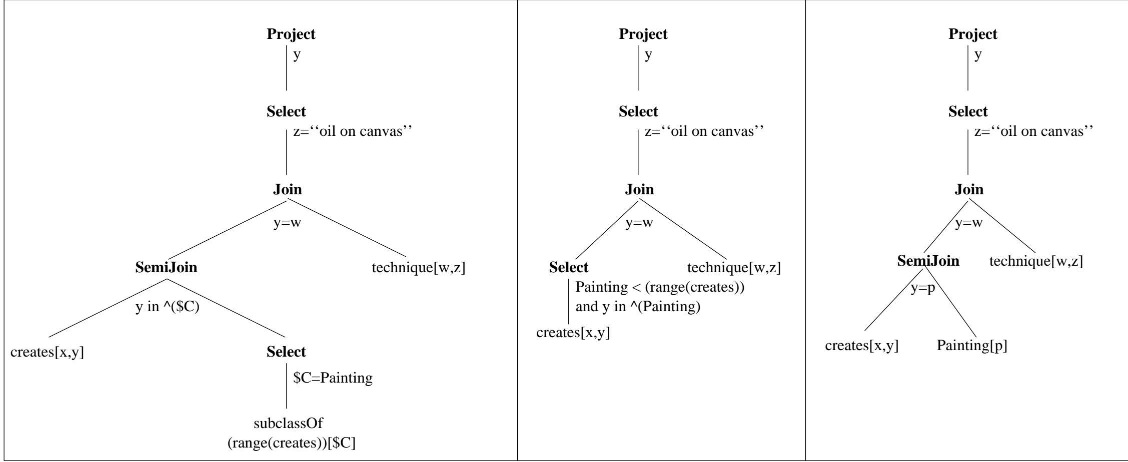

FIG. 3: Example of an RQL Query Optimization

attributes of the tables created for the domain of this property. Consequently, new attributes will be added to the created class tables. The tables created for properties with range a class will remain unchanged. The above representation is applicable to RDF schemas where attribute-properties are single-valued and they are not specialized.

5.2 RQL Query Processing

The *RQL interpreter* consists of (a) the parser, analyzing the syntax of queries ; (b) the graph constructor, capturing the semantics of queries in terms of typing and interdependencies of involved expressions ; and (c) the evaluation engine, accessing RDF descriptions from the underlying database. Since our implementation relies on a full-fledged ORDBMS like PostgreSQL, the goal of the *RQL optimizer* is to push as much as possible query evaluation to the underlying SQL3 engine (communication is based on libpq++, a JDBC-level, C++ PostgreSQL API). Then pushing selections or reordering joins to evaluate RQL path expressions is left to PostgreSQL while the evaluation of RQL functions for traversing class and property hierarchies relies on the existence of appropriate indices (see the last paragraph). The main difficulty in translating an entire RQL algebraic expression (expressed in an object algebra a la [21]) to a single SQL3 query is due to the fact that most RQL path expressions interleave schema with data querying [18]. Consider for instance the following query :

```
select  y
from    {x}creates{y :Painting}.technique{z}
where   z="oil on canvas"
```

During the query graph construction, the various shortcuts (e.g., "`.`") are expanded and variable dependencies are determined. The algebraic translation of the above query is illustrated in the left part of Figure 3. Data variables x, y (w, z) iterate over the extent of *creates* (*technique*) while the class variable $\$C$ iterates over all the subclasses of the range of *creates*. The mixed path expression of our example is translated to a semi-join between the extent of *creates* and its permitable range classes (y in ^$\$C$). Since in this query we are interested only in instances of the class *Painting* we can omit the second branch of the semi-join and rewrite the algebraic expression as illustrated in the middle part of Figure 3. The selection implies an existential condition over the extent of *Painting* whenever class *Painting* is a valid subclass of the range of *creates*. This test is performed once during graph construction and when successful the operation is equivalent to a semi-join over *creates* and *Painting* as illustrated in the right part of Figure 3. The final expression is translated into an SQL3 query as follows (the `*` indicates an extended interpretation of tables, according to the subtable hierarchy) :

```
select X.target
from   creates* X, technique* Y, Painting P
where  X.target = Y.source and X.target = P.uri
       and Y.target='oil on canvas'
```

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). For this, we need to replace class (or property) names by *ids* using an appropriate encoding system (e.g., Dewey, postfix, prefix, etc.) for which a convenient total order exists between the elements in the hierarchy. We are currently working on the choice of a such linear representation of node or edge labels allowing us to optimize queries that involve different kinds of traversals in a hierarchy (e.g., an entire subtree, a path from the root, etc.).

6 Summary and Future Work

In this paper, we presented a declarative language for querying Portal Catalogs, through a series of examples of increasing expressiveness requirements and complexity. We believe that we have illustrated the power of RQL generalized path expressions, subsuming the filtering capabilities (i.e., without any restructuring operators) of semi-structured and XML query languages. A detailed analysis of the evaluation complexity of the language is an ongoing effort. Furthermore, exhaustive performance tests of RQL for various use cases (as the Open Directory and our home made cultural Portal) are currently under elaboration taking into account different

variations of our relational representation, as well as, different encoding schemas for class and property hierarchies. Finally, we are also planning to study the problem of *updates* in RDF description bases as well as the *restructuring* capabilities (e.g., grouping) of RQL required by various Portal applications. As a matter of fact, RQL is a generic tool actually used by several EU projects (i.e., C-Web, MesMuses, Arion and OntoKnowledge[15]) aiming at building, accessing and personalizing Community Web Portals. An online demo of RQL for our Cultural Portal is available at the URL : `http ://139.91.183.30 :9090/RDF/RQL/`.

Acknowledgments We would like to thank Jeen Broekstra and Arjohn Kampman for their valuable comments on the RQL syntax, Jérôme Siméon for helping us to understand the subtleties between RDF and XML schema, Alain Michard for many fruitful discussions on Community Web Portals, and Michel Scholl and Manolis Koubarakis for carefully reading and commenting previous versions of this paper.

Références

[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. *JODL*, 1(1) :5–18, April 1997.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured Data. *JODL*, 1(1) :68–88, April 1997.

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

[5] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The RDFSuite : Managing Voluminous RDF Description Bases. In *Proceedings of the 2nd Int. Workshop on the Semantic Web*, Hong-Kong, May 2001. Available also at http ://-139.91.183.30 :9090/RDF/publications/.

[6] B. Amann, I. Fundulaki, and M.Scholl. Integrating Ontologies and Thesauri for RDF Schema Creation and Metadata Querying. *JODL*, 3(3) :221–236, 2000.

[7] S. Amer-Yahia, H. Jagadish, L. Lakshmanan, and D. Srivastava. On bounding-schemas for ldap directories. In *Proc. of the International Conf. on Extending Database Technology*, pages 287–301, Konstanz, Germany, March 2000.

[8] G. Beged-Dov, D. Brickley, R. Dornfest, I. Davis, L. Dodds, J. Eisenzopf, D. Galbraith, R. Guha, E. Miller, and E. van der Vlist. RSS 1.0 Specification Protocol. Available at http ://purl.org/rss/1.0, August 2000.

[9] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible markup language (XML) 1.0. W3C Recommendation, February 1998. Available at http ://www.w3.org/TR/REC-xml/.

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

[11] P. Buneman, S.B. Davidson, and D. Suciu. Programming Constructs for Unstructured Data. In *DBPL'95*, Gubbio, Italy, 1995.

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

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

[14] Composite Capabilities/Preference Profiles. Available at http ://www.w3.org/Mobile/CCPP, 2001.

[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 *Proc. of International World Wide Web Conf.*, Toronto, Canada, 1999.

[16] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0 : An XML Query Language. Technical report, World Wide Web Consortium, June 2001. Available at http ://www.w3.org/TR/xquery.

[17] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *SIGMOD'94*, pages 313–324, Minneapolis, Minnesota, May 1994.

[18] V. Christophides, S. Cluet, and G. Moerkotte. Evaluating Queries with Generalized Path Expressions. In *SIGMOD'96*, pages 413–422, Montreal, Canada, June 1996.

[19] V. Christophides, S. Cluet, and J. Siméon. On Wrapping Query Languages and Efficient XML Integration. In *SIGMOD'00*, Dallas, TX., May 2000.

[20] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your Mediators Need Data Conversion ! In *SIGMOD'98*, pages 177–188, Seattle, WA., June 1998.

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

[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 Query Languages Workshop*, Cambridge, Mass., 1998.

[24] L. Delcambre and D. Maier. Models for superimposed information. In *WCM'99*, pages 264–280, Paris, France, November 1999.

[25] L. Dempsey and R. Heery. DESIRE : Development of a European Service for Information on Research and Education, 1997. http ://www.ukoln.ac.uk/metadata/-desire/overview/rev_ti.htm.

[15] cweb.inria.fr, aquarelle.inria.fr/mesmuses, dlforum.external.forth.gr :8080, www.ontoknowledge.org, respectively.

[26] A. Deutsch, M.F. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *Proc. of International World Wide Web Conf.*, Toronto, 1999.

[27] M.F. Fernandez, D. Florescu, J. Kang, A.Y. Levy, and D. Suciu. System Demonstration - Strudel : A Web-site Management System. In *SIGMOD'97*, Tucson, AZ., May 1997. Exhibition Program.

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

[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, May 1999. Available at http ://www-caravel.inria.fr/dataFiles/GFSS00.ps.

[30] D.J. Foskett. Theory of clumps. In K. Sparck Jones and P. Willett, editors, *Readings in Information Retrieval*, pages 111–134. Morgan Kaufmann, 1997.

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

[32] H. Jagadish, L. Lakshmanan, T. Milo, D. Srivastava, and D. Vista. Querying network directories. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 133–144, Philadelphia, USA, 1999. ACM Press.

[33] G. Karvounarakis. The RQL Online Documentation. Available at http ://139.91.183.30 :9090/RDF/RQL/-Design.html, 2001.

[34] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 393–402, 1992.

[35] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. SchemaSQL - a language for interoperability in relational multi-database systems. In *Proc. of International Conf. on Very Large Databases (VLDB)*, pages 239–250, Bombay, India, September 1996.

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

[37] D. Maier and L. Delcambre. Superimposed information for the internet. In *WebDB'99*, pages 1–9, Philadelphia, PA, June 1999.

[38] M. Maloney and A. Malhotra. XML schema part 2 : Datatypes. W3C Recommendation, October 2000. Available at http ://www.w3.org/TR/xmlschema-2/.

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

[40] I.S. Mumick and K.A. Ross. Noodle : A Language for Declarative Querying in an Object-Oriented Database. In *Proc. of International Conf. on Deductive and Object-Oriented Databases (DOOD)*, pages 360–378, Phoenix, Arizona, December 1993.

[41] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos : Representing Knowledge about Information Systems. *ACM TOIS*, 8(4) :325–362, 1990.

[42] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. MedMaker : A Mediation System Based on Declarative Specifications. In *Proc. of IEEE International Conf. on Data Engineering (ICDE)*, pages 132–141, New Orleans, LA., February 1996.

[43] D. Plexousakis. Semantical and Ontological Considerations in Telos : a Language for Knowledge Representation. *Computational Intelligence*, 9(1) :41–72, 1993.

[44] Publishing Requirements for Industry Standard Metadata. Available at http ://www.prismstandard.org, 2001.

[45] 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.

[46] H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1 : Structures. W3C Recommendation, October 2000. Available at http ://www.w3.org/TR/xmlschema-1/.

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

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

[49] Web Service Description Language. Available at : www-106.ibm.com/developerworks/library/ws-rdf, 2000.