Declarative Languages for Querying Portal Catalogs

Vassilis Christophides Dimitris Plexousakis Greg Karvounarakis Sofia Alexaki
ICS-FORTH, Vassilika Vouton, P.O.Box 1385, GR 711 10, Heraklion, Greece
{christop, dp, gregkar, alexaki}@ics.forth.gr

1 Introduction

As data is increasingly captured, aggregated, and digitized worldwide, new types of information systems, such as digital libraries and information (subject) gateways, emerge as core technologies of the 21st-century economy. After a first generation of systems focusing on the accessibility of available information resources, nowadays, high quality information collections are smoothly transformed into *Community Web Portals*. These Portals provide the means to select, classify and access, in a semantically meaningful and ubiquitous way, diverse information resources in order to develop and maintain specific communities of interests (e.g., professional, trading, etc.) on corporate intranets or the Web. A key Portal component is the *Knowledge Catalog* holding descriptive information, i.e., *metadata*, about the community resources (e.g., sites, documents, data, etc.). Despite the current developments in standards for describing the content and meaning of information resources (see the W3C Metadata Activity[1]), declarative languages suitable for querying both their semantic descriptions and the employed schemas are still missing. In this paper we present such a high-level query language for Portal Catalogs (e.g., as Open Directory, CNET, XMLNews[2]) created according to the Resource Description Framework (RDF) standard [15, 4].

RDF [15] aims at facilitating the creation and exchange of metadata as any other Web data. RDF resource descriptions are represented as *directed labeled graphs* (where nodes are called *resources* or *literals* and edges are called *properties*) which can be serialized in XML. Furthermore, RDF schema [4] vocabularies are used to define the labels of nodes (called *classes*) and edges that can be used to describe and query resources in specific communities. These labels can be organized into appropriate taxonomies, carrying the inclusion semantics of subjects/topics in a Portal Catalog. In this context, our query language, called *RQL*, relies on a graph data model allowing us to interpret semistructured RDF descriptions by means of one or more RDF schemas. Note that RDF schemas (a) do not impose a strict typing on the data (by e.g., permitting multiple classification, optional and repeated properties); (b) can be easily extended (e.g., through specialization of both classes and property types); (c) may provide only a partial or overlapped interpretation of the underlying data (e.g., by having several, eventually incomplete schemas for the same resource descriptions); and (d) are not entirely separated from the resource descriptions (i.e., they can be queried like normal data). Thus, *RQL* shares the flexibility and utility of the recent proposals for semistructured or XML query languages, while, at the same time, extending their functionality to the RDF schema level by exploring in a *transparent way* the defined taxonomies of classes and properties, as well as, the multiple classification of resources. To the best of our knowledge, *RQL* is the first language to smoothly combine features from thesauri-based information retrieval systems (i.e., *term expansion mechanisms* [12]) with semistructured or XML query languages featuring variables on both property and class names (i.e., *generalized path expressions* [1]).

Our work is motivated by the fact that existing semistructured models (e.g., OEM [18], YAT [8]) cannot capture the semantics of node and edge labels provided by RDF schemas (i.e., taxonomies of classes and property types), while semistructured or XML query languages (e.g., LOREL [2], UnQL [5], StruQL [11], XML-QL [10], XML-GL [7]) are not suited to exploit RDF schema information (i.e., pattern vs. semantic matching of labels). On the other hand, database (relational or object) schema query languages as SchemaSQL [14], XSQL [13] or Noodle [17] fail to fully accommodate RDFS

[1]www.w3.org/Metadata
[2]www.dmoz.org, home.cnet.com, www.xmlnews.org

features such as specialization of properties. Furthermore, they have been designed for data that are more strongly typed than RDF resource descriptions. Finally, existing RDF-enabled systems [19] and logical languages (SiLRI [9], Metalog [16]) suffer from a number of defficiencies: (a) RDF schemas are either ignored or substituted by other formalisms (e.g., F-Logic); (b) RDF containers (e.g., bags, sequences) are not captured; (c) they do not consistently support complex querying on data and schema.

We believe that declarative query languages for metadata like *RQL*, open new perspectives for effective and efficient metadata management. Hence, Portal applications have to specify only which resources need to be accessed, leaving the task of determining how to efficiently access them to the Portal query engine. As a matter of fact, *RQL* is a generic tool that can be used by several applications aiming at building, accessing and personalizing Community Web Portals.

2 Example of a Cultural Community Web Portal

In this section, we briefly recall the main modeling primitives proposed in the Resource Description Framework (RDF) Model & Syntax and Schema (RDFS) specifications [15, 4]. Our presentation relies on an example of 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 management 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 can be constructed using existing ontologies (e.g., the International Council of Museums CIDOC Reference Conceptual Model[3]) or vocabularies (e.g., the Open Directory Topics hierachy[4]) employed by communities[5] and cannot always be extracted directly from resource content or hyperlinks.

In the lower part of Figure 1, we can see 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`). In the rest of the paper we use the notation `&` to denote the involved resource URIs (i.e., identity). For example, `&r4` is first described as an `ExtResource` having two properties: `title` with value the string "Reina Sophia 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 type `Painting` and has a property `exhibited` with value the resource `&r4` and a property `technique` with string value "oil on canvas". Resource `&r2` as well as `&r3` and `&r6` are also *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 type `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) an `fname` property with value "Pablo" and an `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 litteral types) and edges (i.e., properties) that can be employed to describe and query resources are defined in RDF schemas.

In the upper part of Figure 1, we can see two such schemas: the first intended for museum specialists while the second for Portal administrators. The scope of the declarations is determined by the corresponding *namespace* of each schema, e.g., **ns1** (`www.icom.com/schema1.rdf`) and **ns2** (`www.oclc.com/schema2.rdf`). For simplicity, we will hereforth omit the namespaces prefixing class and property names. In the former schema, the property `creates`, has been 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 and they have unique names. Furthermore, both classes and properties can be organized into taxonomies carrying simple 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,

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

[4]www.dmoz.org

[5]Note that the complexity of semantic descriptions depends on the diversity of resources and the breadth of community domains of discourse.

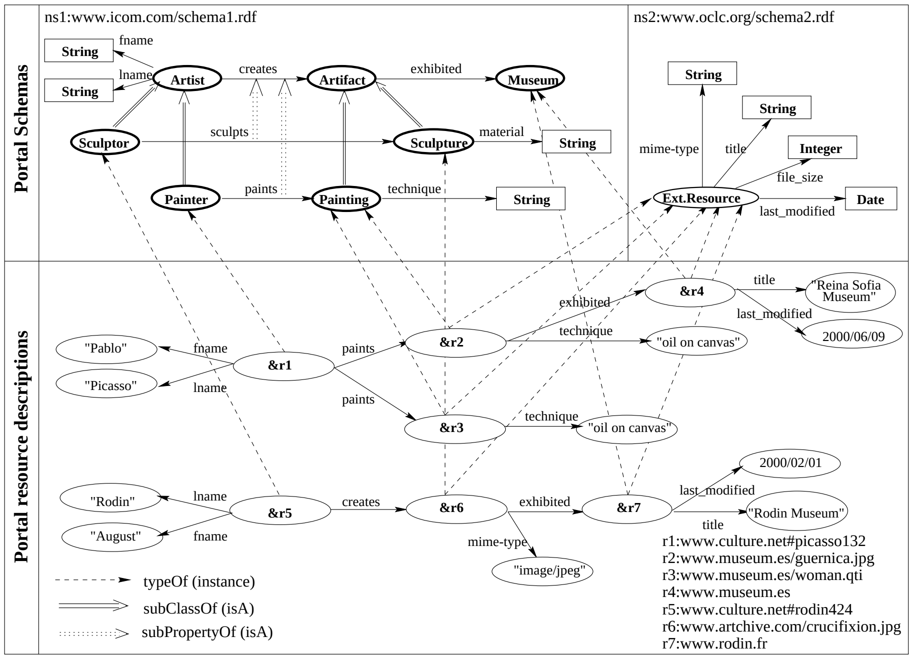

Figure 1: An example of RDF resource descriptions for a Cultural Portal

RDF properties are 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`).

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 is 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 used (e.g., `&r1` is of type `Painter`). In the rest of the paper, the term *description base* will be used to denote a collection of RDF statements. Although not illustrated in Figure 1, RDF also supports structured values called *containers* (i.e., bag, sequence) for grouping statements as well as higher-order statements (i.e., *reification*). Finally, both RDF graph schemas and descriptions can be serialized in XML using various forests of XML trees (i.e., there is not a root XML node).

3 The RDF Query Language: RQL

RQL is a typed query language relying on a functional approach (a la OQL [6]). It is defined by a set of basic queries and iterators which can be used to build new ones through functional composition of side-effect free functions. Furthermore, *RQL* supports generalized path expressions, featuring variables on labels for both nodes (i.e., classes) and edges (i.e., properties). The novelty of *RQL* lies in its ability to smoothly combine schema and data path expressions while exploiting - in a transparent way - the taxonomies of classes and properties as well as multiple classification of resources. As we will see in the sequel this functionality is required by several Community Web Portal applications (e.g., simple browsing, personalization, interactive querying, etc.).

3.1 Browsing Portals using RQL Basic Queries

The core *RQL* queries essentially provide the means to access RDF description bases with minimal knowledge of the employed schema(s). These queries can be used to implement a simple browsing interface for 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 will see 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. `Class` and `Property` are two basic queries which return all the labels of nodes and edges that can be used in an RDF description base. In our example, they will return the URIs of the classes and properties illustrated in Figure 1. Then, for a specific property we can find its definition by applying the corresponding `domain` and `range` 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 properties considered as binary relations. The basic query `creates` will return the bag of ordered pairs of resources belonging to the extent of *creates* (*source* and *target* are simple position indices):

source	*target*
www.culture.net#rodin424	www.artchive.com/crucifixion.jpg
www.culture.net#picasso132	www.museum.es/guernica.jpg
www.culture.net#picasso132	www.museum.es/woman.qti

We can observe that, in the extent of properties we consider the extents of their subproperties (e.g., *paints* and *sculpts*) as well. We believe that using only few abstract labels (i.e., the top-level classes or properties in an RDF schema) to query complex descriptions is an original feature of *RQL*. As we will show in the sequel, properties are the main building blocks for formulating *RQL* path expressions. Finally, common set operators (e.g., `union`, `intersect`) applied to collections of the same type are also supported.

3.2 Personalizing Portal Access using RQL Filters

In order to personalize access to Community Web Portals, more complex *RQL* queries are needed. Portal 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 [3]. An RSS channel is specified by a static 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. For instance, the following query is used to define a channel with Museum resources available in our Cultural Portal.

Q1: *Find the Museum resources and their title.*

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

Here $Museum\{X\}$ is a basic *data path expression* where X ranges over the resource URIs in the extent of class *Museum* and Y over the 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, the source values of *title* may be resources also labeled with any other class name (e.g., *Artifact*, *Museum*, etc.). Then in **Q1** we essentially ignore the schema classes labeling the endpoint instances of the properties. The "." used to concatenate the two path expressions is just a *syntactic shortcut* for an implicit join condition between the *source* values of the *title* extent and X. Hence, **Q1** is equivalent to the query $Museum\{X\}, \{Z\}title\{Y\}$ where $X = Z$. Recall that RDF classes do not define types on which attribute extractor operators like "." could be defined and therefore the expression $X.title$ is meaningless in out setting. The final result will contain the sites `www.museum.es` and `www.rodin.fr` along with their tiltes.

3.3 Querying Portals with Large Schemas

In the previous subsections we have illustrated how *RQL* can be used to specify, in a declarative way, the access functionality actually supported by Portals like Netscape Open Directory. However, such simple browsing interfaces force the user 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 20000 subtopics and currently 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:

Q2: *Find the resources of a type 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 **Q2** 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`. Then, $\$Z$ 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 extent of *creates*. 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.

3.4 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 Topic hierachy) to describe resources. In this context, Portal administrators may not be aware of all the classes and properties they can use to describe resources while RDF schemas carry information which is only implicitly stated (e.g., the polymorphism of the domain and range of properties). Consider for instance the following query:

Q3: *Find all the properties which specialize the property creates and may have as domain the class Painter along with their corresponding domain and 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 **Q3** introduces two variables: $@P$ ranging over `Property`, and $\$Y$ 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 just 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*. The same is true for the range classes of *paints*. 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. We can issue, for instance, the following query:

Q4: *For all the classes in the hierachy rooted at Artist find the properties and their range classes which are directly defined.*

```
select  domain(@P), @P, range(@P)
from    Property{@P}
where   domain(@P) <= Artist
```

domain(@P)	@*P*	range(@P)
Artist	creates	Artifact
Artist	fname	string
Artist	lname	string
Painter	paints	Painting
Sculptor	sculpts	Sculpture

Compared to **Q3** the result of this query will contain only the classes for which a property is explicitly defined, along with its name and range. Note also that the functional nature of *RQL* allows the use of functions anywhere in a filter as long as typing rules are respected: `domain(@P)` is of type class name, as also is the name *Artist*.

References

[1] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Siméon. Querying Documents in Object Databases. *Inter. Journal on Digital Libraries*, 1(1):5–18, April 1997.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured Data. *Inter. Journal on Digital Libraries*, 1(1):68–88, April 1997.

[3] 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. Draft, August 2000.

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

[5] P. Buneman, S.B. Davidson, and D. Suciu. Programming Constructs for Unstructured Data. In *Proceedings of International Workshop on Database Programming Languages*, Gubbio, Italy, 1995.

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

[7] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a Graphical Language for Querying and Restructuring XML Documents. In *Proceedings of International World Wide Web Conference*, Toronto, Canada, 1999.

[8] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your Mediators Need Data Conversion! In *Proceedings of ACM SIGMOD Conf. on Management of Data*, pages 177–188, Seattle, WA., June 1998.

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

[10] A. Deutsch, M.F. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *Proceedings of the 8th International World Wide Web Conference*, Toronto, 1999.

[11] M.F. Fernandez, D. Florescu, J. Kang, A.Y. Levy, and D. Suciu. System Demonstration - Strudel: A Web-site Management System. In *Proceedings of ACM SIGMOD Conf. on Management of Data*, Tucson, AZ., May 1997. Exhibition Program.

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

[13] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 393–402, 1992.

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

[15] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, February 1999. Available at http://www.w3.org/TR/REC-rdf-syntax.

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

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

[18] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, pages 251–260, Taipei, Taiwan, March 1995.

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