Generating On the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL)[1]

Nikos Athanasis[1&2], Vassilis Christophides[1&2], and Dimitris Kotzinos[2]

[1] Institute of Computer Science, Foundation for Research and Technology – Hellas,
P.O. Box 1385, 71110 Heraklio, Greece
{athanasi, christop}@ics.forth.gr

[2] Department of Computer Science, University of Crete,
P.O. Box 2208, 71110 Heraklio, Greece
{kotzino}@csd.uoc.gr

Abstract. Building user-friendly GUIs for browsing and filtering RDF/S description bases while exploiting in a transparent way the expressiveness of declarative query/view languages is vital for various Semantic Web applications (e.g., e-learning, e-science). In this paper we present a novel interface, called GRQL, which relies on the full power of the RDF/S data model for constructing on the fly queries expressed in RQL. More precisely, a user can navigate graphically through the individual RDF/S class and property definitions and generate transparently the RQL path expressions required to access the resources of interest. These expressions capture accurately the meaning of its navigation steps through the class (or property) subsumption and/or associations. Additionally, users can enrich the generated queries with filtering conditions on the attributes of the currently visited class while they can easily specify the resource's class(es) appearing in the query result. To the best of our knowledge, GRQL is the first application-independent GUI able to generate a unique RQL query which captures the cumulative effect of an entire user navigation session.

1 Introduction

Semantic Web (SW) technology and in particular the Resource Description Framework (RDF) [12] and its Schema definition language (RDFS) [3] are increasingly gaining acceptance by modern web applications (e.g., e-learning, e-science) striving to describe, retrieve and process large volumes of information resources spread worldwide. In this context, several declarative languages for querying and specifying views over RDF/S description bases have been proposed in the literature such as RQL [12], and RVL [15] (see [14] for an extensive comparison of SW QLs). However, these languages are mainly targeting experienced users, who need to understand not only the RDF/S data model but also the syntax and the semantics of each language in order to formulate a query or a view in some textual form. Building user-friendly GUIs for browsing and filtering RDF/S description bases while exploiting in a transparent way the expressiveness of declarative languages like RQL/RVL is still an open issue for emerging SW technology.

The idea of using suitable visual abstractions in order to access data collections originates from the early days of the database systems. Commercial systems like the IBM QBE [21] and Microsoft Query Builder [16], as well as research prototypes like PESTO [8], IVORY [5], Lorel's DataGuide-driven GUI [10], BBQ [17], QURSED

[1] This work was partially supported by the EU Projects MesMuses (IST-2000-26074) and Selene (IST-2001-39045).

[18], and XQBE [2] are just few examples of graphical front-ends to relational, object or semistructured/XML databases assisting end-users in formulating queries to the underlying language (e.g., SQL, OQL, or XPath/XQuery) of the DBMS. These visual querying interfaces essentially rely on a graphical representation of the schema constructs (i.e., relations, object classes, XML elements/attributes) supported in each data model along with appropriate end-user interactions allowing to construct custom views and display their results in a default fashion. They are usually implemented as standalone applications targeting experienced users who need to understand not only the schema of the underlying database but also a great part of the functionality of the supported query language (i.e., joins, projections, etc.).

In the context of Semantic Web Portals, several browsing interfaces have been proposed (like RDF Distillery [9], OntoWeb [11], OntoPortal [4], ODESeW [7]) for accessing ontologies and their related information resources. These visual browsing interfaces offer a graphical view of the entire ontology as a tree or a graph of related classes and properties where users can either access directly the classified resources or formulate (attribute-value) filtering queries. They are usually implemented as Web applications for the "general public" using the programming APIs supported by the underlying ontology repository. As a result of their API limitations, the resources accessed each time correspond only to the most recently performed browsing action, regardless of the navigation paths traversed by a user. Thus, SW browsing interfaces ignore the class associations followed by users when visiting one class after the other.

The graphical interface presented in this paper aims to combine the simplicity of SW browsers with the expressiveness of DB query generators in order to navigate the Semantic Web by generating queries on the fly. The proposed interface, called GRQL, relies on the full power of the RDF/S data model for constructing, in a transparent to the end-user way, queries expressed in a declarative language such as RQL. More precisely, after choosing an entry point, users can discover the individual RDF/S class and property definitions and continue browsing by generating at each step the RQL path expressions required to access the resources of interest. These path expressions capture accurately the meaning of its navigation steps through the class (or property) subsumption and/or associations. Additionally, at each navigation step users can enrich the generated queries with filtering conditions on the attributes of the currently visited class while they can easily specify which class of resources should be finally included in the query result. To the best of our knowledge, GRQL is the first application-independent graphical interface able to generate a unique RQL query which captures the cumulative effect of an entire user navigation session. This functionality is vital for Semantic Web Portals striving to support conceptual bookmarks and personalized knowledge maps build on demand according to user's navigation history [19].

The rest of the paper is organized as follows. Section 2 presents the graphical generation of RQL queries using different navigation scenarios related to an example RDF/S schema. Section 3 details the implementation of our generic query generation algorithm including the internal representation model of RDF/S schemas and RQL queries, as well as the GUI demonstrating the intuitive use of GRQL by non-expert users. Finally, Section 4 summarizes our contributions and outlines our future work.

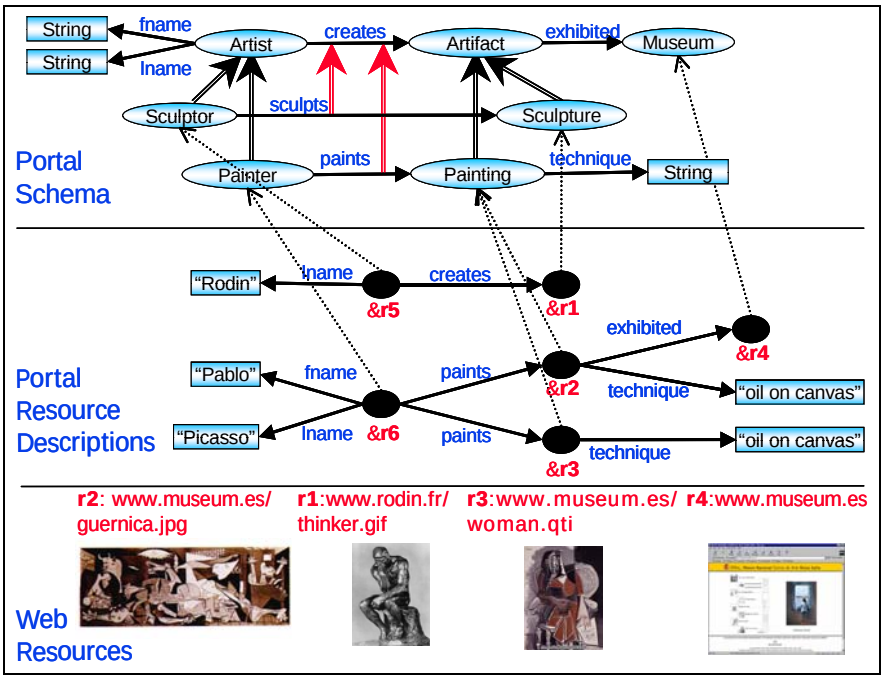


Fig. 1. RDF/S Descriptions of Web Resources in a Cultural Portal

2 From RDF/S Schema Browsing to RQL Queries

In this section we will present our graphical RQL interface (GRQL) and in particular the translation of the browsing actions performed by a user on the RDF/S class (or property) subsumption and/or associations to appropriate RQL path expressions and filters. To this end, we will use a fairly well known example schema originally employed in [12] to present the functionality of RQL.

Fig. 1 illustrates the RDF/S descriptions of four available resources: a sculpture image (resource identified as *r1*), two painting images (resources identified as *r2* and *r3*) and a museum web site (resource identified as *r4*). These descriptions have been created according to an RDF/S schema for cultural applications comprising three top-level classes, namely *Artist*, *Artifact* and *Museum*, which are associated thought the properties *creates* and *exhibited*. Given the flexibility of the RDF/S data model, both classes and properties can be specialized through subsumption relationships. In our example, the class *Artist* is specialized to *Sculptor* and *Painter* while class *Artifact* to *Sculpture* and *Painting*. In the same way the property *creates* (with domain the class *Artist* and range the class *Artifact*) is specialized to *sculpts* (with domain the class *Sculptor* and range the class *Sculpture*) and *paints* (with domain the class *Painter* and range the class *Painting*). Properties *fname, lname,* and *technique* play the role of attributes and range over the literal type *string* (or more generally an XML Schema data type [1, 20]). Then, resource *r2* can be described as an instance of the class

Painting with an attribute *technique* having as value *"oil on canvas"* and it is associated with resource *r4* though the property *exhibited*.

It is fairly obvious that users of our cultural portal need to navigate through the schema in order to access the corresponding resource descriptions. Nevertheless this navigation should take place without having to learn an RDF query language and construct the necessary queries by themselves. Furthermore, they want to distinguish *Artifact* resources for which information about their creator is available (e.g., *r1, r2, r3)* from *Artifact* resources having additional information about their exhibiting museum (e.g., *r2*). In other words, as in the case of static hyperlinks between web pages, users should be able to browse the Semantic Web in order to access resources according to the navigation paths traversed in the conceptual space defined by an RDF/S schema. This is a non-trivial task given that users can arbitrary browse back and forth through both the subsumption relationships of classes (or properties) and the associations of classes captured by properties. In the rest of this section, we will detail how the translation algorithm of the GRQL interface supports the desired functionality.

2.1 Browsing class associations

GRQL provides a tree-shaped graphical representation of the class (or property) subsumption relationships defined in an RDF/S schema, as for instance, the cultural example of **Fig. 1**. Then, users can choose one of the available classes (or properties) to begin their navigation session, while the definition of the selected class (i.e., its subclasses, attributes and associations) is displayed in order to enable them to browse further the DAG of class associations[2]. This elementary browsing action is translated into an appropriate RQL query, which essentially specifies a dynamic view over the underlying resource descriptions according to the performed schema navigation. **Table 1** depicts the choice of the class *Artist* as entry point, as well as the generated RQL query returning all resources classified under *Artist* and recursively its subclasses (i.e., resources *r5* and *r6*). At each browsing step, GRQL displays the path(s) of classes and properties visited insofar during a navigation session. In this way, GRQL not only informs users about their navigation history but also provides the ability to return to a previous browsing step. As we will see later on in this paper, this is useful for specifying which class of resources will be finally included in the constructed view, as well as for navigating through multiple schema paths (see subsection **2.3**).

Table 1. Choice of class *Artist* as entry point to the schema navigation

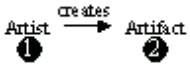
Navigation	RQL query	Path
	`select X1 from Artist{X1}`	`Artist`

After displaying the definition of the class *Artist* the user can access the resources of other associated classes, as for instance, the class *Artifact* (through the *creates*

[2] Cyclic properties, having as domain and range the same class, are not considered in this paper.

relationship). Now the user's view is extended with property *creates* relating instances of the class *Artist* with instances of the class *Artifact*. **Table 2** shows the performed browsing action, the generated RQL query and the corresponding navigation path. Note that the path expression *creates* in the from clause of the RQL query considers resources associated by this property and any of its subproperties such as *paints* and *sculpts* (i.e., *Artifact* resources *r1*, *r2* and *r3*). By default, the resources included in the final result i.e., the select clause of the RQL query, correspond to the current class (or property) selection in the navigation path, as in this example the range class of the property *creates*.

Table 2. Navigation to associated class *Artifact* through property *creates*

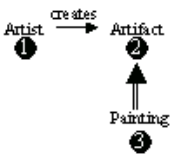
Navigation	RQL query	Path
	`select X2 from {X1}creates{X2}`	`Artist→creates→Artifact`

In the same manner, users can continue their navigation from class *Artifact* to *Museum* through the property *exhibited* and the available view is now specified by the following RQL query: `select X3 from {X1}creates{X2}.exhibited{X3}`. In this example, the constructed view comprises the *Museum* resource *r6*.

2.2 Browsing subsumption relationships

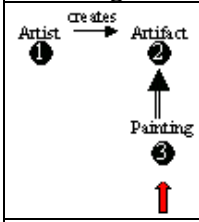
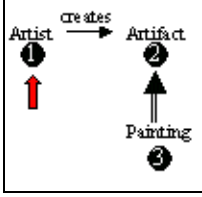
In addition, GRQL gives the ability to interleave at any step browsing on class associations with browsing on class (or property) subsumption relationships. For instance, after the navigation illustrated in **Table 2** a user may choose to visit the subclass *Painting* of *Artifact*. As is shown in **Table 3**, this browsing action has the effect to narrow the user's view only to *Painting*(s) created by an *Artist*. Compared to the RQL query of **Table 2,** the path expression in the from clause has been now accordingly modified to `{X1}creates{X2;Painting}` where the range of the property instead of the default *Artifact* class (and thus initially omitted) is now restricted to *Painting*.

Table 3. Navigation to specialized classes

Navigation	RQL query	Path
	`select X2 from {X1}creates{X2;Painting}`	`Artist→creates→Painting`

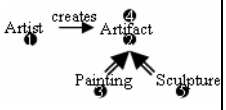
Then, users can either continue browsing through more specialized subclasses, if available, or visit other associated classes of class *Painting*. Note that RDF/S subclasses may have additional associations and attributes, as for instance, *technique*. The user can alternatively return to a previous step in the navigation path by clicking on the name of an already visited class or property (or more generally by clicking the *back* button of a Web browser).

Table 4. Navigating back to an already visited class

Navigation	RQL query	Path
	`select X2 from {X1}creates{X2;Painting}`	`Artist→creates→Painting`
	`select X1 from {X1}creates{X2;Painting}`	`Artist→creates→Painting`

As shown in **Table 4** (solid arrows represent the current position of the user in a navigation path), this browsing action affects only the target class of resources, which are going to be included in the user view i.e., the select clause of the generated RQL query. For example, while the first scenario considers in the view resources of the class *Painting* (represented by variable *X2*), the second scenario returning back to the initial navigation step, considers resources of the class *Artist* (represented by variable *X1*). It should be stressed that, although the user visits the same class *Artist* in both the initial and the final browsing step, the constructed views are different (see **Tables 4**, **1**), since they capture the effect of a sequence of performed browsing actions.

Table 5. Navigating to sibling subclasses in a path

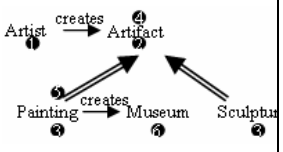
Navigation	RQL query	Path
	`select X2 from {X1}creates{X2;Painting}, Sculpture{X2}`	`Artist→creates→Painting` `Sculpture`

2.3 Browsing multiple navigation paths

If a user revisits the class *Artifact* from the scenario of **Table 4**, the constructed view remains the same as illustrated in **Table 3**. This ensures that the GRQL interface does not generate redundant RQL queries in the case where a user loops in the navigation path. However, since the definition of class *Artifact* is made available once more, the user may choose to visit another subclass such as *Sculpture*. In this scenario, the view contains the *Artist*(s) and their *Painting*(s) computed previously, which are also *Sculpture*(s). As it can be seen in **Table 5** this browsing action is translated into a conjunction of two RQL path expressions, given that the user was firstly interested to *Painting*(s) and then to *Sculpture*(s). Note that conjunctive path expressions in the from clause of an RQL query share common resource variables

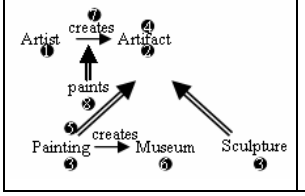
(e.g., variable X2). In this way GRQL interface captures the fact that the same URL may be classified under several unrelated through subsumption classes. The class of resources included in the final result of the view corresponds, as usual, to the current position of the user in the navigation path(s), but there is also the ability to obtain all the resources (e.g., represented by variables X1, X2 and X3) involved in its navigational view by selecting the special *"Include All"* GRQL button.

Table 6. Navigating to associated classes that belong to different navigation paths

Navigation	RQL query	Path
	`select X3 from {X1}creates{X2;Painting}. {;Painting}exhibited{X3}, Sculpture{X2}`	`Sculpture Artist→creates→Painting→ →exhibited→Museum`

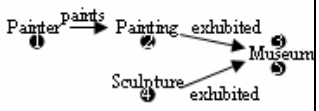
Users may in turn decide to revisit the class *Painting* displayed in the first navigation path and select the associated class *Museum* through the property *exhibited*. In this scenario, its view provides access to *Museum*(s) exhibiting *Painting*(s) that have been created by *Artist*(s), with the additional condition that these *Painting*(s) are also *Sculpture*(s). The generated RQL query by this browsing action is shown in **Table 6**.

Table 7. Navigating to specialized properties

Navigation	RQL query	Path
	`Select X1, X2 from {X1}paints{X2}.{;Painting}exhibited{X3}, Sculpture{X2}`	`Sculpture Painter→paints→Painting→ →exhibited→Museum`

As a different navigation scenario, we consider browsing through specialized properties, like *paints*, which can be reached in our example by clicking on the property *creates* already appearing in the first navigation path of **Table 6**. Compared to subclasses (see **Table 3**), browsing subproperties has as effect to alter both the domain and range classes of properties included in a navigation path. As shown in **Table 7**, this browsing action will construct a new view containing those *Painter*(s) that have *paint*ed *Painting*(s) *exhibited* in a *Museum* with the additional condition that these *Painting*(s) are also *Sculpture*(s). Since these classes are the default domain-range definitions of *paint*, the corresponding restrictions can be omitted in the from clause of the generated RQL query.

Table 8. Navigating to an associated class from multiple different paths

Navigation	RQL query	Path
	select X3 from {X1}paints{X2}. {;Painting}exhibited{X3}, Sculpture{X2}	Sculpture Painter→paints→Painting →exhibited→Museum

Finally, as a last navigation scenario, we consider browsing through different paths that converge to the same class. This class is the common range of properties (same or different) traversed by distinct branching paths. In **Table 8**, we assume that the user has navigated twice to the class *Museum* through the property *exhibited*: first from the class *Painting* and then from the class *Sculpture*. Given the semantics of the navigation to sibling classes (**Table 5**), our user is intrested in *Painting(s)*, which are also *Sculpture(s)*, and the previous browsing action results to the addition only of the subpath Painting→exhibited→Museum. Note that the addition of the subpath Sculpture→exhibited→Museum is redundant, since the generated RQL query will always return the same results as previously: the corresponding view provides access to *Painters* that have created *Painting*, which are also *Sculptures*, exhibited in a *Museum*.

It becomes clear that GRQL enables users to browse towards all directions in an RDF/S schema (i.e., subsumption and association relationships of classes and properties), including backtracking and choice of branching navigation paths. The views constructed at each step carry the information of the previous navigation and augment it with the additional resources that can be actually accessed by the current browsing action. In other words, GRQL creates dynamic views based on the incremental construction of adequate RQL queries capturing the effect of browsing actions performed by a user during its navigation session. To begin a new session, users have just to return back to the initial tree-shaped interface of class (or property) subsumption relationships and choose another entry point to the schema.

2.4 Enriching Path Expressions with Filtering Conditions

In addition to the ability to navigate through the RDF schema paths, users can formulate filtering conditions on the attributes of the class visited at each navigation step. The comparison operators which can be used in these conditions are related to the data types of the class attributes specified in XML Schema [1, 20], e.g. strings, datetime, numerical like integer and float, and enumerations. Thus, GRQL enables users to select graphically the desirable filtering condition that can be used with a particular attribute data type and fill the comparison values in a dedicated text area.

For example, if the RQL query of **Table 4** should be restricted to obtain *Artist*s, whose last name (attribute named *last_name*) contains a "P", the user has to insert the required information ("P") and select the "like" operator for filtering. The constructed conjunctive RQL query is given below:

```
select X1 from {X1}:creates{X2;Painting}, {X1}last_name{X4} where X4 like "*P*"
```

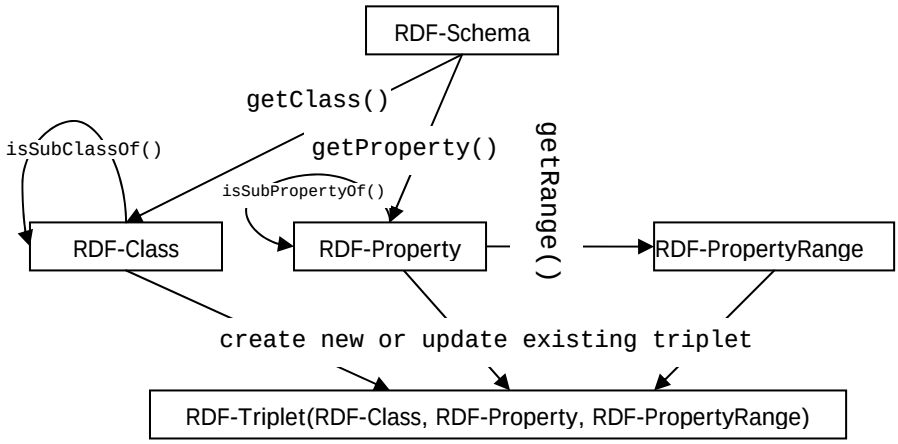


Fig. 2. The GRQL Java classes representation of RDF/S schemas

It becomes clear that filtering provides the ability to select resources that their attributes conform to user specified conditions in conjunction with their associations to other resources specified by the user schema navigation paths. As a result RQL queries can define custom views that present to users a snapshot of the underlying RDF/S description base by hiding unnecessary information.

3 Implementation of the GRQL Translation Algorithm and GUI

In this section we detail the translation algorithm of the user browsing actions on an RDF/S schema into appropriate RQL queries. To this end, we first introduce the GRQL interface internal representation of RDF/S schemas and RQL queries. Finally, we present the GRQL GUI, which allows browsing arbitrary RDF/S schemas and resource descriptions by generating RQL queries on the fly.

3.1 GRQL Internal Model of RDF/S Schemas and RQL Queries

An RDF/S schema is first loaded into the main memory during the GRQL system initialization by using appropriate Java classes. That way, schema navigation is efficiently implemented by accessing directly the required, at each step, class and property definitions instead of generating new RQL queries (e.g., to obtain subclasses or subproperties). Note that, GRQL is able to obtain during initialization the entire schema information stored in a RSSDB database [0] through a single RQL query.

As we can see in **Fig. 2**, the result of the initial schema query is processed by the java class RDF-Schema, while each RDF class is represented by the java class RDF-Class and each RDF property by the java class RDF-Property (java class RDF-PropertyRange captures either a literal type or a class serving as range of a property). Schema navigation performed by the user is represented as a list of objects of type RDF-Triplet containing references to objects of type RDF-Property, RDF-Class and RDF-PropertyRange. Then methods (like getDomain(), getRange(), isSubClassOf(), isSubPropertyOf()) are used to construct the appropriate RDF-Triplets according to the browsing actions performed by a user.

3.2 GRQL Translation Algorithm of Browsing Actions

The creation of dynamic views during the user's navigation relies on the generation of an appropriate RQL query. In order to create correctly this query we have implemented an algorithm translating the user navigation into schema triplets, which are used in the sequel to produce the corresponding RQL query, as well as visualize the traversed navigation paths. The algorithm relies on the RDF/S schema information represented by the Java classes of **Fig. 2**. A triplet consists of statements of the form <*subject, predicate, object*> where *predicate* captures the properties while *subject* and *object* capture the classes traversed during schema navigation.

Recall that GRQL interface allows users to navigate though classes and subclasses, as well as properties and subpropeties. Note also that properties are selected either explicitly by their name, or implicitly by their range given a specific domain class in the navigation path. At each browsing action performed by a user the GRQL translation algorithm will either create a new triplet or modify an existing one or leave everything unchanged in case of loops. The triplets belong either to the same path that the user is currently traversing (in which case the new triplet's domain is the range of the current triplet), thus creating a chain or to a new path (in which case the new triplet's domain is the same with the domain of the current triplet), thus creating tree-shaped paths. Thus, GRQL is able to construct and manipulate any kind of navigation path, including linear and branching paths with common origin and/or destination as depicted in **Fig. 3**. The pseudocode of the GRQL translation algorithm is given in **Fig. 4** (the class and property selected by a user are denoted as selClass and selProp).

Going back to the first navigation scenario of the previous section, one can easily correlate the triplets created by our algorithm with the navigation paths traversed by a user and consequently with the generated RQL queries. So the initial selection of the *Artist* class as an entry point produces the first triplet: *<Artist, empty, empty>* (lines 2 – 5 of **Fig. 4**). Then, when the user navigates to an associated class, i.e. selects the *Artifact*(s) *create*(d) by the *Artist*(s), the triplet is modified to *<Artist, creates, Artifact>*. It should be stressed that we could obtain the same triplet in one step, if in the initial selection the property *creates* was chosen instead of the class *Artist* (lines 34 – 37). In any case, by selecting a subclass (*Painting*) the user chooses a more specific description about the available resources and thus the triplet is once more modified to *<Artist, creates, Painting>* (lines 15 – 18).

In the backtracking scenario when the user revisits a class, the triplets produced insofar remain the same. On the other hand, when the user selects a class not specializing the domain of the current triplet (i.e., sibling), the algorithm creates a new triplet corresponding to a branching path. For example, by selecting *Sculpture*(s) *create*(d) by *Artist*(s) instead of *Painting(s)*, the triplet *<empty, empty, Sculpture>* is added to the list of existing triplets (lines 20 – 25). Once again users could alternatively have chosen to view the *Sculpture*(s) *sculpt*(ed) by the *Sculptor*(s) by navigating through the subproperty *sculpts* of *creates*, which results to the modification of the triple *<Artist, creates, Painting>* to *<Sculptor, sculpts, Sculpture>* (lines 43 - 44) and the addition of the triplet *<empty, empty, Painting>* (lines 51 – 55). In both cases the user's navigation path has two branches and the RQL query is updated accordingly.

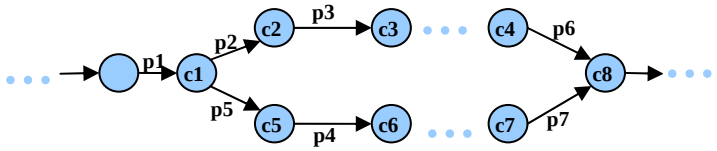


Fig. 3. Schema navigation paths constructed and manipulated by GRQL

Assuming now that the user wants to further expand his navigation along the initial path (*<Artist, creates, Artifact>*) in order to also obtain the *Museum*s that the *Artifacts* are exhibited at, the new triplet *<Artifact, exhibited, Museum>* will be added (lines 64 - 69) to the list of existing triples. If the user narrows the scope of its navigation by selecting the subclass *Painting*, this browsing action affects both triplets having *Artifact* as subject or object by modifying them to *<Artist, creates, Painting>*, *<Painting, exhibited, Museum>* (lines 15 – 18 and 10 – 13 respectively).

In the same manner, triplets are created when users add filtering conditions to their navigation paths. In this case though, triplets are constructed based on the attribute, its value and corresponding condition and the associated class. Referring to the example presented in 2.4 and for exactly the same RQL query the constructed triplets will be: *<Artist, creates, Painting>*, *<Artist, last_name, like "*P*">*.

It becomes clear that the GRQL translation algorithm captures successfully a sequence of complex browsing actions performed by a user at the schema level. Then, at each navigation step the generated triples are employed to construct dynamically the RQL query defining a user view. Last but not least, the GRQL translation algorithm ensures that the generated RQL queries are minimal, i.e., there do not exist other less complex RQL queries, which can produce the same result. As a matter of fact, our algorithm computes at each browsing step the minimal RQL query corresponding to the conjunction of the current and the previous navigation paths. For instance, assuming that a user selects initially the property *creates*, then refines its range to the subclass *Painter* and finally narrows its view to the subproperty *paints*, GRQL generates three RQL queries for which the following containment relationships are true:

Q3 select * **Q2** select * **Q1** select *
from {X1}paints{X2} ⊆ from {X1;Painter}creates{X2} ⊆ from {X1}creates{X2}

Since the result of **Q2** is contained in the result of **Q1**, the conjunction of the queries (or more precisely of their path expressions) generated at the current (2) and previous steps (1) results to a redundant RQL query, which can be simplified as follows:

Q2 select * **Q2'** select *
from{X1;Painter}creates{X2} ≡ from{X1;Painter}creates{X2},{X1}creates{X2}

In the same way, the minimal RQL query produced at the navigation step 3 is **Q3**. Readers are referred to [6] for formal foundations of RQL query containment.

```
(00) if (user navigates through subclasses)
(01) {
(02)   if (the navigation path is empty)
(03)   {//create a new triplet where its domain is the selected class
(04)     create triplet (selClass, empty, empty);
(05)   }
(06)   else
(07)     //iterate through the already created triplets
(08)     for each triplet (c1,p,c2) in list of navigation paths
(09)     {//if the selected class is a subclass of triplet's domain
(10)       if (selClass.isSubClassOf(c1) )
(11)       {//modify the triplet's domain to the selected class
(12)         modify triplet (selClass, p, c2);
(13)       }
(14)       //if the selected class is a subclass of triplet's range
(15)       else if (selClass.isSubClassOf(c2))
(16)            {//modify the triplet's range to the selected class
(17)              modify triplet (c1, p, selClass);
(18)            }
(19)            //if the selected class is a sibling of triplet's domain/range
(20)            else if (selClass.isSubClassOf(p.getDomain()) or
(21)                     selClass.isSubClassOf(p.getRange()))
(22)                 {//create a new branching triplet for path nodes
(23)                   if (candidate triplet is not in list of navigation paths
(24)                       and not exist other triplets in path specializing the
                                candidate triplet)
(25)                     create triplet (selClass, empty, empty); }
(26)     }
(27) }
(28) // user navigates through properties and subproperties
(29) else
(30) {//get the domain of the selected property
(31)   domain = selProp.getDomain();
(32)   //get the range of the selected property
(33)   range =  selProp.getRange();
(34)   if (the list of navigation paths is empty)
(35)   {//create a triplet with the selected property, its domain and range
(36)     create triplet (domain, selprop, range);
(37)   }
(38)   else
(49)     //iterate through the already created triplets
(40)     for each triplet (c1,p,c2) in list of navigation paths
(41)     { //if the selected property selProp is a subproperty of the current
(42)       //triplet's property
(43)       if (selProp.isSubPropertyOf(p))
(44)          {//modify the property of the current triplet to selProp
(45)           modify triplet (domain, selProp, range);
(46)           //if triplet's domain is sibling of the selProp's domain
(47)           if !((domain==c1) or (domain.isSubClassOf(c1))
(48)              // create a new branching triplet for path nodes
(49)              if (candidate triplet is not in list of navigation paths)
(50)                  create triplet (c1, empty, empty);
(51)           //if triplet's range is sibling of the selProp's range
(52)           if !((range==c2) or (range.isSubClassOf(c2))
(53)              // create a new branching triplet for path nodes
(54)              if (candidate triplet is not in list of navigation paths)
(55)                  create triplet (empty, empty, c2);
(56)       }
(57)       //if the selected property differs from the triplet's property
(58)       else if (selProp != p)
(59)            {//adjust the domain of the immediate following triples in path
(60)              if (range.isSubClassOf(c1))
(61)              { //modify the triplet's domain to the selected property range
(62)                modify triplet (range, p, c2);
(63)              }
(64)              //if current triplet differs from the selected property
(65)              else if ((c1 != domain) and (c2 != range))
(66)              { //create a new branching triplet for path edges
(67)                if (candidate triplet is not in list of navigation paths
(68)                    and not exist other triplets in path specializing the
                              candidate triplet)
(69)                  create triplet (domain, selProp, range);
(70)              }
(71)            }
(72)     }
(73) }
```

Fig. 4. Pseudocode of the class navigation triplet construction

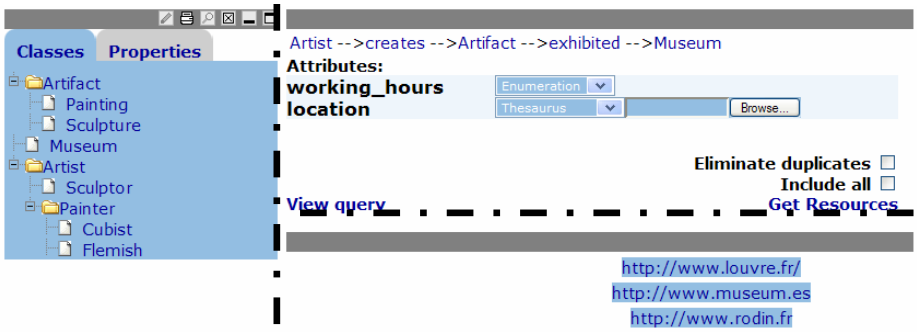


Fig. 5. A Snapshot of the GRQL GUI after navigating at the schema level

3.3 The GRQL Graphical User Interface

The GRQL GUI consists of three basic interaction areas. The left area provides a tree-shaped display of the subsumption hierarchies[3] of both the classes and properties defined in an RDF/S schema. Selecting the name of a class or property, results to a new navigation session. This session considers the selected class or property as entry point for further schema browsing. In this respect, the right upper area of the GRQL GUI allows users to explore progressively the individual RDF/S class and property definitions and generate navigational and/or filtering RQL queries. Finally, the right lower area visualizes the constructed query/view results. A snapshot of the GRQL GUI is illustrated in **Fig. 5**.

The left area of the GUI displays the predefined views over the class and property hierarchies (e.g., corresponding to a particular RDF/S schema namespace). Each tree node is labeled with a class or a property name and users can access their subclasses and subproperties by expanding the tree nodes. Each tree node is essentially a hyperlink that, after activation, fetches in the right area of the GUI the complete definition of the selected class or property. More precisely, for each selected class the following items are displayed: its subclasses, its associations with other classes and its attributes. For each selected property the domain and range classes are displayed, as well as its subproperties. As have presented in the previous section, selections on classes associations and/or subsumption result to the construction of appropriate schema navigation paths.

GRQL also supports navigation at the resource level. When the *"Get Resources"* hyperlink is activated, the lower area of the GUI will display the resources matching the generated RQL path expressions and filters, i.e., populating the constructed view. If the "*Include All*" button is also selected, the resources of all the classes taking part in the navigation path(s) will be displayed. In the opposite, (and this is the default behavior), only the resources of the last visited class will be displayed (or the resources in the domain and range of the last visited property). Additionally the user has the ability to *"Eliminate Duplicates"*, if such an action is desirable. For educational purposes, a user has the ability to inspect the generated RQL query by clicking

[3] In case of multiple subsumption, the same class or property appears as a child of all its parent nodes.

on the *"View Query"* hyperlink. After computing the resources that correspond to a current position in the navigation path, users can select a specific resource URI to obtain its full semantic description (i.e., the classes under which are classified, their attribute values and associations to other resources) or they can continue their navigation at the schema level.

Finally, an appropriate navigation bar is employed to visualize the navigation history of a user and therefore the so-constructed RQL query. Hyperlinks of all constituent path parts enable users to revisit a class or property and follow an alternative navigation path. In a nutshell, GRQL GUI supports unconstrained browsing through both RDF/S resource descriptions and schemas.

4 Summary and Future Work

In this paper we have presented GRQL, a graphical query generator assisting non-expert users in browsing RDF/S resource descriptions and schemas available on the Semantic Web. More precisely, GRQL enables users to browse towards all directions in an RDF/S schema (i.e., subsumption and association relationships of classes and properties), including backtracking and choice of branching navigation paths. The most notable GRQL feature is its ability to translate, into a unique RQL query, a sequence of browsing actions during a navigation session performed by a user.

As recognized by previous work on web navigation patterns [19], non-expert users need intuitive tools for capturing resources they have visited in previous navigation steps. We believe that GRQL addresses successfully this need, by enabling SW applications to support conceptual bookmarks and personalized knowledge maps build on demand according to users' navigation history. We are currently implementing this functionality in the context of the ICS-FORTH Semantic Web Portal Generator[4].

Finally, we are planning to extend the GRQL graphical interface to also support disjunctive queries, as well as updates of RDF/S descriptions and schemas. In this way, users will be equipped with a uniform graphical interface for navigating, filtering and updating RDF/S description bases and schemas without needing to learn dedicated declarative languages or to program in application-specific APIs.

References

0. Alexaki S., Christophides V., Karvounarakis G., Plexousakis D., Tolle K.: *The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases.* In Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb), in conjunction with the Tenth International World Wide Web Conference (WWW), Hongkong, May 1, 2001.
1. Biron P. V., and Malhotra A.: *XML SchemaPart 2: Datatypes*. W3C Recommendation 02 May 2001.
2. Braga D., Campi A., and Ceri S.: *XQBE: A Graphical Interface for XQuery Engines*. In Proceeding of the 9th International Conference on Extending Database Technology (EDBT), Heraklion, Crete, Greece, March 14-18, 2004.
3. Brickley D., Guha R.V., and McBride B.: *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Recommendation 10 February 2004.

[4] http://athena.ics.forth.gr:8999/RQLdemo/

4. Carr L., Kampa S., and Miles-Board T.: *OntoPortal: Building Ontological Hypermedia with the OntoPortal Framework.* MetaPortal Final Report 2001. The semantic portal is available at: http://www.ontoportal.org.uk
5. Chang S., Lee D and Kim H. A.: *Visual Browse/Query Tool for Navigating and Retrieving Multimedia Data on Object Databases.* In Proceedings of the First International Conference on Advanced Multimedia Content Processing Osaka, Japan, November 1998.
6. Christophides V., Karvounarakis G., Koffina I., Kokkinidis G., Magkanaraki A., Plexousakis D., Serfiotis G., and Tannen V.: *The ICS-FORTH SWIM: A Powerful Semantic Web Integration Middleware*, First International Workshop on Semantic Web and Databases (SWDB), collocated with VLDB 2003, Humboldt-Universitat, Berlin, Germany, September 7-8, 2003.
7. Corcho, O., Gómez-Pérez, A., López-Cima, A., and del Carmen Suárez-Figueroa, M.: *ODE-SeW: Automatic Generation of Knowledge Portals for Intranets and Extranets*. The semantic portal is available at: http://www.esperonto.net/semanticportal/jsp/frames.jsp
8. Carey M., Haas L., Maganty V., and Williams J.: *PESTO: An Integrated Query/Browser for Object Databases.* In Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), September 3-6, 1996, Bombay, India.
9. Gibson, D.: *The RDF Distillery*. University of Queensland, 2002. Available at: http://www.kvocentral.com/reports/diongibsonthesis.pdf
10. Goldman R., and Widom J.: DataGuides: *Enabling Query Formulation and Optimization in Semi-structured Databases*. Stanford Technical Report 1997. Available at: http://www-db.stanford.edu/lore/pubs/dataguide.pdf
11. Jarrar M., Majer B., Meersman R., Spyns P., Studer R., Sure Y., and Volz R.: *OntoWeb Portal: Complete ontology and portal*. In Proceedings of the 17th National Conference on Artificial Intelligence, Austin/TX, USA, July 30-August 3, 2000, AAAI Press/MIT Press.
12. Karvounarakis G., Alexaki S., Christophides V., Plexousakis D., and Scholl M.: *RQL: A Declarative Query Language for RDF*, The Eleventh International World Wide Web Conference (WWW), Honolulu, Hawaii, USA, May 7-11, 2002.
13. Klyne G., Carroll J.J, and McBride B.: *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. 10 February 2004.
14. Magkanaraki A., Karvounarakis G., Christophides V., Plexousakis D., and Anh T..: *Ontology Storage and Querying.* ICS-FORTH Technical Report No 308, April 2002.
15. Magkanaraki A., Tannen V., Christophides V., and Plexousakis D.: *Viewing the Semantic Web Through RVL Lenses*. In Proceedings of the Second International Semantic Web Conference (ISWC), Sanibel Island, Florida, USA, 20-23 October, 2003.
16. Microsoft Visual InterDev, Available at http://msdn.microsoft.com/vinterdev/
17. Munroe K., and Papakonstantinou Y.: *BBQ: A Visual Interface for Browsing and Querying XML*. In Proceedings of the 5th IFIP 2.6 Working Conference on Visual Database Systems Fukuoka – Japan, May 10-12, 2000.
18. Petropoulos M., Papakonstantinou Y. and Vassalos V.: *Graphical Query Interfaces for Semistructured Data: The QURSED System*. In ACM Transactions on Internet Technology (TOIT), vol 5/2, 2005.
19. Tauscher L., and Greenberg S.: *How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems*. In International Journal of Human Computer Studies, Special issue on World Wide Web Usability 1997. 47(1): p.97-137.
20. Thompson H.S., Beech D., Maloney M., and Mendelsohn N.: *XML Schema Part 1: Structures*. W3C Recommendation 2 May 2001.
21. Zloof, M.: *Query By Example*. In Proceedings of the National Compute Conference, AFIPS, Vol. 44, 1975, pp. 431-438.