RQL: A Declarative Query Language for RDF

G. Karvounarakis S. Alexaki V. Christophides
D. Plexousakis M. Scholl

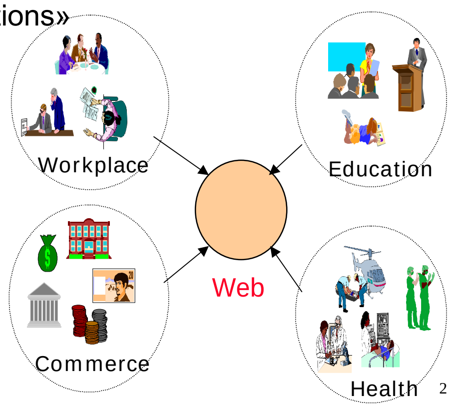
Computer Science Department, University of Crete
and Institute for Computer Science - FORTH
Heraklion, Crete

CEDRIC/CNAM and INRIA
292 Rue St Martin
75141 Paris, Cedex 03, France

Modern Web Applications …

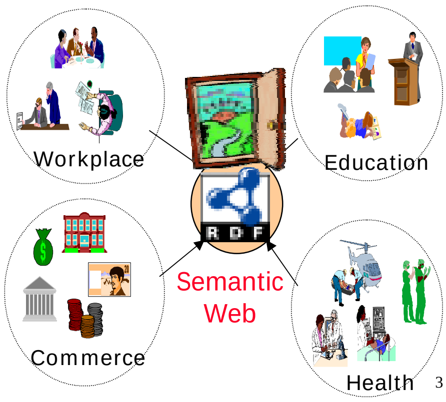
- Metadata (i.e., descriptive information) about information resources (e.g., documents, services) are crucial for:
  - Digital Museums & Libraries & Archives:
    - build on-line «Memory Organizations»
  - Corporate Knowledge Servers:
    - build «Semantic Web Portals»
  - Electronic MarketPlaces:
    - build «Virtual Enterprises»

… and many more!



… and the Semantic Web

- Modern Web applications cant benefit a lot from the Semantic Web & RDF/S
  - a standard representation language for resource descriptions with
  - a humanly readable / machine understandable syntax
  - enabling content syndication via superimposed resource descriptions
  - interpreted within or across communities using extensible descriptive schemas



What Do We Need?

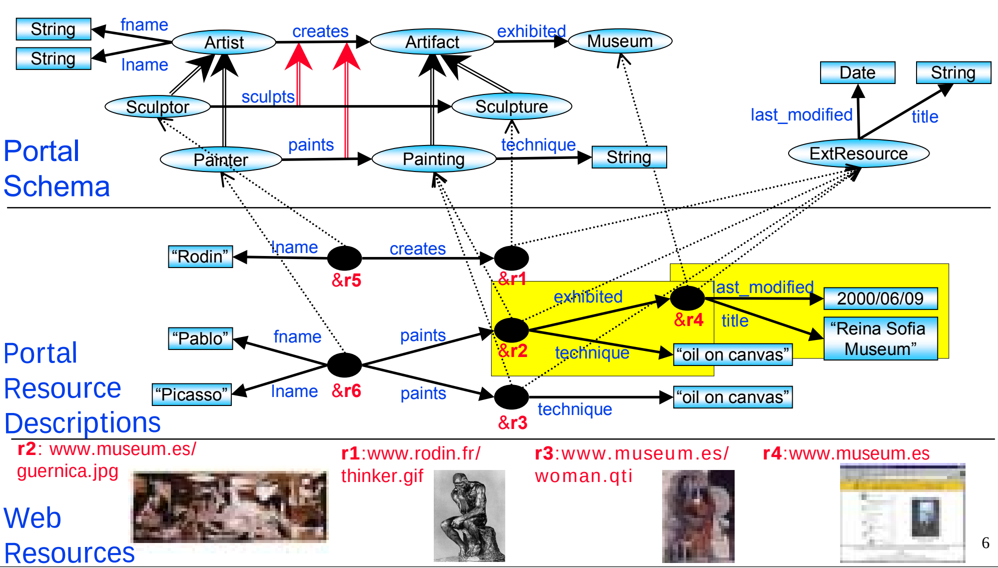
- Advanced Management of Community Metadata
  - ❶ Large Description Schemas:
    - UNSPSC:16506 classes, Getty AAT:130000 terms, ODP:385,965 topics
  - ❷ Voluminous Description Bases:
    - ODP 700M of descriptions for 3,339,355 sites
- Declarative Query Languages for Conceptual Modeling and Querying
  - ❶ Interleave schema with data querying
  - ❷ Optimize access to resource descriptions

- Our approach: take advantage of three decades of research in DB technology to support
  - declarative access and logical / physical independence for RDF description bases

Outline

- Example of a Portal Catalog for Cultural Communities
- Describing and Querying Community Resources
  - A Formal Data Model for RDF/S
  - The RDF Query Language (RQL)
- RDF Suite Architecture
- Summary

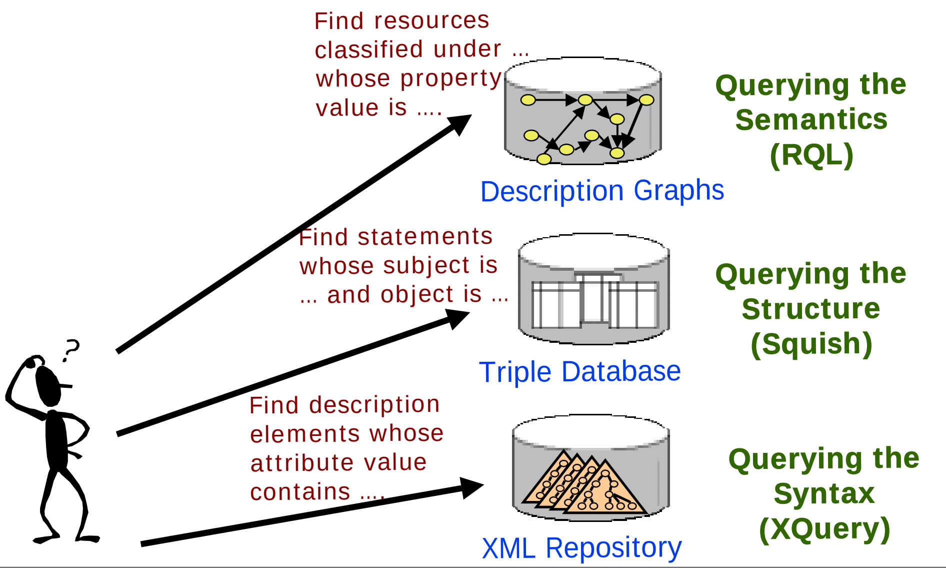
Building a Cultural Community Web Portal using RDF



Resource Description Framework (RDF/S)

- RDF: Resource Descriptions
  - Data Model: Directed Labeled Graphs
    - Nodes: Resources (URIs) or Literals
    - Edges: Properties – Attributes or Relationships
    - Labels: Nodes (Class names) and Edges (Property names)
    - Statement: assertion of the form *resource, property, value*
    - Description: collection of statements concerning a resource
  - XML syntax
- RDF Schema (RDFS): Schema Vocabularies
  - Specialization of both classes & properties (simple & multiple)
  - Multiple classification under several classes
  - Unordered, optional, and multi-valued properties
  - Domain and range polymorphism of properties

The RDF Query Language Issue



RDF/S vs. Well-Known Formalisms

- Relational or Object Database Models (ODMG, SQL)
  - Classes don't define table or object types
  - Instances may have associated quite different properties
  - Collections with heterogeneous members
- Semistructured or XML Data Models (OEM, UnQL, YAT, XML Schema)
  - Labels only on nodes or edges
  - Class and property subsumption is not captured
  - Heterogeneous structures reminiscent to SGML exceptions
- Knowledge Representation Languages (Telos, DL, F-Logic)
  - Absence of complex values (bags, sequences)

- We need a data model to define semantics of a data manipulation language
  - A query language describes in a declarative fashion, the mapping between an input instance of the data model to an output instance of the data model !

Why a Type System for RDF ?

- For error detection & safety:
  - to correctly understand statements of interest
    - e.g., don't confuse resource URIs with class/property names!
  - to enforce safety of operations
    - e.g., don't do float arithmetic on classes!
  - to check valid compositions of operations
    - e.g., don't ask the subproperties of the range of a class!

- For performance:
  - to design better storage (improving clustering, etc.)
  - to efficiently process queries (rewriting path expressions, etc.)

- We need a full-fledged Data Definition Language for RDF !
  - RDF Schema is viewed more as an ontology & modeling tool

5

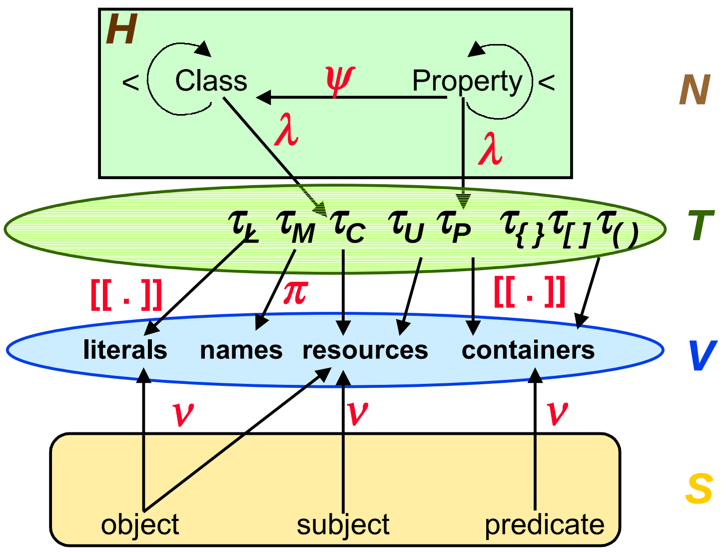
A Formal Data Model for RDF/S

- Type System:

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

- Interpretation Function:
  - Literal types: $[[\ \tau_L\ ]] = dom(\tau_L)$
  - Resource types: $[[\ \tau_u\ ]] = u \in U$
  - Class types: $[[\ \tau_C\ ]] = \{\nu \mid \nu \in \pi(c)\} \cup \{ [[\ c'\ ]]) \mid c' < c\}$
  - Property types: $[[\ \tau_p\ ]] = \{ [\nu_1, \nu_2] \mid \nu_1 \in [[\ \mathit{domain}(p)\ ]], \nu_2 \in [[\ \mathit{range}(p)\ ]]\ \} \cup \{ [[\ p'\ ]] \mid p' < p\}$
  - MetaClass types: $[[\ \tau_M\ ]] = \{\nu \mid \nu \in \pi(m)\} \cup \{ [[\ m'\ ]]) \mid m' < m\}$
  - Bag types: $[[\ \{\tau\}\ ]] = \{ \{\nu_1, \ldots, \nu_j\} \mid j > 0, \forall i \in [1..j]\ \nu_i \in [[\ \tau\ ]]\ \}$
  - Seq types: $[[\ [\tau]\ ]] = \{ [1{:}\nu_1, 2{:}\nu_2, \ldots, n{:}\nu_n] \mid n > 0, \forall i \in [1..n]\ \nu_I \in [[\ \tau_I\ ]]\ \}$
  - Alt types: $[[\ (1{:}\tau_1 + 2{:}\tau_2 + \ldots + n{:}\tau_n)\ ]] = \{i{:}\nu_I \mid, \forall i \in [1..n]\ \nu_I \in [[\ \tau_I\ ]]\ \}$

A Formal Data Model for RDF/S

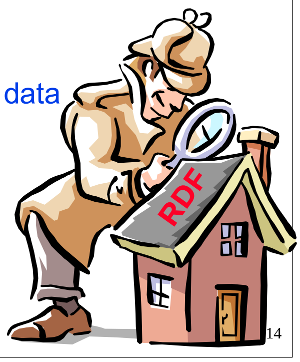


A Formal Data Model for RDF/S

- An RDF schema is a 5-tuple: RS = $(V_S, E_S, H, \psi, \lambda)$
  - V_S a set of nodes
  - E_S a set of edges
  - H = (N,<) a well-formed hierarchy of names
  - ψ an incidence function: $E_s \rightarrow V_s \times V_s$
  - λ a labeling function: $V_S \cup E_S \rightarrow \mathbf{T}$
- An RDF description base, instance of a schema RS, is a 5-tuple: RD = $(V_D, E_D, \psi, \nu, \lambda)$
  - V_D a set of nodes
  - E_D a set of edges
  - ψ an incidence function: $E_D \rightarrow V_D \times V_D$
  - ν a valuation function: $V_D \rightarrow \mathbf{V}$
  - λ a labeling function: $V_D \cup E_D \rightarrow 2^{\mathbf{N}} \cup \{Bag, Seq, Alt\}$:
    - $\forall$ u $\in V_D$, $\lambda \rightarrow$ n $\in$ **C** $\cup$ {Bag, Seq, Alt}: $\nu(u) \in$ [[n]]
    - $\forall$ e $\in E_D$ [u,u'], $\lambda \rightarrow$ p $\in$ **P** $\cup$ {1,2,3,… }: $[\nu(n),\nu(n')] \in$ [[p]]

The RDF Query Language: RQL

- Declarative query language for RDF description bases
  - relies on a typed data model (literal & container types + union types)
  - follows a functional approach (basic queries and filters)
  - adapts the functionality of semistructured or XML query languages to RDF, but also:
    - treats properties as self-existent individuals
    - exploits taxonomies of node and edge labels
    - allows querying of schemas as semistructured data



Using Names to Access RDF Schema/Data Graphs

- Querying the RDF/S (or user-defined) meta-schema names
  - **Class**
  - **Property**
  - **Literal**

Includes Painter & Sculptor

- Querying the RDF/S user-defined schema names
  - Artist
  - creates

Includes paints & sculpts

- The Namespace Clause
  - ns1:ExtResource

  using namespace ns1= &ns2:www.oclc.org/schema.rdf

Querying Large RDF Schemas with RQL

- Basic Class Queries
  - subclassof(Artist, n)
  - subclassof^(Artist)
  - superclassof(Painter , n)
  - superclassof^(Painter)
  - topclass
  - leafclass
- Basic Property Queries
  - subpropertyof(creates , n)
  - subpropertyof^(creates)
  - superpropertyof(paints , n)
  - superpropertyof^(paints)
  - topproperty
  - leafclass
- Basic Class and Property Queries
  - domain(creates)
  - range(creates)

Class & Property Querying

- Find the domain and range of the property creates

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

- Which classes can appear as domain and range of property creates

  select **$X**, **$Y** from {**$X**}creates{**$Y**} or

  select **X**, **Y** from **Class**{**X**}, **Class**{**Y**}, {;**X**}creates{;**Y**}

- Find all properties defined on class Painting and its superclasses

  select **@P**, **range**(**@P**) from {;Painting}**@P** or

  select **P**, **range**(**P**)
  from **Property**{**P**}
  where **domain**(**P**) **>=** Painting

RQL Query Result

property	class
exhibited	Museum

property	literal
technique	string


```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  - <rdf:Bag ID="bag177976867">
    - <rdf:li>
      - <rdf:Seq>
          <rdf:li rdf:type="property" rdf:resource="exhibited" />
        - <rdf:li>
          - <rdf:Alt>
              <rdf:li rdf:type="class" rdf:resource="Museum" />
            </rdf:Alt>
          </rdf:li>
        </rdf:Seq>
      </rdf:li>
    - <rdf:li>
      - <rdf:Seq>
          <rdf:li rdf:type="property" rdf:resource="technique" />
        - <rdf:li>
          - <rdf:Alt>
              <rdf:li rdf:type="literal" rdf:resource="string" />
            </rdf:Alt>
          </rdf:li>
        </rdf:Seq>
      </rdf:li>
    </rdf:Bag>
  </RDF>
```

Schema Navigation using RQL

- Iterate over the subclasses of class Artist

  select **$X** from Artist{**$X**} or

  select **X** from subclassof(Artist){**X**}

- Find the ranges of the property exhibited which can be reached from a class in the range of property creates

  select **$Y**, **$Z** from creates{**$Y**}.exhibited{**$Z**} or

  select **$Y**, **$Z** from creates{**$Y**}, exhibited{**$Z**}
  where **$Y** **<=** domain(exhibited)

- Find the properties that can be reached from a range class of property creates, as well as, their respective ranges

  select ***** from creates{**$Y**}.**@P**{**$$Z**} or

  from **Class**{**Y**}, (**Class union Literal**){**Z**}, creates{;**Y**}.**@P**{;**Z**}

Querying Complex Portal Descriptions with RQL

- Find all resources

  Resource

Multiply classified resources

- Find the resources of type ExtResource and Sculpture

  ExtResource **intersect** Sculpture

  ExtResource **minus** Sculpture

  ExtResource **union** Sculpture

- Count the total number of Painter resources

  count(Painter)

Aggregate functions

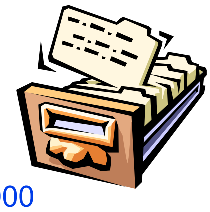
10

Filtering RDF Descriptions with RQL

- Find the file size of the resource with URI "www.artchive.com/rembrandt/abraham.jpg"

```
select X
from {X}file_size{Y}
where X = &www.artchive.com/rembrandt/abraham.jpg
```

Conditions on URIs



- Find the resources that have been modified after year 2000

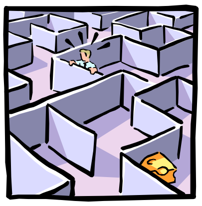
```
select X
from {X}last_modified{Y}
where Y >= 2000-01-01
```

Conditions on Dates

Navigating in Description Graphs using RQL

- Find the Museum resources that have been modified (i.e., data path with node and edge labels)

```
select X
from Museum{X}.last_modified{Y}
```



- Find the resources that have been created and their respective titles (i.e., data path using only edge labels)

```
select X, Z from creates{Y}.title{Z}
```

- Find the titles of exhibited resources that have been created by a Sculptor (i.e., multiple data paths)

```
select Z, W
from Sculptor.creates{Y}.exhibited{Z}, {Z}title{W}
```

Using Schema to Filter Resource Descriptions

- Find the properties emanating from ExtResources and their source and target values

  select **x** , **@P** , **y**
  from {**x**;ExtResource}**@P**{**y**}

Data paths foreseen in the schema

- Find the properties applied on instances of the class ExtResource and their source and target values

  select **x, @P, y**
  from ExtResource{**x**}.**@P**{**y**}

Data paths not foreseen in the schema

Notice the difference

resource http://www.museum.es/guernica.jpg	property exhibited	resource http://www.museum.es
resource http://www.museum.es/guernica.jpg	property technique	string oil on canvas
resource http://www.museum.es/woman.qti	property technique	string oil on canvas
resource http://www.museum.es	property title	string Reina Sofia Museum
resource http://www.museum.es	property last_modified	date 2000-06-09T12:30:34+00:00

Discover the Schema of RDF Descriptions

- Find the classes under which is classified the resource with URL "www.museum.es"

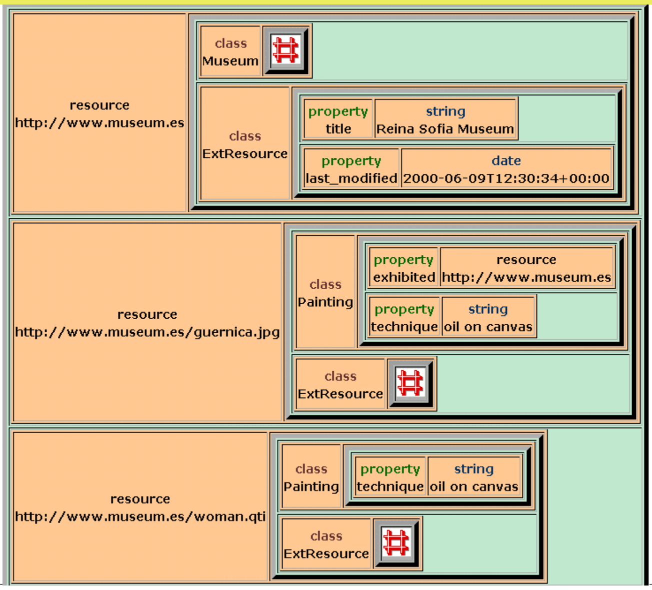
typeof (&www.museum.es)

Multiply classified resources

- Find the description of resources whose URI match "www.museum.es"

```
select $C, (select @P, Y
            from {Z ; $Z} @P {Y}
            where X = Z and $C = $Z)
from $C {X}
where X like "*http://www.museum.es*"
```


RQL Query Result

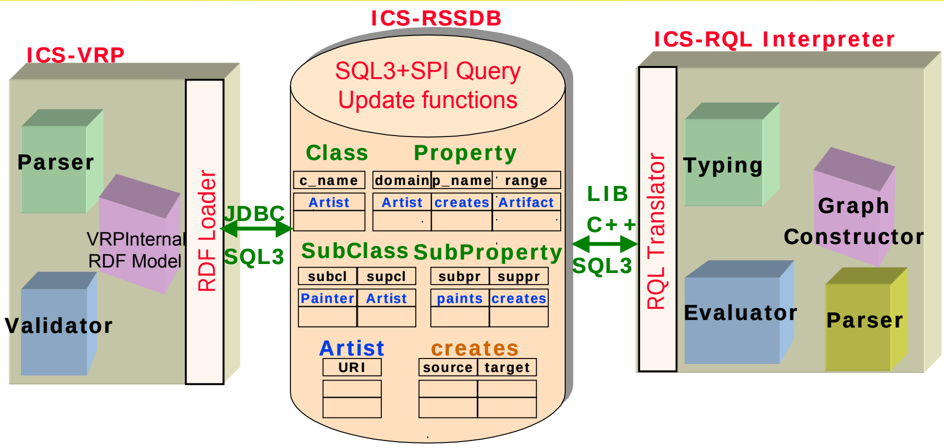


And if you still like triples …

- Find the description of resources which are not of type ExtResource

```
(
  (select X, @P, Y from {X} @P {Y})
   union
  (select X, type, $X from $X {X})
)
minus
(
  (select X, @P, Y from {X:ExtResource}@P{Y})
   union
  (select X, type, ExtResource from ExtResource {X})
)
```

The ICS-FORTH RDFSuite Architecture



- Specific Representation outperforms the Generic Representation (triple-based) by a factor up to 95,538 for queries like
  - Find the resources having a property with a specific (or range of) value(s)
  - Find the instances of a class having a given property

Summary

- RDFSuite addresses the needs of effective and efficient RDF metadata management by providing scalable tools for validation, storage, querying
  - RQL is the first declarative language for uniformly querying RDF schemas and resource descriptions
  - RSSDB is the first RDF Store using schema knowledge to automatically generate an Object-Relational (SQL3) representation of RDF metadata
- Functionality evaluation:
  - Designed in the context of the EC project C-Web (http://cweb.inria.fr)
  - Implemented in the context of the EC project MesMuses (http://cweb.inria.fr/mesmuses)
  - Accepted for use in several ongoing projects
    - Ontoknowledge (http://www.ontoknowledge.org)
    - Arion (http://dlforum.external.forth.gr:8080)
    - …

Summary

- Performance evaluation:
  - Testbed: Open Directory RDF dump
    - 505650 schema + 5331603 data triples
  - Optimization opportunities:
    - Schema and Data Indexing Techniques (transitive closure queries)
    - Heuristics for Algebraic Transformations (schema and data queries)
- Ongoing efforts:
  - RQL view, update and distribution aspects

The BNF grammar of RQL

```
• ns_query     ::= query [ "using namespace" nsdeflist ]
• query        ::= "(" query ")"
                 | "topclass" | "topproperty" | "leafclass" | "leafproperty"
                 | "subClassOf" [ "^"] "(" query [","  query ] ")"
                 | "superClassOf" [ "^"] "(" query [","  query ] ")"
                 | "subPropertyOf" ["^"] "(" query [","  query ] ")"
                 | "superPropertyOf" ["^"] "(" query [","  query ] ")"
                 | "domain" "(" query ")"
                 | "range" "(" query ")"
                 | "typeOf" "(" query ")"
                 | "namespace" "(" query ")"
                 | "count" "(" query ")"
                 | "avg" "(" query ")"
                 | "min" "(" query ")"
                 | "max" "(" query ")"
                 | "sum" "(" query ")"
                 | "bag(" query, [ query ]")“
                 | "seq(" query, [ query ]")“
                 | query "[" query "]“
                 | query "in" query
                 | query set_op query
                 | query comp_op query
                 | query bool_op query
                 | "not" query
                 | constant | ["^"] identifier
                 | ["^"] var | sfw_query
                 | "exists" var query ":" query
                 | "forall" var query ":" query
```


The BNF grammar of RQL

```
sfw_query      ::= "select" projslist "from" rangeslist [ "where" query ]
comp_op        ::= "<" | "<=" | ">" | ">=" | "=" | "!=" | "like“
set_op         ::= "union" | "intersect" | "minus“
bool_op        ::= "and" | "or“
constant       ::= integer_literal | real_literal | quoted_string_literal
                 | quoted_char_literal | date | "true" | "false" | "&" identifier
var            ::= data_var | class_var | type_var | property_var
data_var       ::= identifier
class_var      ::= "$" identifier
type_var       ::= "$" "$" identifier
property_var   ::= "@" identifier
projslist      ::= "*" | query { "," query }
rangeslist     ::= pathexpr { "," pathexpr }
pathexpr       ::= pathelem { "." pathelem }
pathelem       ::= [ "{" from_to "}" ] query [ "{" from_to "}" ]
from_to        ::= [ data_var ] [ ";" (class_var | type_var | identifier) ]
                 | class_var | type_var
nsdeflist       ::= nsdef { "," nsdef } nsdef ::= identifier "= " "&" identifier
```


16

Comparing RQL to other QLs

Query Language \ Criteria	Standard	Data Model	Language of origin	Closure of queries	Orthogonality of input / output data	Generality
RQL	RDF/S	Graph	OQL	Yes	Yes	No
SquishQL / RDQL	RDF/S	Triple	SQL	No	No	No
RDFQL	RDF/S	Triple	SQL	No	No	No
RDFPath	RDF/S	Tree	XPath	No	No	No
VERSA	RDF/S	Graph	LISP	Yes	No	No
TRIPLE	RDF/S	Triple	F-Logic	No	No	No
Description Logics QLs	DAML/OIL	Triple	DL	No	No	No
TMQL	Topic Map	Graph	SQL	No	No	Yes
Tolog	Topic Map	Triple	Datalog	No	No	No

ICS-FORTH & Univ. of Crete
May 2002

Comparing RQL to other QLs

	Criteria \ Language	RQL	SquishQL	RDFQL	RDFPath	VERSA	TRIPLE
Modeling Constructs	Namespaces / Multiple Schema	Yes	Yes	Yes	Yes	Yes	Yes
	Data Types	strings, dates, integers, reals, URI, thesauri and enumerated types	strings and integers	strings, dates, integers, reals, URI	strings	strings, URI, numbers,sets, lists, booleans	strings, integers
	Multiple Inheritance / Instantiation	Yes	Yes	Yes	Yes	Yes	Yes
	Container Values	Yes	Yes	Yes	Yes	No(?)	No
	Reification	No	No	No	No	No	Yes
Schema Querying	Ancestor / Descendant traversal of class / property hierarchies	Yes	No (only direct)	No (only direct)	No (only direct)	No (only direct)	Yes
	Filtering conditions on class / property hierarchies	(in)equality, like, subsumption, check, namespace querying	like (~), equality	Lexicographical ordering on class/property names, equality	equality	(in)equality, string containment	(in)equality subsumption test

Comparing RQL to other QLs

	Criteria \ Language	RQL	SquishQL	RDFQL	RDFPath	VERSA	TRIPLE
Data Querying	Class/ Property extent queries	Yes	Yes (only direct)	Yes (only direct)	Yes (only direct)	Yes	Yes
	Complete Boolean filters (negation, (con/dis)junction)	Yes	No (conjunction)	Yes	No (conjunction)	Yes	Yes
	Set-based operations	Yes	No	No	No	Yes	Yes
	Arithmetic operations	Yes	No	No	No	No	No
	Container values constructors	Yes	No	No	No	Yes	No
Data/ Schema Querying	Generalized path expressions	Yes	No	No	No	No	No
	Existential/ Universal quantifiers	Yes	No	No	No	No	Yes
	Nested queries	Yes	No	No	No	Yes	No

Comparing RQL to other QLs

	Criteria \ Language	RQL	SquishQL	RDFQL	RDFPath	VERSA	TRIPLE
Additional Features	Aggregate functions	Yes	No	Yes (only count)	No	Yes	No
	Grouping functions	No	No	No	No	No	No
	Sorting functions	No	No	Yes	No	Yes	No
	Built-in data functions	Yes (thesauri terms)	No	Yes (math/ string/ date)	No	Yes (conversion functions for data types supported)	No
	Arbitrary function support	No	No	Yes	No	No	No
	User-defined inference rules	No	No	Yes	No	No	Yes
	View definition primitives	No	No	Yes	No	No	No

18

Comparing RDFSuite to other Platforms

	Ref.	Doc.	Tutorial	Version	Platform	Demo	Pricing Policy
ICS-RDF Suite	Yes	Yes	Yes	1.5	Solaris/Linux	Yes	GPL compatible License
Sesame	Yes	Yes	Yes	3-Alpha	(Java)	Yes	LGPL License
Inkling	Yes	Yes	No	Alpha	(Java)	Yes	GPL/MPL License
RDFdb	Yes	Yes	No	0.46	Linux, Bsd, Solaris	NO	Mozilla License
RDFStore	No	Yes	No	0.42	(Perl)	Yes	Free Distribution
EOR	No	Yes	No	1.01	(Java)	Yes	Dublin Core Open Source License
Redland	Yes	Yes	No	0.9.10	Linux, Solaris, FreeBSD…	Yes	LGPL/Mozilla License
Jena	Yes	Yes	No	1.3.2	(Java)	No	Jena License
RDF Gateway	No	Yes	Yes	0.6	Windows NT/2000	Yes	RDF Gateway License
Triple	Yes	No	No	2002/03/14	(Java)	Yes	Semantic Web Foundation for Open Source License
KAON	Yes	Yes	No	2002/01/17	(Java)	No	KAON License
Cerebra	No	Yes	No	1.1	Windows/ Linux	No	Cerebra License
Empolis K42	No	Yes	No	1.1.1	(Java)	Yes	Empolis Ltd Licence
Ontopia KS	Yes	No	No	1.3	(Java 1.3)	Yes	Developer/Runtime License

Comparing RDFSuite to other Platforms

	Query Language	Implem. Language	Storage DB	Updates (schema+data)
ICS-RDF Suite	RQL	Java /C++	**ORDBMS** (SQL3 compliant, e.g PostgreSQL)	Yes
Sesame	RQL*	Java	**ORDBMS** (PostgreSQL)	Yes
Inkling	SquishQL	Java	**In-memory/ Persistence** (supporting JDBC, e.g PostgreSQL)	No
RDFdb	SquishQL*	C	**Persistence** (SleepyCat)	Yes
RDFStore	SquishQL	C, Perl	**In-memory/ Persistence** (e.g files, BerkeleyDB, SDBM)	No
EOR	Triple matching	Java	**Persistence** (SQL databases, e.g MySQL)	Yes
Redland	Triple matching	C	**In-memory/ Persistence** (SleepyCat/ BerkeleyDB)	Yes
Jena	RDQL	Java	**In-memory/ Persistence** (e.g BerkeleyDB, Interbase, PostgreSQL)	Yes
RDF Gateway	RDFQL	?	**RDBMS**	Yes
Triple	Triple	Java	**In-memory**	?
KAON	F-Logic	Java, Python	**In-memory/ Persistence** (e.g files, KAON server, RDBMS)	Yes
Cerebra	DL-based	Java	**Distributed data** (CORBA)	-
Empolis K42	TMQL	Java	**Persistence storage** (K42 Generic Store, other DBMS)	Yes
Ontopia KS	Tolog	Java	**In-memory/RDBMS/ OODB**	Yes

19

Comparing RDFSuite to other Platforms

	Inference Support	API Support	Scalability/ Performance	Export Data Format
ICS-RDF Suite	Yes	C++/ Java/ SQL functions	DBMS scales linearly with the number of triples/ 505650 schema + 5331603 data triples tested	RDF
Sesame	Yes	HTTP/SOAP	?	RDF
Inkling	No	Java	?	Triples in ASCII
RDFdb	Yes	C, Perl	~ 20 million triples tested	Triples in ASCII
RDFStore	Yes	Perl	1470000 triples stored in a ~98 MB database/ ~183 read operations/sec	N-Triples, RDF
EOR	No	HTTP, Java, SQL/JDBC	?	Triples rendered with XSL
Redland	No	Java, C, Perl, Python, Tcl	tested with 1.5M stored statements/ query speed is 6.200 statements/sec	Triples
Jena	No	Java	In-memory storage has been used with 600K statements/ for the SQL store is around 10ms/statement load, 1-7 ms/returned-statement search	Triples in ASCII
RDF Gateway	Yes	ADO, JDBC	?	Triples in ASCII
Triple	Yes	Java	?	Lisp, XML, DOT, DAML, ASCII
KAON	Yes	Java	?	?
Cerebra	Yes	Java	?	?
Empolis K42	Yes	Java/RMI	~ 500MB tested/ 0.08 sec for look-up of an object by name for first access	Topic Maps (XTM)
Ontopia KS	Yes	Java/J2EE	?	XTM, XML version of ISO 13250