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!
Modern Web applications can 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
  - Large Description Schemas:
    - UNSPSC:16506 classes, Getty AAT:130000 terms, ODP:385,965 topics
  - Voluminous Description Bases:
    - ODP 700M of descriptions for 3,339,355 sites
- Declarative Query Languages for Conceptual Modeling and Querying
  - Interleave schema with data querying
  - 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

Portal Schema

Portal Resource Descriptions

Web Resources
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

- **Querying the Structure (Squish)**
  - Find statements whose subject is ... and object is ...
- **Querying the Syntax (XQuery)**
  - Find description elements whose attribute value contains ....
- **Querying the Semantics (RQL)**
  - Find resources classified under ... whose property value is ....

XML Repository

Description Graphs

Triple Database
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
A Formal Data Model for RDF/S

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

Interpretation Function:
- **Literal** types: \[[\tau_L]\] = \text{dom}(\tau_L)
- **Resource** types: \[[\tau_U]\] = \{u \in U \}
- **Class** types: \[[\tau_C]\] = \{v \mid v \in \pi(c)\} \cup \{[[c']] \mid c' < c\}
- **Property** types: \[[\tau_P]\] = \{([v_1, v_2] \mid v_1 \in [[\text{domain}(p)]] \land v_2 \in [[\text{range}(p)]]\}
  \cup \{[[p']] \mid p' < p\}
- **MetaClass** types: \[[\tau_M]\] = \{v \mid v \in \pi(m)\} \cup \{[[m']] \mid m' < m\}
- **Bag** types: \[[\{\tau\}\] = \{\{v_1, v_2, \ldots, v_j\} \mid j > 0, \forall i \in [1..j] v_i \in [[\tau]]\}
- **Seq** types: \[[\{\tau\}\] = \{(1:v_1, 2:v_2, \ldots, n:v_n) \mid n > 0, \forall i \in [1..n] v_i \in [[\tau]]\}
- **Alt** types: \[[\{1: \tau_1 + 2: \tau_2 + \ldots + n: \tau\}\] = \{i:v_i, \forall i \in [1..n] v_i \in [[\tau_i]]\}
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
  - \( \psi \) an incidence function: \( E_S \rightarrow V_S \times V_S \)
  - \( \lambda \) a labeling function: \( V_S \cup E_S \rightarrow T \)

- 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
  - \( \psi \) an incidence function: \( E_D \rightarrow V_D \times V_D \)
  - \( \nu \) a valuation function: \( V_D \rightarrow V \)
  - \( \lambda \) a labeling function: \( V_D \cup E_D \rightarrow 2^N \cup \{ \text{Bag, Seq, Alt} \} \):
    - \( \forall u \in V_D, \lambda \rightarrow n \in C \cup \{ \text{Bag, Seq, Alt} \}: \nu(u) \in \llbracket n \rrbracket \)
    - \( \forall e \in E_D[u,u'], \lambda \rightarrow p \in P \cup \{1,2,3,\ldots\}: [\nu(n),\nu(n')] \in \llbracket p \rrbracket \)

---

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**

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

- 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)`
  - `superpropertyof(creates)`
  - `superpropertyof^^(creates)`
  - `topproperty`
  - `leafclass`

- **Basic Class and Property Queries**
  - `domain(creates)`
  - `range(creates)`
Class & Property Querying

- Find the domain and range of the property creates
  \[
  \text{seq} ( \text{domain}(\text{creates}), \text{range}(\text{creates}))
  \]

- Which classes can appear as domain and range of property creates
  \[
  \text{select} \ X, \ Y \ \text{from} \ \{X\}\text{creates}\{Y\} \ \text{or}
  \text{select} \ X, \ Y \ \text{from} \ \text{Class}\{X\}, \ \text{Class}\{Y\}, \ \{;\text{creates}\{;Y\}
  \]

- Find all properties defined on class Painting and its superclasses
  \[
  \text{select} \ \@P, \ \text{range}(\@P) \ \text{from} \ \{;\text{Painting}\}\@P \ \text{or}
  \text{select} \ P, \ \text{range}(P) \ \text{from} \ \text{Property}\{P\} \ \text{where} \ \text{domain}(P) >= \ \text{Painting}
  \]

RQL Query Result

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description id="bg0177976867">
    <rdf:Seq>
      <rdf:li rdf:resource="exhibited"/>
      <rdf:li rdf:resource="Museum"/>
    </rdf:Seq>
  </rdf:Description>
  <rdf:Description id="bg0177976867">
    <rdf:Seq>
      <rdf:li rdf:resource="technique"/>
      <rdf:li rdf:resource="string"/>
    </rdf:Seq>
  </rdf:Description>
</rdf: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 $Y \leq \text{domain(exhibited)}$

- 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

- 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)
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
  ```

- Find the resources that have been modified after year 2000
  
  ```
  select X
  from {X}last_modified{Y}
  where Y >= 2000-01-01
  ```

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
  
  \[
  \text{select } x, \; @P, \; y \quad \text{from} \quad \{x;\text{ExtResource}\}_@P\{y\}
  \]

- Find the properties applied on instances of the class ExtResource and their source and target values
  
  \[
  \text{select } x, \; @P, \; y \quad \text{from} \quad \text{ExtResource}\(x\)\_@P\{y\}
  \]

Notice the difference

<table>
<thead>
<tr>
<th>resource</th>
<th>property</th>
<th>resource</th>
<th>property</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.museum.es/guernica.jpg">http://www.museum.es/guernica.jpg</a></td>
<td>exhibited</td>
<td><a href="http://www.museum.es">http://www.museum.es</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://www.museum.es/guernica.jpg">http://www.museum.es/guernica.jpg</a></td>
<td>technique</td>
<td>oil on canvas</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://www.museum.es/woman.qti">http://www.museum.es/woman.qti</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>title</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reina Sofia Museum</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://www.museum.es">http://www.museum.es</a></td>
<td>last_modified</td>
<td>2000-06-09T12:30:34+00:00</td>
</tr>
</tbody>
</table>
Discover the Schema of RDF Descriptions

- Find the classes under which is classified the resource with URL “www.museum.es”
  \[\text{typeof} \ (\&\text{www.museum.es})\]

- Find the description of resources whose URI match “www.museum.es”
  \[
  \text{select} \ \$C, \ \{\text{select} \ \@P, \ Y \\
  \text{from} \ \{Z; \$Z\} \ @P \ \{Y\} \\
  \text{where} \ X = Z \ \text{and} \ \$C = \$Z\}
  \]
  \[
  \text{from} \ \$C \ \{X\} \\
  \text{where} \ X \ \text{like "*http://www.museum.es*"}
  \]

RQL Query Result
And if you still like triples …

- Find the description of resources which are not of type ExtResource
  
  \[
  \text{(select } X, @P, Y \text{ from } \{X\} @P \{Y\}\text{ union (select } X, \text{ type, } $X \text{ from } $X \{X\}\text{)) minus (select } X, @P, Y \text{ from } \{X:ExtResource\}@P\{Y\}\text{ union (select } X, \text{ type, ExtResource from ExtResource } \{X\}\text{))}
  \]

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
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)
  - ...

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 ]
"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

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 = "\t" &" identifier
### Comparing RQL to other QLs

<table>
<thead>
<tr>
<th>Query Language</th>
<th>Standard</th>
<th>Data Model</th>
<th>Language of origin</th>
<th>Closure of queries</th>
<th>Orthogonality of input/output data</th>
<th>Generality</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQL</td>
<td>RDF/S</td>
<td>Graph</td>
<td>OQL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SquishSQL/RDQL</td>
<td>RDF/S</td>
<td>Triple</td>
<td>SQL</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RDFQL</td>
<td>RDF/S</td>
<td>Triple</td>
<td>SQL</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RDFPath</td>
<td>RDF/S</td>
<td>Tree</td>
<td>XPath</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>VERSA</td>
<td>RDF/S</td>
<td>Graph</td>
<td>LISP</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TRIPLE</td>
<td>RDF/S</td>
<td>Triple</td>
<td>F-Logic</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Description Logics QLs</td>
<td>DAML/OIL</td>
<td>Triple</td>
<td>DL</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Description Logics QLs</td>
<td>Topic Map</td>
<td>Graph</td>
<td>SQL</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Description Logics QLs</td>
<td>Topic Map</td>
<td>Triple</td>
<td>Datalog</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### Comparing RQL to other QLs

<table>
<thead>
<tr>
<th>Criteria</th>
<th>RQL</th>
<th>SquishSQL</th>
<th>RDFQL</th>
<th>RDFPath</th>
<th>VERSA</th>
<th>TRIPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespaces/ Multiple Schema</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data Types</td>
<td>Strings, dates, integers, reals, URI, thesauri and enumerated types</td>
<td>Strings and integers</td>
<td>Strings, dates, integers, reals, URI</td>
<td>Strings</td>
<td>Strings, URI, numbers, sets, lists, booleans</td>
<td>Strings, integers</td>
</tr>
<tr>
<td>Multiple Inheritance/ Instantiation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Container Values</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Reification</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ancestor / Descendant traversal of class/property Hierarchies</td>
<td>Yes</td>
<td>No (only direct)</td>
<td>No (only direct)</td>
<td>No (only direct)</td>
<td>No (only direct)</td>
<td>Yes</td>
</tr>
<tr>
<td>Filtering conditions on class/property Hierarchies</td>
<td>(in)equality, subsumption, check, namespace querying</td>
<td>like (~), equality</td>
<td>Lexicographical ordering on class/property names, equality</td>
<td>equality</td>
<td>(in)equality, string containment</td>
<td>(in)equality subsumption test</td>
</tr>
</tbody>
</table>

### Generality

- **RDF/S**: Graph
- **Triples**: Tree
- **LISP**: Subject, predicate, object
- **DAML/OIL**: SNOMED CT
- **VERSA**: LISP
- **TMQL**: Topic Map
- **Tolog**: F-Logic
- **Description Logics QLs**: DAML/OIL
- **RDFQL**: RDF/S
- **RDFPath**: RDF/S
- **SquishSQL/RDQL**: RDF/S
- **VERSQ**: RDF/S
- **TRIPLE**: RDF/S
- **Description Logics QLs**: DAML/OIL
- **TMQL**: Topic Map
- **Tolog**: Topic Map

### Languages

- **RQL**: RDF/S
- **SquishSQL/RDQL**: RDF/S
- **RDFQL**: RDF/S
- **RDFPath**: RDF/S
- **VERSQ**: RDF/S
- **TRIPLE**: RDF/S
- **Description Logics QLs**: DAML/OIL
- **TMQL**: Topic Map
- **Tolog**: Topic Map

### Data Models

- **Graph**: RDF/S
- **Triples**: RDF/S
- **LISP**: RDF/S
- **DAML/OIL**: RDF/S
- **Topic Map**: RDF/S
- **F-Logic**: RDF/S

### Languages of Origin

- **OQL**: RDF/S
- **SQL**: RDF/S
- **XPath**: RDF/S
- **Datalog**: RDF/S
- **LISP**: RDF/S
- **F-Logic**: RDF/S
- **DAML/OIL**: RDF/S
- **Topic Map**: RDF/S
- **Tolog**: RDF/S

### Closure of Queries

- **Yes**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **Yes**: RDF/S
- **No**: RDF/S

### Orthogonality of Input/Output Data

- **Yes**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **Yes**: RDF/S
- **No**: RDF/S

### Generality

- **Yes**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **No**: RDF/S
- **Yes**: RDF/S
- **No**: RDF/S
### Comparing RQL to other QLs

<table>
<thead>
<tr>
<th>Criteria</th>
<th>RQL</th>
<th>SquishQL</th>
<th>RDFQL</th>
<th>RDFPath</th>
<th>VERSA</th>
<th>TRIPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Querying</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class/ Property extent queries</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complete Boolean filters (negation, (con/ dis)junction)</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Set-based operations</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Arithmetic operations</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Container values constructors</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Generalized path expressions</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Existential/ Universal quantifiers</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Nested queries</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Data Schema Querying</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregate functions</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Grouping functions</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sorting functions</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Built-in data functions</td>
<td>Yes (thesauri terms)</td>
<td>No</td>
<td>Yes (math/ string/ date)</td>
<td>No</td>
<td>(conversion functions for data types supported)</td>
<td>No</td>
</tr>
<tr>
<td>Arbitrary function support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>User-defined inference rules</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>View definition primitives</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
## Comparing RDFSuite to other Platforms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ICS-RDF Suite</td>
<td></td>
<td></td>
<td>1.5</td>
<td>Solaris/Linux</td>
<td>Yes</td>
<td>Yes</td>
<td>Java /C++</td>
<td>RQL</td>
<td>ORDBMS (SOL3 compliant, e.g PostgreSQL)</td>
<td>Yes</td>
</tr>
<tr>
<td>Sesame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Java</td>
<td>RQL*</td>
<td>ORDBMS (PostgreSQL)</td>
<td>Yes</td>
</tr>
<tr>
<td>Inking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Java</td>
<td>SqushQL</td>
<td>In-memory/ Persistence (Supporting JDBC, e.g PostgreSQL)</td>
<td>No</td>
</tr>
<tr>
<td>RDFdb</td>
<td></td>
<td></td>
<td>0.46</td>
<td>Linux, Solaris,</td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>RDFdb</td>
<td>Persistence (SleepyCat)</td>
<td>Yes</td>
</tr>
<tr>
<td>RDF Store</td>
<td></td>
<td></td>
<td>0.42</td>
<td></td>
<td>No</td>
<td>No</td>
<td>C</td>
<td>RDFStore</td>
<td>In-memory/ Persistence (e.g BerkeleyDB, Interbase)</td>
<td>No</td>
</tr>
<tr>
<td>EOR</td>
<td></td>
<td></td>
<td>1.01</td>
<td></td>
<td>No</td>
<td>No</td>
<td>C, Perl</td>
<td>EOR</td>
<td>Persistence (mailboxes, e.g MySQL)</td>
<td>Yes</td>
</tr>
<tr>
<td>Redland</td>
<td></td>
<td></td>
<td>0.9.10</td>
<td>Linux, Solaris,</td>
<td>Yes</td>
<td>No</td>
<td>Java</td>
<td>Redland</td>
<td>Persistence (SleepyCat, BerkeleyDB)</td>
<td>No</td>
</tr>
<tr>
<td>Jena</td>
<td></td>
<td></td>
<td>1.3.2</td>
<td></td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>Jena</td>
<td>In-memory/ Persistence (e.g BerkeleyDB, Interbase, PostgreSQL)</td>
<td>Yes</td>
</tr>
<tr>
<td>RDF Gateway</td>
<td></td>
<td></td>
<td>0.6</td>
<td>Windows NT/2000</td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>RDF Gateway</td>
<td>In-memory/ Persistence (CORBA)</td>
<td>No</td>
</tr>
<tr>
<td>Triple</td>
<td></td>
<td></td>
<td>2002/03/14</td>
<td>Java</td>
<td>Yes</td>
<td>Yes</td>
<td>Java</td>
<td>Triple</td>
<td>In-memory/ Persistence (e.g files, KAON server, RDBMS)</td>
<td>No</td>
</tr>
<tr>
<td>KAON</td>
<td></td>
<td></td>
<td>2002/01/17</td>
<td>(Java)</td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>KAON</td>
<td>In-memory/ Persistence (e.g files, KAON server, RDBMS)</td>
<td>No</td>
</tr>
<tr>
<td>Cerebra</td>
<td></td>
<td></td>
<td>1.1</td>
<td>Windows/Linux</td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>Cerebra</td>
<td>Persistence (JDBC, Interbase, PostgreSQL)</td>
<td>Yes</td>
</tr>
<tr>
<td>Empolis K42</td>
<td></td>
<td></td>
<td>1.1.1</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Java</td>
<td>Empolis K42</td>
<td>Persistence (JDBC, Interbase, PostgreSQL)</td>
<td>No</td>
</tr>
<tr>
<td>Ontopia KS</td>
<td></td>
<td></td>
<td>1.3</td>
<td></td>
<td>No</td>
<td>No</td>
<td>Java</td>
<td>Ontopia KS</td>
<td>In-memory/ Persistence (e.g MySQL)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
# Comparing RDFSuite to other Platforms

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Inference Support</th>
<th>API Support</th>
<th>Scalability/ Performance</th>
<th>Export Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICS-RDF Suite</td>
<td>Yes</td>
<td>C++, Java, SQL functions</td>
<td>9MBF scales linearly with the number of triples 506530 schema + 537603 data triples tested</td>
<td>RDF</td>
</tr>
<tr>
<td>Sesame</td>
<td>Yes</td>
<td>RMI, SOAP*</td>
<td>?</td>
<td>RDF</td>
</tr>
<tr>
<td>Inkling</td>
<td>No</td>
<td>Java</td>
<td>?</td>
<td>Triples in ASCII</td>
</tr>
<tr>
<td>RDFlib</td>
<td>Yes</td>
<td>C, Perl</td>
<td>~ 30 million triples tested</td>
<td>Triples in ASCII</td>
</tr>
<tr>
<td>RDFStore</td>
<td>Yes</td>
<td>Perl</td>
<td>1470000 triples stored in a ~8.8 MB database, 183 read operations/sec</td>
<td>N-Triples, RDF</td>
</tr>
<tr>
<td>EOR</td>
<td>No</td>
<td>HTTP, Java, SQL/JDBC</td>
<td>?</td>
<td>Triples rendered with XSL</td>
</tr>
<tr>
<td>Redland</td>
<td>No</td>
<td>Java, C, Perl, Python, Tcl</td>
<td>In-memory storage has been used with 600K statements for the SQL store is around 10ms/statement load, 1-7 ms/returned-statement search</td>
<td>Triples in ASCII</td>
</tr>
<tr>
<td>RDF Gateway</td>
<td>Yes</td>
<td>ADO, JDBC</td>
<td>?</td>
<td>Triples in ASCII</td>
</tr>
<tr>
<td>Triple</td>
<td>Yes</td>
<td>Java</td>
<td>?</td>
<td>Lisp, XML, DOT, DAML, ASCII</td>
</tr>
<tr>
<td>KAON</td>
<td>Yes</td>
<td>Java</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Cerebra</td>
<td>Yes</td>
<td>Java</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Empolis K42</td>
<td>Yes</td>
<td>Java/RMI</td>
<td>~ 80MB tested 0.05 sec for lookup of an object by name for first access</td>
<td>Topic Maps (XTM)</td>
</tr>
<tr>
<td>Ontopia KS</td>
<td>Yes</td>
<td>Java/J2EE</td>
<td>?</td>
<td>XTM, XML version of ISO 13250</td>
</tr>
</tbody>
</table>

*Note: RMI, SOAP are not supported by RDFSuite.*

---

May 2002

ICS-FORTH & Univ. of Crete

In-memory storage has been used with 600K statements for the SQL store is around 10ms/statement load, 1-7 ms/returned-statement search.

Redland uses Java, C, Perl, Python, Tcl.

RDF Gateway supports ADO, JDBC.

Triple supports Java.

Jena uses Java.

Cerebra uses Java.

Empolis K42 uses Java/RMI.

Ontopia KS uses Java/J2EE.