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 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
  1. Large Description Schemas:
     - UNSPSC:16506 classes, Getty AAT:130000 terms, ODP:385,965 topics
  2. Voluminous Description Bases:
     - ODP 700M of descriptions for 3,339,355 sites

- Declarative Query Languages for Conceptual Modeling and Querying
  1. Interleave schema with data querying
  2. 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
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 Semantics (RQL)**: Find resources classified under ... whose property value is ...
- **Querying the Syntax (XQuery)**: Find description elements whose attribute value contains ...
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:
  \[ \tau = \tau_1 | \tau_U | \tau_C / \tau_P / \tau_M / \{ \tau \} | (1:1+2:1+...+n:1\} | (1:1+2:1+...+n:\tau) \]

- Interpretation Function:
  - Literal types: \[ [[ \tau_L ]] = \text{dom}(\tau_L) \]
  - Resource types: \[ [[ \tau_U ]] = u \in U \]
  - Class types: \[ [[ \tau_C ]] = \{ v | v \in \pi(c) \} \cup \{ [[ c' ]] | c' < c \} \]
  - Property types: \[ [[ \tau_P ]] = \{ [v_1, v_2] | v_1 \in [[ \text{domain}(p) ]], v_2 \in [[ \text{range}(p) ]] \} \cup \{ [[ p' ]] | p' < p \} \]
  - MetaClass types: \[ [[ \tau_M ]] = \{ v | v \in \pi(m) \} \cup \{ [[ m' ]] | m' < m \} \]
  - Bag types: \[ [[ \{ \tau \} ]] = \{ \{ v_1, ..., v_j \} | j > 0, \forall i \in [1..j] v_i \in [[ \tau ]] \} \]
  - Seq types: \[ [[ [ \tau ] ]] = \{ [1:v_1, 2:v_2, ..., n:v_n] | n > 0, \forall i \in [1..n] v_i \in [[ \tau ]] \} \]
  - Alt types: \[ [[ (1:1+2:1+...+n:1) \] = \{ i:v_i | i \in [1..n] v_i \in [[ \tau ]] \} \]
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: \( V_D \cup E_D \rightarrow 2^N \cup \{\text{Bag, Seq, Alt}\} \):
    - \( \forall u \in V_D, \lambda \rightarrow n \in C \cup \{\text{Bag, Seq, Alt}\}: \nu(u) \in [[n]] \)
    - \( \forall e \in E_D[u,u'], \lambda \rightarrow p \in P \cup \{1,2,3,...\}: [\nu(n),\nu(n')] \in [[p]] \)

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

- 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)
  - 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
  \[
  \text{seq ( domain(creates), range(creates) )}
  \]

- 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

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Seq>
    <rdf:Alt>
      <rdf:Seq rdf:resource="exhibited" />
      <rdf:Alt rdf:resource="Museum" />
      <rdf:Alt rdf:resource="technique" />
      <rdf:Alt rdf:resource="string" />
    </rdf:Alt>
  </rdf:Seq>
</rdf:Description>
```
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**

- 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"
  ```rql
  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
  ```rql
  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)
  ```rql
  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)
  ```rql
  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)
  ```rql
  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
  
  \[
  \text{select } x, @P, y \\
  \text{from } \{x;\text{ExtResource}\}@P\{y\}
  \]

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

Notice the difference

<table>
<thead>
<tr>
<th>resource</th>
<th>property</th>
<th>resource</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>
</tr>
<tr>
<td>resource</td>
<td>property</td>
<td>string</td>
</tr>
<tr>
<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>resource</td>
<td>property</td>
<td>string</td>
</tr>
<tr>
<td><a href="http://www.museum.es/woman.qti">http://www.museum.es/woman.qti</a></td>
<td>technique</td>
<td>oil on canvas</td>
</tr>
<tr>
<td>resource</td>
<td>property</td>
<td>string</td>
</tr>
<tr>
<td><a href="http://www.museum.es">http://www.museum.es</a></td>
<td>title</td>
<td>Reina Sofia Museum</td>
</tr>
<tr>
<td>resource</td>
<td>property</td>
<td>date</td>
</tr>
<tr>
<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”

\[
\text{select } @C, (\text{select } @P, @Y \\
\text{from } \{ @Z ; @Z \} \text{ at } @P \{ @Y \} \\
\text{where } X = @Z \text{ and } @C = @Z) \\
\text{from } @C \{ X\} \\
\text{where } X \text{ like } ^*\text{http://www.museum.es}* ^*\]

RQL Query Result
And if you still like triples …

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

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)
    • …

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
ns_query ::= ( query )
"topclass" | "topproperty" | "leafclass" | "leafproperty"
"subClassOf" | "subPropertyOf" | "superClassOf" | "superPropertyOf"
"domain" | "range" | "typeOf"
"count" | "avg" | "max" | "min" | "sum"
"bag" | "seq" | "query">

query ::= ( query )
"topclass" | "topproperty" | "leafclass" | "leafproperty"
"subClassOf" | "subPropertyOf" | "superClassOf" | "superPropertyOf"
"domain" | "range" | "typeOf"
"count" | "avg" | "max" | "min" | "sum"
"bag" | "seq" | "query">

nsdeflist ::= nsdef { nsdef }
nsdef ::= identifier = "&" identifier

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 ::= "$" identifier
type_var ::= "$" type_var
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 ]
data_var | class_var	nsdeflist ::= nsdef { nsdef } nsdef ::= identifier = "&" identifier
### Comparing RQL to other QLs

<table>
<thead>
<tr>
<th>Query Language</th>
<th>Criteria</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></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></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></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></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></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></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></td>
<td>DAML/OIL</td>
<td>Triple</td>
<td>DL</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TMQL</td>
<td></td>
<td>Topic Map</td>
<td>Graph</td>
<td>SQL</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Tolog</td>
<td></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>Language</th>
<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>
<td></td>
</tr>
<tr>
<td>Data Types</td>
<td>strings, dates, integers, reals, LRI, thesauri and enumerated types</td>
<td>strings, dates, integers, reals, LRI</td>
<td>strings, dates, integers, reals, LRI</td>
<td>strings</td>
<td>strings, LRI, numbers, sets, lists, booleans</td>
<td>strings, integers</td>
<td></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>
<td></td>
</tr>
<tr>
<td>Container Values</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No(?)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Reification</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></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>
<td></td>
</tr>
<tr>
<td>Filtering conditions on class/property hierarchies</td>
<td>(inequality, like, subsumption, check, namespace querying)</td>
<td>like (&quot;&quot;), equality</td>
<td>lexical ordering on class/property names, equality</td>
<td>equality</td>
<td>(inequality, string containment, (inequality subsumption test)</td>
<td>(inequality subsumption test)</td>
<td></td>
</tr>
</tbody>
</table>
### Comparing RQL to other QLs

#### Data Querying

<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>Class/ Property extent queries</td>
<td>Yes</td>
<td>Yes (only direct)</td>
<td>Yes (only direct)</td>
<td>Yes (only direct)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complete Boolean filters (negation, (con/ dis)junction)</td>
<td>Yes</td>
<td>No (conjunction)</td>
<td>Yes</td>
<td>No (conjunction)</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>No</td>
<td>Yes</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>
</tbody>
</table>

#### Data/Schema Querying

<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>Aggregate functions</td>
<td>Yes</td>
<td>No</td>
<td>Yes (only count)</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 (multi-string data)</td>
<td>No</td>
<td>Yes (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>No</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>
</tr>
</thead>
<tbody>
<tr>
<td>ICS-RDF Suite</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>1.5</td>
<td>Solaris/Linux</td>
<td>Yes</td>
</tr>
<tr>
<td>Sesame</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>3.0</td>
<td>Alpha (Java)</td>
<td>Yes</td>
</tr>
<tr>
<td>Inkling</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>0.46</td>
<td>Linux, BSD, Solaris</td>
<td>No</td>
</tr>
<tr>
<td>RDFdb</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>0.42</td>
<td>Perl</td>
<td>Yes</td>
</tr>
<tr>
<td>RDF Store</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>1.01</td>
<td>(Java)</td>
<td>Yes</td>
</tr>
<tr>
<td>Redland</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>0.9.10</td>
<td>Linux, Solaris, FreeBSD</td>
<td>Yes</td>
</tr>
<tr>
<td>Jena</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>1.3.2</td>
<td>(Java)</td>
<td>No</td>
</tr>
<tr>
<td>RDF Gateway</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>0.6</td>
<td>Windows NT/2000</td>
<td>Yes</td>
</tr>
<tr>
<td>Triple</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>2002/03/14</td>
<td>(Java)</td>
<td>Yes</td>
</tr>
<tr>
<td>KAON</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>2002/01/17</td>
<td>(Java)</td>
<td>No</td>
</tr>
<tr>
<td>Cerebra</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>1.1</td>
<td>Windows/Linux</td>
<td>No</td>
</tr>
<tr>
<td>Empolis K42</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>1.1.2</td>
<td>(Java)</td>
<td>Yes</td>
</tr>
<tr>
<td>Ontopia KS</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>1.3</td>
<td>(Java 1.3)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Comparing RDFSuite to other Platforms

<table>
<thead>
<tr>
<th>Query Language</th>
<th>Implem. Language</th>
<th>Storage DB</th>
<th>Updates (schema+data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICS-RDF Suite</td>
<td>ROL</td>
<td>Java /C++</td>
<td>ORDBMS (SOL3 compliant, e.g PostgreSQL)</td>
</tr>
<tr>
<td>Sesame</td>
<td>ROL *</td>
<td>Java</td>
<td>ORDBMS (PostgreSQL)</td>
</tr>
<tr>
<td>Inkling</td>
<td>SquashQL</td>
<td>Java</td>
<td>In-memory/ Persistence (Supporting JDBC, e.g PostgreSQL)</td>
</tr>
<tr>
<td>RDFdb</td>
<td>SquashQL *</td>
<td>C</td>
<td>Persistence (Sleepycat)</td>
</tr>
<tr>
<td>RDF Store</td>
<td>SquashQL</td>
<td>C, Perl</td>
<td>In-memory/ Persistence</td>
</tr>
<tr>
<td>EOR</td>
<td>Triple matching</td>
<td>Java</td>
<td>Persistence (SOL databases, e.g MySQL)</td>
</tr>
<tr>
<td>Redland</td>
<td>Triple matching</td>
<td>C</td>
<td>In-memory/ Persistence</td>
</tr>
<tr>
<td>Jena</td>
<td>RDFQL</td>
<td>Java</td>
<td>In-memory/ Persistence</td>
</tr>
<tr>
<td>RDF Gateway</td>
<td>RDFQL</td>
<td>?</td>
<td>RDBMS</td>
</tr>
<tr>
<td>Triple</td>
<td>Triple</td>
<td>Java</td>
<td>In-memory</td>
</tr>
<tr>
<td>KAON</td>
<td>F-Logic</td>
<td>Java, Python</td>
<td>In-memory/ Persistence</td>
</tr>
<tr>
<td>Cerebra</td>
<td>DL-based</td>
<td>Java</td>
<td>Distributed data</td>
</tr>
<tr>
<td>Empolis K42</td>
<td>TMQSL</td>
<td>Java</td>
<td>Persistence storage</td>
</tr>
<tr>
<td>Ontopia KS</td>
<td>Tolog</td>
<td>Java</td>
<td>In-memory/RDBMS/ ODBMS</td>
</tr>
</tbody>
</table>
# Comparing RDFSuite to other Platforms

<table>
<thead>
<tr>
<th></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</td>
<td>5MMx scales linearly with the number of triples 550650 schema + 533763 data triples tested</td>
<td>RDF</td>
</tr>
<tr>
<td>Sesame</td>
<td>Yes</td>
<td>HTTP/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>RDFdb</td>
<td>Yes</td>
<td>C, Perl</td>
<td>~ 50 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 ~98 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; query speed is 6.200 statements/sec</td>
<td>Triples in ASCII</td>
</tr>
<tr>
<td>Jena</td>
<td>No</td>
<td>Java</td>
<td>?</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>~ 500MB tested 3.08 sec for look-up of an object by name for final 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 13050</td>
</tr>
</tbody>
</table>

---

Comparing RDFSuite to other Platforms