Understanding data by their context using RDF

Karsten TOLLE, *Database and Informationssystems,* Johann *Wolfgang Goethe-Universität Frankfurt am Main (Germany)*

***Abstract*--The Resource Description Framework (RDF) was designed to enable machines to understand the semantic of encoded data. There exist different ways how to understand, use and maintain context information with it. In this paper we will look at the benefits this could bring for intelligent systems and will present the applications *RDF-Source related Storage System (RDF-S3)* and *easy RQL (eRQL)* that are build to support its users with different context information for a better understanding of the data.**

***Index Terms*—knowledge representation, semantic networks, truth maintenance, artificial intelligence**

I. Introduction

One mandatory objective for intelligent systems is that they interact with their environment. This means they receive inputs and react the way they are programmed and/or trained. Depending on the system the range for input is either very limited, or the context of the data needs to exist or be included to the input too. This can be compared with human interaction, e.g., one friend says to another: "Yesterday our team won 2:0!" The two friends probably know which sport and team they are talking about, since they know each other. An intelligent system (IS) would need to know these context information too, to be able to process such an input the right way. There are different ways to enable an IS to understand the context and the given input. There are artificial intelligence approaches trying to extract the semantic from the human language. Another approach is to already encode the input and its context information in a way that the semantic behind it can be retrieved by the IS. This is the one that will be concentrated on in this paper. The World Wide Web Consortium (W3C) worked out the Resource Description Framework (RDF) and the Web Ontology Language (OWL) for exactly this reason, to enable machines to understand the semantic behind the data. The emergence of RDF is expected to enable metadata interoperability across different communities and applications by supporting common conventions on metadata syntax, structure, and semantics. RDF data can be regarded as a set of little sentences, each having a subject, a predicate and an object. These sentences are also called (RDF-) statements or triples. A set of RDF triples can also be represented as a graph, whereby the subjects and objects are the nodes and the predicates are directed edges pointing form the subject to the object.

One central term of this paper is *context.* There is no clear and universally accepted definition for context within knowledge bases and therefore not within RDF. An overview of existing interpretations can be found in [1]. For RDF I see three different situations where the term context is used. I therefore split context into subparts. First, the context given by the surrounding graph, I call this *internal context*. The second situation I call the *external context*, like source information, time of accessing the source, etc. Including this information into the RDF model can result in confusion and will result in untrustworthy data. Third, the context used to identify triples for a clear and easier handling of sets of triples, e.g., to merge/unmerge graphs or to set access privileges.

I worked out a storage system for RDF data called *RDF-S3 (RDF-Source related Storage System)* and a query language on top called *eRQL (easy RDF Query Language)*. One of the peculiarities of it is the ability to store and provide context information. RDF-S3 stores external context information, as a minimum the URI of the source used to load the RDF data. This enables the system to allow deletions and updates of single documents that had been entered before. This external context information can be retrieved and used by eRQL. E.g., queries can be restricted to run only on defined sources and it is possible to exclude some which are be not relevant or untrustworthy sources before the execution. In addition eRQL provides the ability to view results inside their internal context. More precisely the surrounding graph up to a range defined by the query will be included to each triple fitting the query. This provides a better understanding of the returned results.

An IS working on RDF data could benefit from adopting RDF-S3 and eRQL. This can be demonstrated by the previous example of the two friends talking about their team. With RDF-S3 all input data will be stored together with the information where the data came from. This means if the context information which team and sports a person is talking about is not included to the input, it can be tried to infer this information from previous inputs of that person. In case it can not be inferred this way, so, there is no other chance than to ask the person as we would do in a human to human conversation.

You will find a brief introduction to RDF capturing the needed parts for this paper in section II. In section III the definition for context in the range of RDF is further discussed

Manuscript received September 7, 2004.

Karsten Tolle is with the Databases and Information Systems (DBIS), Johann Wolfgang Goethe-University of Frankfurt, Robert-Mayer-Strasse 11-15, D-60325 Frankfurt, Germany (e-mail: tolle@dbis.informatik.uni-frankfurt.de).

and existing approaches how to handle context in RDF are analyzed. This is followed by some ideas how an IS could benefit from these approaches and the use of RDF in section IV. The section V then describes RDF-S3 and eRQL in more detail, that can be used as a basis for an IS knowledge base. Finally you can find some conclusions and future work in section VI.

II. Brief introduction to RDF

In this section I will describe those parts of RDF you will need to understand the ideas of the paper. Of course I will stay on a very abstract level. To get a more detailed introduction please have a look at the RDF Primer [4].

RDF data can be regarded as a set of little sentences, each having a subject, a predicate and an object (SPO). These sentences are also called (RDF-) statements or triples. A set of RDF triples can be represented as a graph, whereby the subjects and objects are the nodes and the predicates (also called *properties*) are directed edges pointing form the subject to the object. In RDF everything is a *resource*, meaning it can be identified by a Uniform Resource Identifier (URI). The only exception are native values, called *literals*, like integers or strings. The literals can only appear in the object part of a triple.

RDF comes with a set of predefined predicates and some classes that can be used to categorize *resources*. To extend this vocabulary RDF Schema (RDF/S) can be used to declare and define further properties and classes. RDF/S is using RDF triples to build up these new vocabularies. In that sense RDF and RDF/S are self-contained. Both are so strong related which means when talking about RDF we normally mean RDF together with RDF/S.

An RDF graph can be transformed into an RDF/XML representation for exchange purposes. This is the normal encoding you will encounter RDF data in the Web. Those RDF/XML files can then point to external vocabularies, also called *namespaces*, to reuse them. When building the graph of an RDF/XML file, you will need to include the graph of the namespaces into it.

For this paper also the RDF construct of an *reified statement* is important. A reified statement identifies one triple, so that assertion can be made about it. It can for example be used for indirect speech. Suppose we have the sentence: "Julia said she likes ballet. ", you can build: "Astrid said that Julia said she likes ballet. " Note: To build a reified statement, it is worth to point to the subject, the predicate and object of the triple. This means you need three additional triples in the graph (in reality you will need four, since the node identifying the triple must be instantiated to the class rdf:Statement).

When you have an RDF graph given, you can try to infer from it additional information or test if given facts are valid within the graph. A more precise description how this can be done can be found in [2].

III. Context within the Semantic Web Community

We can find a common definition for context in dictionaries, e.g., in [11] we can read as a definition for context: *"The part of a text or statement that surrounds a particular word or passage and determines its meaning. The circumstances in which an event occurs; a setting."* However, there is no clear and universally accepted definition for context in the area of knowledge base systems and therefore also none in the Semantic Web community. An overview of existing interpretations of the term *context* in the area of knowledge base systems can be found at [1]. This section first gives an overview of existing approaches and will then give my conclusion which approach should be used.

A. Existing approaches

For RDF and therefore for the Semantic Web, we can find a list of approaches how to deal with context information: reified statements, Quadruples (Quads), Named Graphs and Triples Plus Context Node. These approaches are introduced below.

1. Reified statements – using reified statements, you can identify single statements and assert additional information to it. Graham Klyne describes in [3] how reified statements combined with containers can be used for more complex context handling. This approach stays completely in the existing RDF model. All information can be handled therefore by existing RDF tools and there is no semantic extension needed. However, converting a triple into an reified statement needs four additional triples. This results in a blow-up of triples. This can be tried to be compensated, but the resulting graph will get rather complicated and therefore also its handling. Already Graham Klyne used in [3] an abbreviated form to write reified statements using quadruples.

2. Quadruples (Quads) – this approach is discussed in [8]. It extends the RDF triples by a forth component. A Quad is therefore a four-tuple <C, S, P, O> (the order here is not important) where C is a context and S, P and O are the triple components of RDF. The C is used to group triples and can be used as a normal resource to make statements about these groups. This means it can be used also as a subject (S) or an object (O). In some cases the C can have a *null value*. Here an example:

```
[_:cxt _:Julia ex:likes ex:Ballet]
[_:cxt _:Julia ex:daughterOf _:Astrid]
[null _:cxt ex: loadedBy "Karsten"]
[null _:cxt ex: loadingDate "20.10.2004"]
```

These statements assert that, the facts "Julia likes Ballet" and "Julia is the daughter of Astrid" are both loaded by "Karsten Tolle" on date "October 20 2004".

The drawbacks here is that by extending the RDF model, existing RDF tools can not handle quads. Additionally the semantic of the context part is not defined, it is just introduced as context. This is very flexible but might cause confusion too.

3. Named Graphs – are a reformulation of quads. A more detailed description can be found in [9] or [10]. With Named Graphs the RDF triples are grouped to sets (called graphs) and named by a global name (URI) or an anonymous node. Again

the graph names can be used as a subject or object part of the RDF triples to make assertions to the whole graph. Using the TriG [6] syntax the upper example can be translated into the following Named Graph:

```
_:G1 (_:Julia ex:likes ex:Ballet.
      _:Julia ex:daughterOf _:Astrid.)
_:G2 (_:G1 ex:loadedBy "Karsten".
      _:G1 ex:loadingDate "20.10.2004".)
```

As a difference to the quads, Named Graphs are coming with a clearer interpretation. In [10] we can read:

We do not directly semantically interpret Named Graphs; however an RDF(S) interpretation I (as in [2]) conforms with a set of Named Graphs N when:

For every Named Graph ng ∈ N, we have I(name(ng)) = ng

Additional they state in [10] that Named Graphs are downward compatible with RDF.

4. Triple Plus Context Node – as in Quads a Triple Plus Context Node is a four-tuple <C, S, P, O>. The difference to the quads is that the context node C is semantically separated from the RDF part, meaning you can not make an assertion to all statements having the same C part *inside* the SPO part. By leaving the RDF part and the C part separate it is easy to fade out the C to return to the pure RDF model. The semantic definition of the context node here also depends on the application. It can be used for grouping or identifying statements for merging, unmerging, update reasons or to determine the source URL of the triples. By the separation of the C part from the SPO part, the examples given for the Quads and Named Graphs, can not be represented using only Triples Plus Context Nodes. However, by identifying and grouping the triples by the context node, the assertions can be made outside in a separate model.

As a bottom-line we see that all four approaches are actually very close to each other, Named Graphs are a reformulation of Quads, reified statements can be written by quadruples and Triples Plus Context Nodes can be viewed as a subset of Quads having higher restrictions on using the context part.

B. Which approach to choose?

To find a solution which approach to choose, we need to clarify the term context. My proposal for the dilemma of having no common accepted definition for it, is to break it down into subparts. In [5] I differentiated context into a) *internal context*, defined as the surrounding RDF graph for one or a set of triples, b) *external context*, like source information, time of loading, etc. and c) *further context usages*, e.g. for access privileges or more easy handling of sets of triples like merging or unmerging them.

Depending on the context type you want to express, also the needed features are different. The external and further context is context on another semantic level than the data themselves. It is therefore useful to be able to differentiate between the data and the context information. This is in particular important when trust or confidence of the data is needed. In the first three approaches this differentiation is not possible, the main goal for these approaches is a better handling for the internal context. The Triple Plus Context Node approach is the only one that allows this differentiation, whereby it has the drawback of the low expressiveness for the internal context.

This means for dealing with internal context one should choose one of the first three, whereby the Named Graphs are coming with an easy handling and a clearer interpretation. For external context one should use Triples Plus Context Node. The question remains, what to do if both, internal and external context is important. In such cases, since the first approach using reified statements stays inside the RDF model, it can be combined with the Triple Plus Context Node approach. Anyway, the mentioned drawbacks of reified statements remain. A combination of Quads or Named Graphs with the Triples Plus Context Nodes could result in a quintuple <EC, IC, S, P, O> having an external context node EC and an internal context node IC. Note that the additional information that will be made about the external context (like when the information was stored and by whom) is not included to the quintuple. As a conclusion I would say that there currently is no approach that captures all needed features. Since the underlying model also should not get to complicated – otherwise it will not be used – I would vote for an coexistence of different approaches. So, for each application the more fitting one can be chosen, combined or extended.

C. Some additional note on internal and external context

It is worth to mention that the internal context may differ from situation to situation. Suppose a knowledge base containing hundreds of RDF files. This results in a big RDF graph, where the single files might affect each other. This means loading or deleting a file might change the internal context of others. The same is true about namespaces. It is stated that namespaces should not be changed and that a new URL should be used to update a namespace, but reality has shown that namespace changes are done frequently. In addition it is possible to extend schemas and continue describing their elements elsewhere. In [7] we requested to avoid this and in the meantime the RDF and RDF Schema namespaces have been changed accordingly, but it is still possible.

For the external context we have a different situation. The external context will not differ over time once a file is loaded into a repository. Of course it might be that the original file will be updated or its URL is changed, but this does not affect the repository. The external context can be viewed as metadata for the given RDF information, which means it is located on a separate level.

IV. Bringing together the Semantic Web and IS

Coming from the Semantic Web area, I first want to clarify my understanding of an Intelligent System (IS): *An IS is a system that interacts with its environment and learns during its existence, so that it reaches certain objectives more often.* However, this might not be a complete definition for all areas related to IS, but to follow this paper it should be reasonable.

Let us think of an IS containing a knowledge base using RDF, to keep the information it collects from outside and the external context information (where did the information came from, when, under which circumstances, etc.). I assume here that the knowledge base is just used to store the data and not to manipulate them. Coming back to the introducing example of two friends talking to each other and let one of them be our IS. In case we take the complete knowledge base of the IS to infer information from it, it can result in an overkill of information. The same is true for human interaction, you know more than one sport and your friend probably too, anyway, you both know by context what was mend. The question is now, which kind of context is important to solve our problem, internal or external context? I believe both, whereby the external context is the important in first place to find the relevant information. The internal context is then important to find the correct results within this relevant information and to be able to interpret them.

For example, the external context can be used to focus on the information that has been entered by our friend into the IS. Additionally, just the information entered during the last time could be taken into account. An aspect we so fare did not looked at, could be the information about the surrounding environment. Sitting in a baseball stadium of course can have an impact and would belong to the external context too. One could then only take those information into account that have been entered by the friend during a similar situation.

When we want to build an RDF knowledge base that can be used for an IS, it is therefore important to handle the external context information. To provide confidential information and an easy handling, it is also needed to separate these information form the data themselves. The only approach supporting this differentiation is the *Triple Plus Context Node*. Note that additional information as we can find in the Quads or Named Graph example in the section above, like who and on which date loaded something into the repository, needs to be stored separately. With RDF-S3, that will be described in the next section, we build a basis that can be viewed as a starting point to realize such an RDF knowledge base.

The internal context is essential for the interpretation of given data and therefore also important for an IS. Suppose the query: *What is the width of this bridge?* The IS would first of all needs to find out which bridge we are talking about. In case it figure it out, it might return the result "*42*". Now having this result in return, we are forced to interpret it. 42 meter, 42 feet, 42 something and how did the system figured it out? A more precise response could be: *There is a Web page at xyz written by Michael Paul Thoma, stating that the width of the Golden Gate Bridge is 90 and its height is 746 and all units are in feet. There is additional another page at opq stating the width of it is 42.* As you can see by this response, we might get useful extra information, in this case the units. We also can get information that might enable us to choose between different contra dictionary answers 90 or 42. In those cases the combination of internal and external context is important so solve the contra diction. Of course there might be information returned that is not relevant at all, we did not asked for the height of the bridge. Anyway, also this information might be useful, since it is related to the subject, but there is the danger of having so many additional information that the real answer is overseen. This means we need a flexible way how to increase or decrease the size for the internal context. eRQL for this reason supports this by a flexible way to include surrounding information depending on its distance to the result. Of course there is some potential for more sophisticated approaches depending on the application and the underlying RDF vocabulary.

V. Details on RDF-S3 and eRQL

In this section the two applications RDF-S3, eRQL and their fundamental ideas are described. Both, RDF-S3 and eRQL are implemented as 100% pure Java™ open source applications[1]. They where developed and tested using *IBM DB2 Universal Database v8.1 (ESE)* underneath but should run also with other relational databases that support foreign keys and nested queries (just pointing this out since MySQL 4.0 does not support nested queries).

A. RDF-Source related Storage System (RDF-S3)

For each RDF statement stored via RDF-S3, the system also stores the URL of the file used to enter it. This means it generates Triples Plus Context Nodes that will be stored, whereby the context node is semantically fixed to the source information (URL) the triple came from. Additionally, the time of loading the file is kept too. With these two information the deletion or update of single already entered sources can be performed. RDF-S3 has an extra GUI to perform updates and deletions easily. The system can be extended to store additional external context information too, but since it depends on the application which external context information are needed, those two are the only one supported out of the box. RDF-S3 comes with an easy to handle graphical user interface as shown in Fig. 1. It allows the settings for the database connection, the definition of the input file or files, output options, validation options, etc. The system is based on the ICS-FORTH Validating RDF Parser (VRP). This allows additional semantic validation that can be done to ensure a higher data quality.

The storage information about how many different kinds of RDF constructs have been stored and the needed time for it can be saved in an output file. The information can either be encoded as a plain file using delimiters to separate the entries or as an RDF/XML file. The idea of this file is to serve as basis to test the storage performance of the system for the different RDF constructs.

The internal storage structure of RDF-S3 combines the two current storage approaches *Generic Representation* and the *Schema specific Representation* by redundant storage[2].

[1] *RDF-S3* (including *eRQL*) can be downloaded at: www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/RDFS3/.

[2] In contrast to the SpecRepr, RDF-S3 so far does not differ between schema and meta-schema level.

1) Generic Representation

The *Generic Representation* (*GenRepr*) can be simplified as storing all triples in one table having three rows for the three parts of a triple. This approach is very easy to understand. When the queries stay on the statement level, e.g., *give me all statements containing a specific resource*, this approach works perfect. But the drawbacks of it is getting obvious when the queries getting more complex and include schema information, e.g., *give me all statements of a specific class*. For such queries the system needs to create a self join, which results in a performance overkill [12].

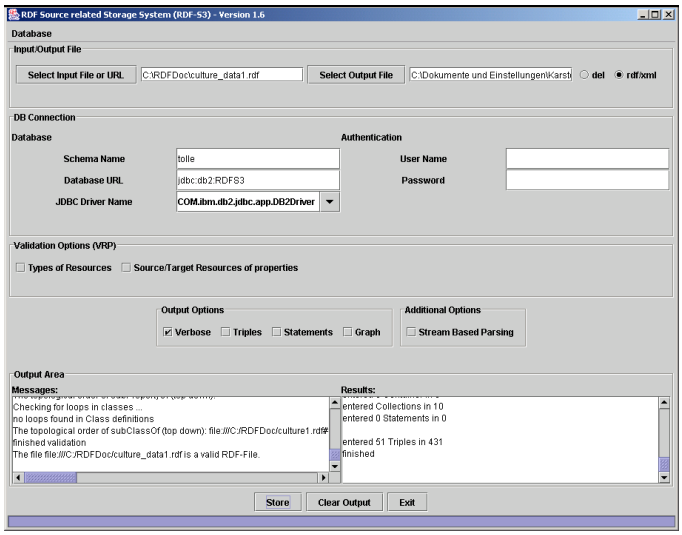

Fig. 1. Screenshot of the RDF-S3 GUI version 1.6.

2) Schema Specific Representation

The *Schema specific Representation* (*SpecRepr*) results in a complex set of tables. This approach also includes a layer structure to distinguish between meta information, schema information and the data themselves. By entering new schema information you also need to enter new tables. This results in an increasing number of database objects (tables) the database management system needs to handle[3]. In the end we can think of this representation as a presorting of the triples. Therefore, the query performance for queries over schema information is better for the SpecRepr compared to the *GenRepr*. See [12] for more details and examples. But for queries on the statement level you might need to scan nearly all existing tables.

By the combination of these two representations a query system on top can choose which representation fits best the needs of the single queries. *Statement level queries* can be evaluated on the table for the GenRepr and *schema queries* could be made on the tables used for the SpecRepr.

B. easy RDF Query Language

The easy RDF Query Language (*eRQL*) was historically constructed as a wrapper for RQL [13] that uses a SQL like syntax and can therefore not serve as an end-user query language for information portals etc. This changed with the evolution of RDF-S3 and eRQL. eRQL now is working directly on top of RDF-S3.

The main goal of eRQL is to be simple enough to be used without any knowledge of the underlying ontology used to describe the data. Also the query syntax itself should be intuitive. This goal is reached by being close to the syntax of *Google*. By simply entering keywords that can be combined by *AND* or *OR*, queries can be performed. As a result for these queries you would get those triples in return, that contain the given word as either subject, predicate or object. By putting a keyword into quotation marks, the request is restricted to literals. The matching triples are called *direct hits*. One peculiarity of eRQL is to return not only the triples fitting the request, but also those surrounding them (called *PointOfInterest Mode – POI Mode*). This way eRQL includes internal context information to the result. Therefore, a better understanding of the result can be reached as described in section IV. The distance how much of the surrounding graph should be returned can be defined by the number of leading ~ signs. The default is already one, meaning if you enter the query "bridges", the system will search for all direct hits. For each of them, the system will then will include those triples that are connected to them to the result. For a query with a distance of zero, the query needs to be enclosed by brackets "[...]". The result then only contains the single direct hits (called *Statement Mode*).

Since RDF-S3 stores also the source information of the triples, this knowledge can be used too. With the so called *Document Mode* in eRQL single or a group of sources can be either left out or a query can be restricted to them. The syntax for the document mode is: "*<query; source_list; restrict>*". The *query* can be any valid eRQL query, the *source_list* is a list of source URLs separated by comma and *restrict* is either *0* to leave out the defined sources or *1* to restrict the query to them. As an abbreviation the sources in the *source_list* can also be identified by internal IDs, that must be retrieved from the database before. Therefore, a valid query could look like: "*<bridges; 5,6; 0>*". This would execute the query "bridges" on all stored information except those that comes from the sources internally identified by *5* and *6*.

In addition to this functionality eRQL in the meantime also supports the most RQL schema functions to get a quick overview of the underlying ontology. A complete list of functions and further possibilities supported by eRQL can be found in the appendix. These functions are splitted into *general schema functions* that do not need an input and *schema functions on resources* that will need an eRQL query as input parameter, like *domain(query)*, that will return the domain definitions of the properties fitting the given query.

As RDF-S3 also the implementation of eRQL comes with an graphical user interface shown in Fig. 2. It includes the settings for the database connection and also the mapping from source URLs to the internal used IDs to abbreviate them in the document mode.

[3] Depending on the DBMS the existence of to many tables reduces the performance.

The result that will be returned depends on the query. General schema functions will return lists of found resources. Results for schema functions on resources will list the resources together with the resource they belong to, e.g., the query *domain(father)* will list all properties fitting the query "*father*" together with their defined rdfs:domain classes.

In case of triples being returned, they are grouped by the POIs they belong to. Triples always returned including their source information. For schema functions the source information is only included when the document mode was activated.

To work with the returned eRQL results there is either the possibility to reuse the internal Java classes or to store the results in an RDF format using a vocabulary given by the application. As in RDF-S3 the result also includes the processing time for the query.

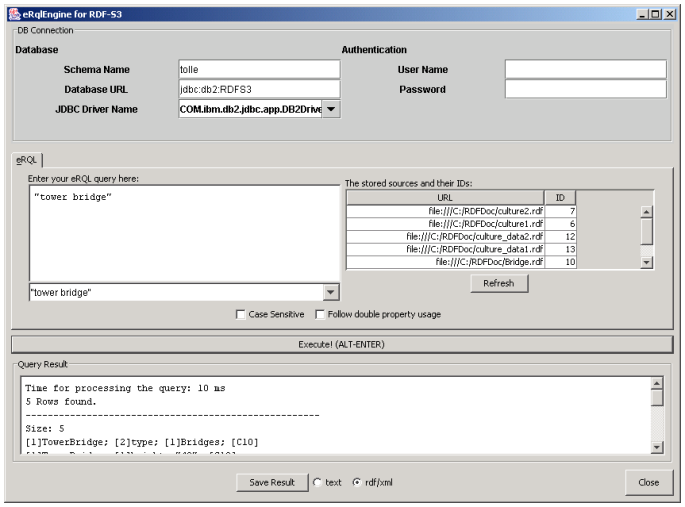

Fig. 2. Screenshot of the eRQL implementation GUI version 1.6.

VI. Conclusion

The external context can be used for an IS to reduce the knowledge base that is needed to answer a given question. This is not only important to minimize the response time, but also to blend out possible wrong answers. Depending on the person you talk to "our team" can have a different semantic meaning. For a confidential handling it is important to be able to differentiate between the information and their external context information coming with it. RDF-S3 supports this differentiation by storing the data as Triples Plus Context Nodes. With the introduced eRQL the external context information can be accessed and used inside the queries. It therefore allows the needed reduction of the knowledge base. In addition eRQL provides the ability to include internal context information as defined in the query to the result. The results are therefore more expressive and can be understood without generating additional queries.

What is currently missing for RDF-S3 and the implementation of eRQL is a full performance test and evaluation, to compare it with other existing tools and to find the bottlenecks in the implementation. A first step in this direction is done by the systems to provide the storage and retrieval time as a feedback.

Appendix

A. Short description of the current eRQL syntax

POI Query or POI Mode: *one-word-query* – will return all stored triples, where the given query word is contained by either part of the triple. The surrounding graph with a distance of 1 is included to the result. A cumulative usage of '~' can be used to increase the distance. Wildcards *"*"* and *"?"* can be used, whereby *"*"* stands for any string and *"?"* for exact one character.

Resource Query : *res(one-word-query)* – as the POI Query but will ignore literals as object part.

Literal Query : *"query_string"* – as the POI Query but concentrates on literals in the object part only. The query_string can contain spaces.

Statement Mode : *[query]* – concentrates on the *direct hits*, there is no surrounding graph included.

Document Mode: *<query; source_list; restrict>* – *restrict* can be either *0* or *1*, in case of *1* the query is only executed on the given sources, in case of *0* these sources are left out.

TABLE I
Overview of *schema functions on resources* supported by eRQL.

Syntax	Description
directInstancesOf(q), di(q), dI(q)	Will return all direct instances for the classes fitting the query q.
domain(q), d(q), D(q)	Will return the defined domain classes for the properties fitting the query q.
instancesOf(q), i(q), I(q)	Will return all instances (including the instances of subclasses) for the classes fitting the query q.
range(q), r(q), R(q)	Will return the defined range classes for the properties fitting the query q.
subClassOf(q), subc(q), subC(q)	Will return all sub classes of the classes fitting the query q.
subPropertyOf(q), subp(q), subP(q)	Will return all sub properties of the properties fitting the query q.
superClassOf(q), superc(q), superC(q)	Will return all super classes of the classes fitting the query q.
superPropertyOf(q), subp(q), subP(q)	Will return all super properties of the properties fitting the query q.

With the version 1.6 of RDF-S3 and eRQL also *schema functions* are supported. The schema functions can be separated into *general function* that do not need an input and *functions on resources* that will need a query as input to find fitting resources. Whereas the POI Mode does not work for schema functions, meaning there will be no surrounding graph be returned, the Document Mode is compatible with schema functions. A list and short description of all currently supported schema functions can be found in Table I and Table II.

TABLE II
Overview of general schema function supported by eRQL.

Syntax	Description
classes(), c(), C()	Will return all defined classes.
container(), con(), CON()	Will return all defined containers.
literals(), l(), L()	Will return all used literals.
properties(), p(), P()	Will return all defined properties.
reifiedStatements(), rs(), RS()	Will return all defined reified statements.
triples(), t(), T()	Will return all triples.

ACKNOWLEDGMENT

I want to thank Vassilis Christophides (ICS-FORTH) who started my interest on RDF in 1999 and for the successful cooperation during the years after. Thanks to Fabian Wleklinski for his work on *eRQL* and to IBM (laboratory in Böblingen) for supporting me in scope of the *IBM Scholar Program*.

REFERENCES

[1] Bob Jansen: Context: A real problem for large and shareable knowledge bases, Building/Sharing Very Large Knowledge Bases (KBKS'93), Tokyo, December 1993
[2] Patrick Hayes and Brian McBride: RDF Semantics, W3C Recommendation, Feb. 2004
[3] Contexts for RDF Information Modelling, Graham Klyne, http://www.ninebynine.org/RDFNotes/RDFContexts.html
[4] RDF Primer, W3C Recommendation, Feb. 2004
[5] Karsten Tolle and Fabian Wleklinski: Trust and context using the RDF-Source related Storage System (RDF-S3) and easy RQL (eRQL), Berliner XML Tage 2004
[6] Chris Bizer: The TriG Syntax, Working Draft, online at: http://www.wiwiss.fu-berlin.de/suhl/bizer/TriG/
[7] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis and K. Tolle, The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, 2nd International Workshop on the Semantic Web, in conjunction with Tenth International World Wide Web Conference (WWW10), Hongkong, May 2001
[8] Robert MacGregor, In-Young Ko: Representing Contextualized Data using Semantic Web Tools, International Workshop on Practical and Scalable Semantic Systems (PSSS1) October 2003
[9] Jeremy J. Carroll, Patrick Stickler: RDF Triples in XML. Extreme Markup Languages 2004, in Montréal Canada.
[10] Jeremy J. Carroll, Christian Bizer, Patrick Hayes and Patrick Stickler: Named Graphs, Provenance and Trust, HP internal report HPL-2004-57, May 2004
[11] The American Heritage Dictionary of the English Language, Fourth Edition 2000, Houghton Mifflin Company
[12] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis: *On Storing Voluminous RDF Descriptions: The case of Web Portal Catalogs*, 4th International Workshop on the Web and Databases (WebDB'01), May 24-25, 2001.
[13] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, K. Tolle: Querying the Semantic Web with RQL, Computer Networks and ISDN Systems Journal, Vol. 42(5), August 2003