

Trust and context using the RDF-Source related Storage System (RDF-S3) and easy RQL (eRQL)

Karsten Tolle¹, Fabian Wleklinski²

¹ Institut für Informatik, Datenbanken und Informationssysteme (DBIS)
Johann Wolfgang Goethe-Universität Frankfurt am Main
D-60325 Frankfurt am Main
tolle@dbis.informatik.uni-frankfurt.de

² eWorks
D-60487 Frankfurt am Main
Wleklinski@eworks.de

Abstract. There exist different ways how to understand and use context information for RDF data. This paper summarizes these different ways and concentrates on the source information. It shows what can be achieved by knowing the source information and will present the *RDF-Source related Storage System (RDF-S3)*. RDF-S3 is an application implementing one way to keep track of the source information for each stored RDF triple. On top of RDF-S3 we developed an extended version of *easy RQL (eRQL)* that makes use of the source information supported by RDF-S3. Therefore queries can be restricted to trusted sources and results can be viewed inside their RDF graph context.

1 Introduction

The amount of data inside the World Wide Web is increasing continuously. That's why it is getting more and more difficult to retrieve relevant information by using current search engines that are based on pattern matching. The Semantic Web could solve this problem by annotating metadata and thus enabling semantic search engines. The key is that machines should be able to understand metadata – not only by syntax, but also on it's semantic level. These metadata of the Semantic Web are therefore encoded by the *Resource Description Framework (RDF)*. 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*.

There already exist systems which store RDF triples. However, storing triples without being able to track back to their source is not sufficient for many applications. Especially in RDF, where *everybody can say anything about everything*, it is mandatory for the users to know the source of the given information. Otherwise, users are not able to trust them. In section 2 this issue will be discussed further and

even extended by the notion of context, which can be much more than just the source information.

The RDF-Source related Storage System (RDF-S3) is an application for storing RDF triples. Furthermore, it keeps the source information of each triple and it is therefore possible to track back to their source. In addition, RDF-S3 implements a mixture of the generic representation (*GenRepr*) [A101b] and the schema specific representation (*SpecRepr*) [A101b] to overcome the drawbacks each of these representations has on its own. Section 4 introduces the internal structure of RDF-S3 and will mention how one can profit of the additional source information kept for each triple.

On top of RDF-S3 we further developed and extended our query language called *easy RQL* (*eRQL*). The peculiarities of eRQL are a) the possibility to run queries only against certain, selected sources kept in RDF-S3 and b) the ability to see the returned results inside their RDF graph context. Section 5 gives a very short introduction into eRQL.

Section 6 points out our conclusion and future steps and finally section 7 gives an overview of related work and other systems having similar targets.

2 Discussion about context, provenance and quads in RDF

The source information of (RDF-) data is important for knowledge bases. Without the ability to track back the source of data, users will probably not trust them. Imagine a tool answering your question without telling you, which sources it has used, where they can be found and without naming an author, who would really trust and use this tool? Moreover, the source information is needed for update purposes and other features that will be shown later.

In RDF it is even more important to keep the source information of data, since everybody can say anything about everything. The following example shows how much damage can be done if you are not able to keep track of the source information.

Example: Suppose there is a product P that is offered by different providers. Let us further assume that the RDF graph shown in Figure 1 is stored inside our knowledge base. The graph "tells" us that there is a company xyz that offers the product P for a price of 100 \$, and that there is another company abc that offers the same product for 80 \$. Well, this might be true, but it also might be that this graph was placed into the Web by the company abc (or another competitor), to show that he sells P cheaper than xyz – this might be true, but it might be faked as well (we all know internet spam). In reality it might be that company xyz sells P for just 70 \$. This means that we cannot trust this information. It means, that we cannot trust any information without knowing their sources, without having "evidences". If we knew that each offer was grabbed from the corresponding company's web site, it would be much easier to trust the given information.

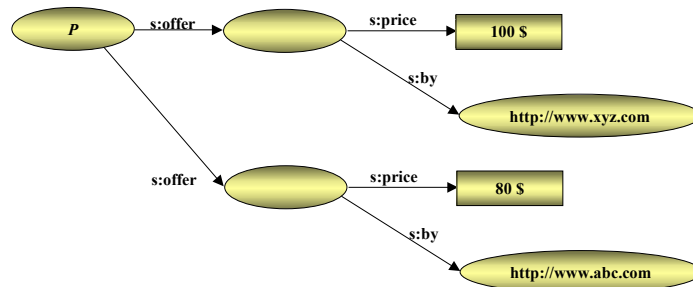


Figure 1. Example RDF graph representing two offers for a product *P*.

Also Edd Dumbill showed the importance to have the ability to track the source information within RDF data [Du03]. His consequence was to include a *context part* into each triple. With the context part he is now dealing with quadruples (quads), where as Dave Beckett argues in [Be04] that using quads would not be compatible with the RDF model and is therefore a non-goal. Instead of quads he proposes a *context node* to be added together with each triple to the RDF storage. However, is there really a big difference between a quad and a triple with an additional context node? The difference is that with quads the context part can be used inside the triple part of other quads, while the context node is just an extra information that can be used by applications but can not be used inside other triple parts, hence in RDF. Thus using context nodes in addition to triples leaves the RDF model untouched and existing RDF tools can be used without changes. Using quads the model is changed and therefore already existing RDF-tools can not be used any longer.

However, by using quads it is easier to express certain things. Edd Dumbill for example expressed by referring to the context part by whom and when the last update had been performed. The problem here is that there is no commonly fixed semantic for the context part of a quad, which means that each application using quads needs to be adjusted to the corresponding semantic. Chris Blizer goes even one step further by introducing a stating part in addition to his cRDF Data Model [Bi04]. For certain applications it might be worth it to explore its possibilities. We believe that it is too granular and therefore too complicated for common use.

A possibility to hold on to the RDF model and still be able to model context information is to use the *RDF reified statements*. In [MK03] Robert MacGregor and In-Young Ko are pointing out that this solution is not practical. The main reasons are that a) it results in a blow-up of needed triples, b) it is difficult to read, c) it is difficult to write queries and finally d) it is much more difficult to handle and therefore less efficient.

So far we used the term *context* without any definition. We can find common definition for context in dictionaries, e.g., in [Pi00] 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. An overview of existing interpretations of the term *context* in the area of knowledge base systems can be found at [Ja93].

RDF is used to represent knowledge, we therefore have the same problematic defining context in RDF as we have in knowledge bases. However, let us have a look what different ways of using context we currently find in RDF. Context information in RDF so far is used for triple identification, to merge/unmerge graphs, for keeping track of the source, to handle sub graphs more easily, to handle privileges for insertion and access, etc. Beside this, the context also affects the semantic/meaning. One could say that, e.g., properties have a fix identifier (URI) and therefore their semantic is fixed, as defined in the according namespace. However, in RDF the definition of a property can be extended inside other namespaces. This means the semantic changes whether or not you take the additional namespaces into account. Furthermore, the namespaces themselves are also not reflected inside the RDF model. This means restricting properties not to be modified in other namespaces as the one they are defined in, as forced in [A101a], could be one way to solve this particular problem, but would not be conform to the RDF model. For a clearer semantic it is also forced in [A101a] that each property should have a defined range. For the property *rdf:value*, which does not have a range definition, we can read in [HM04]: "...the intended meaning is often clear from the context ...". In this case the graph surrounding a triple is mend to be the context of the triple.

To summarize, we have three different situations where the term *context* is used in RDF. First, the context given by the surrounding graph, we call this *internal context*. The way how to handle and interpret this internal context is mainly discussed in [HM04]. The second situation we call the *external context*, like source information, time of creation, name of the author and much more, that are not included into the RDF model itself, thou they could be. Third, the context used to identify triples for a clear and easier handling of sets of triples, e.g., to merge/unmerge graphs (since this identification is not coming from inside the RDF model).

Let us concentrate on the external context and more precisely on the area of keeping track of the source of information, which was already been mentioned at the beginning of this section without a clear definition. We use the term *source of information* for indicating the place the information came from. There exists also the term *provenance* which might be used as a synonym. But provenance information is much more. It includes the information about changes that have been done during the lifetime of the information. Below, you can find a short example showing the difference of source and provenance information.

Example: The RDF/XML file with the URL <http://www.A.de/G.rdf> was created at time t_1 by a person *A*. It was then moved or copied by person *B* at time t_2 to <http://www.BC.us/G.rdf> and finally was changed at a later time t_3 by person *C*. When the file <http://www.BC.us/G.rdf> is stored into a repository it's URL (<http://www.BC.us/G.rdf>) becomes it's *source information*. However, it's *provenance information* includes the information by whom the information were produced, who changed them and when this was done. In this case:

t_1 : <http://www.A.de/G.rdf> created by *A*
 t_2 : <http://www.A.de/G.rdf> copied to <http://www.BC.us/G.rdf> by *B*
 t_3 : <http://www.BC.us/G.rdf> changed by *C* [changes]

To keep track of all provenance information from the creation until the point in time you explore the data is impossible in many cases. When we think of data from the Web there normally is no way to explore their provenance. However, the source information is always available!

Our conclusion of this discussion is, that there are two main arguments for using context nodes instead of quads. First, quads are not compatible with the RDF model and second, the distinction between the given RDF information and information that is given in addition, like external context information, is much more complicated when using quads, whereas an additional context nodes can be easily distinct from RDF triples. Especially for trust systems this is an important issue. We therefore decided to use context nodes instead of context parts (quads).

The *RDF-Source related Storage System* (RDF-S3) implements the idea of keeping the source URL for each triple by an additional context node, without affecting the RDF data model. Since the semantic for the source URL is clearly defined, RDF-S3 can benefit from it by supporting additional information. Currently RDF-S3 keeps for a specific source the time information of the last done update. This information could be used for an automated updating system. It can also easily be extended by further information, such as: "who added the specific source to the system?"

3 RDF-Source related Storage System

The Semantic Web is not established in the industry yet. To do so, it is important to provide storage systems that will build upon existing and well known systems like (O)RDBMS and SQL. Therefore, we concentrated our work and research on this area. However, there are also other approaches using different database paradigms, e.g. [WLS03,KR03], that should be explored to enhance the Semantic Web when it is established.

The RDF-Source related Storage System (RDF-S3) is an application that:

- keeps track of the source of each stored triple, without affecting the RDF model,
- allows update and deletion of inserted sources,
- tries to overcome the query performance problems of the GenRepr and SpecRepr approaches by merging these approaches,
- can be used with any SQL3 conform RDBMS via JDBC (tested with *IBM DB2 Universal Database* and *MySQL*),
- provides graphical user interfaces for easy handling,
- provides an API to handle the additional source information of each triple.

The internal structure of RDF-S3 is shown in Figure 2. It comprises of a loader to store the data into the repository and an API to access the RDF data together with their source information. You can see also in Figure 2 that the loader is build on top of the ICS-Validating RDF Parser (VRP) [To00]. This is similar to the database loader RSSDB of the ICS-RDF-Suite [A101a] (RSSDB does not support a source tracking). Using VRP gives the user the ability to validate the data on a semantic level against RDF constraints, e.g., domain and range constraints, before entering them into

the database. It is important to note that the validation of the single source does not guarantee that the combination of different sources will be valid. Anyway, working in this area showed that the semantic validation of VRP results in a clear improvement of data quality.

To enable this validation an internal in memory model needs to be created. For very large files this might not be practical, since the size of memory is limited. RDF-S3 therefore allows additionally a stream based insertion of triples without semantic validation, that does not need such an in memory model. It is worth noting that stream based insertion takes more time, since RDF-S3 has to enter each triple separately and can not benefit of the sorting and knowledge provided by the internal VRP model. Another way to handle big RDF files is to split and load them separately. It goes without saying that wrong splitting can cause errors, too.

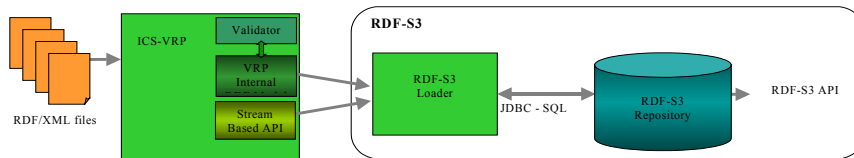


Figure 2. Overview of the internal structure and workflow of RDF-S3.

The RDF-S3 API extends classes that are used for the VRP internal RDF model. Depending on the Java type casting, the Java objects returned by the RDF-S3 API can either be viewed as representing pure RDF, or as representing RDF enriched with source information. This means that tools just dealing with the RDF model (without dealing with source information) can be easily linked up with RDF-S3.

To load RDF data into the database the graphical user interface (GUI) can be used. The GUI provides the ability to enter the information for the database connection and authentication and the settings for the underlying VRP parser.

Before the data can be loaded into the database some initial tables need to be created. Table 1 gives an overview and a short description of these initial tables. This initialization can be started under the menu item *Database* of the GUI. The namespaces of RDF and RDFS belong to each RDF graph. Therefore, they will be stored into the repository directly after creating the initial tables. To do so, RDF-S3 uses the RDF/XML serialization as it is stored within VRP's vocabulary package, but they can be updated later on as any other source.

Table 1. The tables that will be generated during the initialization phase of the database.

Database Table Name	Description
Resources	For storing all resources and literals and mapping them to their internal integer IDs.
Sources	For keeping the information about the already inserted sources.
Namespaces	Used to abbreviate the URIs of classes, properties and other constructs.
Literals	Contains literals including their type and language specification.
POI	Point-Of-Interest table, the monolith table for the GenRepr.
Classes	For keeping all stored classes.

Classes Source Usage	Stores the source relationship of the classes.
Subclasses	An extra table for the class hierarchy.
Properties	For keeping all stored properties.
Properties Source Usage	Stores the source relationship of the properties.
Subproperties	An extra table for the property hierarchy.
Containers	For keeping all stored containers.
Containers Source Usage	Stores the source relationship of the containers.
Statements	For keeping all stored reified statements.
Simple	Keeps the information if the DBMS or RDF-S3 takes care for the ID generation.

As a first benefit RDF-S3 allows the deletion and update (delete and reenter) of single sources. In addition to the source URL the system provides the information when the last update or insertion of the specific resources took place. A separate GUI (as shown in Figure 3) allows an easy handling for deleting and updating already inserted sources. The RDF and RDFS namespaces you see in Figure 5 with the IDs 1 and 2 are handled as normal sources and can be updated in the same way.

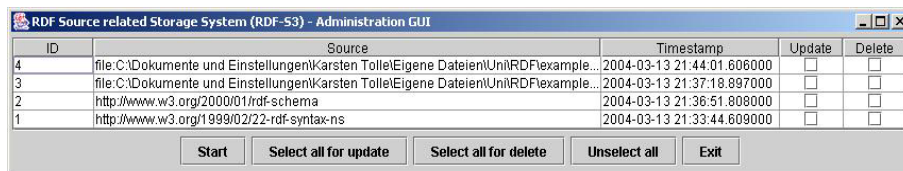


Figure 3. Graphical user interface to update or delete single sources from the RDF-S3 storage.

The internal storage structure of RDF-S3 combines the two storage approaches (GenRepr & SpecRepr) by redundant storage¹. This way both query types (schema queries & data queries) can run on the representation that is more effective for its type. Naturally redundant storage needs extra memory and the DBMS needs to do some extra work for keeping the information synchronized. The synchronization is ensured by a widespread usage of foreign keys. However, since this extra work is only needed during the insertion and deletion, this does not affect the query performance. To minimize the storage volume (especially because of the redundant storage), we decided to encode the resource URIs as integers. Of course this results in some extra work during the retrieval of the resource URIs.

A first version of RDF-S3 was published in October 2003. It is a 100% pure Java™ open source application². The system was implemented and tested on *IBM DB2 Universal Database v8.1 (ESE)*. In addition, it was tested with *MySQL 4.0* where the loading worked fine. However, during the deletion of single sources RDF-S3 makes usage of nested queries, which are not supported by *MySQL 4.0*. Nested queries will be supported by *MySQL 4.1*, but unfortunately this version is not

¹ In contrast to the SpecRepr, RDF-S3 so far does not differ between schema and meta-schema level.

² Published as a free open source Java implementation under GPL compatible license of *Johann Wolfgang Goethe-University DBIS (Germany)* and *ICS-FORTH (Greece)*. RDF-S3 can be downloaded at: www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/RDFS3/.

fully stable yet. Furthermore, it is important to mention that RDF-S3 uses foreign keys. Currently only the InnoDB MySQL storage engine supports foreign keys. Without a support of foreign keys a proper deletion and update can not be performed.

There certainly is much more we could keep in addition to the source information. There is the time of loading (which is already kept by RDF-S3), the person who loaded the source to the storage, trust ratings about the source (e.g., by an external rating services) or even provenance information (if available) as described in section 2. Yet, there are already many ways on how to benefit from just keeping the source information in addition to each triple. In the following list you can find some of our ideas:

- It provides the ability to delete complete sources and therefore also to update them. It is important to point out here that this update does not include schema evolutions. This is an extra part that needs to be explored separately and goes beyond the scope of this paper. Anyway, keeping the source information for each triple can be useful for a schema evolution within an RDF storage. It might be that the schema evolution should not affect all graphs from every source.
- In addition with the time of loading and updating, it could be used as a starting point for a versioning system inside the RDF storage.
- It offers the possibility to refer to the source and therefore to retrieve the source itself to look for further information. This could be interesting for frequently changing data or for just viewing a human readable version, e.g. in HTML.
- The source information can be used for a first basic trust mechanism. Remember the example related to Figure 1 of section 2. In case that you know that the source of the given triples belongs to the company that makes this offer, you could assume that this information is true. Moreover, by rating the different sources you could judge how much they can be trusted.
- It helps to explore contradictions between different sources. E.g., two classes are stated to have no intersection (e.g. by *owl:disjointWith*), but there exists a resource being instance of both classes. For the system there is no way to solve the situation, but by supporting the user with the URLs where the information was found, he has the ability to decide which one he trusts. An advanced system could further try to help the user by supporting him with statistics, like the numbers of sources that are stating one or the other point of view, or feedback from other users that explored the contradiction before.

4 easy RQL (eRQL)

To make usage of the source information kept in RDF-S3, we further developed the query language *easy RQL (eRQL)* [W104]. The name was given, because eRQL was originally designed on top of the *RDF Query Language (RQL)*. The implementation that comes with RDF-S3 goes a different way and accesses directly the RDF-S3 data for performance reasons. The query results of eRQL contain the fitting triples plus their context node (the source information). One main design-goal for this query language was to be as simple as possible. The user of the query language should be

able to create queries without knowing the schema or the way the data is stored. The idea was that the user can simply enter search strings, similar to current query engines he already knows (like *Google*) and retrieves reasonable results. This goal was achieved by the possibility of *one-word-queries*, e.g. "Picasso", that can be combined using boolean operators. The most valid Google-queries are valid eRQL-Queries as well. In addition eRQL supports three different modes, namely *Statement-Mode*, *POI-Mode* and *Document-Mode*:

The *Statement-Mode* corresponds to queries of existing RDF query languages and returns the triples plus their context node that fit the query. This mode is activated by enclosing the query into brackets "[...]".

The *POI-Mode*, which stands for *Point Of Interest-Mode*, includes for each triple fitting the query also the surrounding triple up to the distance defined via the query. This way we try to overcome the problem that many information and returned results are only interpretable by knowing their surrounding triples (hence the *internal context*). The POI-Mode is default but can be explicitly activated by enclosing the query into braces "{...}". In case the included neighborhood should be extended the braces can be used cumulative, e.g. "{{Picasso}}" includes all statements containing 'Picasso' plus their neighborhood with a distance of two (thus the neighborhood's neighborhood). It is the default modus which means the queries "Picasso" and "{Picasso}" are equal. Alternatively, for a more easy way to write the symbol "~" can be used, e.g. "~Picasso".

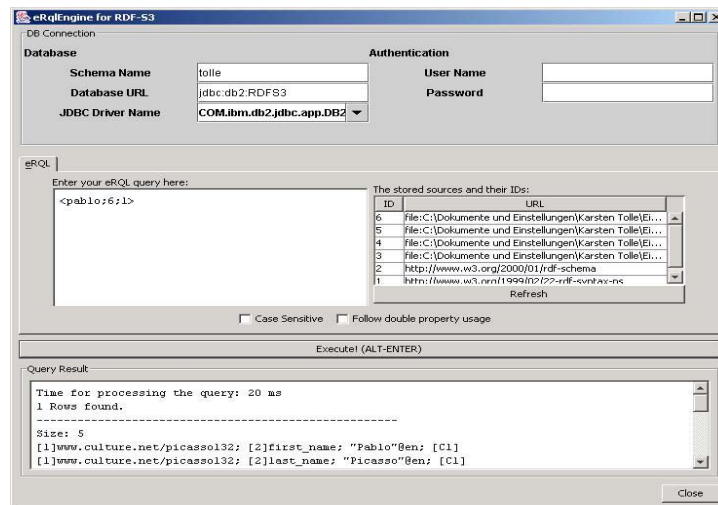


Figure 4 Screenshot of the GUI for retrieving data from RDF-S3 using eRQL.

Finally there is the *Document-Mode* which we changed from the original version described in [W104]. In this mode the source information is not only returned in the result but also used for querying itself. It needs three inputs, an eRQL query q , a list of source Ids or URLs $source-list$ (separated by comma) and b which must be either 0 or 1. The mode is active by enclosing these three into a left angle bracket and a right angle bracket and by separate them with semicolons: "< q ; $source-list$; b >". Depend-

ing on b , the query q is either evaluated only on the given set of sources ($b = 1$), or it is evaluated on all sources stored in RDF-S3 except those specified in the *source-list* ($b = 0$). Using this mode the user can select or unselect sources he does or doesn't trust. As shown in the user can retrieve the source ID information from the table on the right side in the middle of the GUI.

shows the query "`<pablo;6;1>`". This means we are searching only for those triples contained in the source with the ID 6 containing the string "pablo" in either one of their resource URIs or their literal value. Since the POI-Mode is default, also the surrounding triples with a distance of 1 will be included into the result. The complete result for the given query is shown in Figure 6. After denoting the processing time for the query, we read in the result: *1 Row found.* This means there is just one triple fitting the query, we call it a *hit*. Afterwards the result returns each hit plus its internal context with a distance as defined in the query (one in this case). The hit itself will always be in first position. Figure 6 also shows that for each triple also the source information is returned.

```

Time for processing the query: 20 ms
1 Rows found. ——— number of hits
-----
Size: 5
[1]www.culture.net/picasso132; [2]first_name; "Pablo"@en; [C1]
[1]www.culture.net/picasso132; [2]last_name; "Picasso"@en; [C1]
[1]www.culture.net/picasso132; [2]paints; [3]woman.qti; [C1]
[1]www.culture.net/picasso132; [2]paints; [3]guernica.jpg; [C1]
[1]www.culture.net/picasso132; [4]type; [2]Cubist; [C1]
-----
Namespaces:
[1] file:C:\Dokumente und Einstellungen\Karsten Tolle\Eigene
Dateien\Uni\RDF\examples\RDF_examples\culture_data2.rdf#
[2] file:D:\tmp_local_users\alexaki\RDF_vrp2_5\RDF_examples\culture.rdf#
[3] http://www.museum.es/
[4] http://www.w3.org/1999/02/22-rdf-syntax-ns#
Sources:
[C1] file:C:\Dokumente und Einstellungen\Karsten Tolle\Eigene
Dateien\Uni\RDF\examples\RDF_examples\culture_data2.rdf

```

Figure 6 Result for the query `<pablo;6;1>`, given as the output of the eRQL GUI of RDF-S3. The ID 6 used in the query is the internal ID from RDF-S3 for the file `culture_data2.rdf`.

Additional new developed features of eRQL are:

- The possibility to choose between a case sensitive and a non case sensitive search.
- A one-word-query q can be restricted to be evaluated only on resources (by using: $res(q)$) or on literals (by using double quotes: " q ").
- Wildcards "*" and "?" can be used, whereby "*" stands for any string and "?" for exact one character.

5 Future Work and Conclusion

With RDF-S3 we provide the possibility to keep track of the source information. By using context nodes we can easily distinguish between triple information coming from

outside and the information generated by RDF-S3. This increases the credibility of the system. With eRQL, users of RDF-S3 can profit by the source information to concentrate on trusted sources and to get the information where the single triples are coming from, which means he sees the *external context* of the results. Additionally eRQL provides an easy way to see the results in their *internal context* by including surrounding triples for each hit.

RDF-S3 needs some extra work during the insertion and deletion. This is also reflected in the processing time needed. Therefore, one of the main goals is to improve the performance for these phases to minimize this drawback. A performance improvement can be achieved by optimizing the parameter settings of the underlying database and by shifting parts of RDF-S3 to the database side to reduce the communication overhead (by using stored procedures).

To further improve the query performance, it is planned to explore the usage of labeling schemas [Ch03] for class and property hierarchies inside RDF-S3. For better scalability reasons, storage variations will be tested, e.g., variations for the SpecRepr, as described in [Ka03], to reduce the number of tables. In addition it is planned to run tests with resource URI abbreviations by substituting the namespace part as done in Jena [Mc02], which is so far only done for classes and properties within RDF-S3 but not for simple resources.

For a more easy reading and understanding of the returned results of eRQL we are working on a system to visualize the RDF result graph. In this area we have in mind to restrict the shown graph to a certain maximum size that is manageable by the users. The results are therefore shown separate for each hit, and by a ranking system more important information are highlighted.

6 Related Work

The Redland RDF Application Framework [Be04] and RDF-S3 are both using context nodes. RDF-S3 benefits from a clearer semantic for it. However, for some applications it might be worth to modify or extend this semantic. Another difference is the underlying programming language which is C for the Redland and Java for RDF-S3.

In Jena [Mc02] you can name a model you enter to the database, which can be used for source information. The main difference to RDF-S3 is Jena's different storage approach that is based on the generic representation. The drawbacks of this approach are discussed in [Al01b].

The RDF Gateway [Ne03] of Intellidimension is based on a deductive database and provides an optional context with each triple. The context node in this system can be used either to identify the source of a statement or as a security context to handle privileges for actions on a table like select, delete or insert. They also use the term quads but their context does not affect the RDF model and is therefore closer to Redland and RDF-S3.

RDF-S3 is influenced by the ICS-FORTH RDFSuite [Al01a]. It uses the ICS-VRP and includes the SpecRepr. eRQL was influenced by RQL [Ka03] and RDQL [Se04]

but both languages are much more complicated and do not support context information in the way eRQL does.

References

- [Al01a] 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
- [Al01b] 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 2001
- [Be04] Dave Beckett: Redland RDF Application Framework – Context, online at: <http://www.redland.opensource.ac.uk/notes/context.html>
- [Bi04] Chris Bizer: Brainstorming about Trust, Context and Justification, online at: <http://www.wiwiss.fu-berlin.de/suhl/bizer/trustcontextjustification/>
- [Ch03] V. Christophides, D. Plexousakis, M. Scholl, S. Tourtounis: On Labeling Schemes for the Semantic Web, 12th International World Wide Web Conference (WWW'03), May 2003
- [Du03] Edd Dumbill: Tracking provenance of RDF, July 2003, online at: <http://www-106.ibm.com/developersworks/xml/library/x-rdfprov.html>
- [HM04] Patrick Hayes and Brian McBride: RDF Semantics, W3C Recommendation, Feb. 2004
- [Ja93] Bob Jansen: Context: A real problem for large and shareable knowledge bases, Building/Sharing Very Large Knowledge Bases (KBKS'93), Tokyo, December 1993
- [Ka03] 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
- [KR03] Joseph B. Kopena and William C. Regli: Design Repositories for the Semantic Web with Description-Logic Enabled Services, First International Workshop on Semantic Web and Databases, Sept. 7-8 2003
- [Mc02] The Jena Semantic Web toolkit, Brian McBride, Jena Project, Hewlett-Packard Laboratories (HPL), O'Reilly xml.com, <http://www.xml.com/pub/r/1285>, 2002
- [MK03] Robert MacGregor, In-Young Ko: Representing Contextualized Data using Semantic Web Tools, International Workshop on Practical and Scalable Semantic Systems (PSSS1) October 2003
- [Ne03] Andrew Newman, JRDF (Java RDF), online at: <http://jrdf.sourceforge.net>
- [Pi00] The American Heritage Dictionary of the English Language, Pickett, Joseph P. et al. (Ed.) Fourth Edition 2000, Houghton Mifflin Company
- [Se04] RDQL – RDQL - A Query Language for RDF, Andy Seaborne, W3C Member Submission 9 January 2004, online at: <http://www.w3.org/Submission/RDQL/>
- [To00] Karsten Tolle: Analyzing and Parsing RDF, Master's thesis, University of Hannover, 2000
- [Wl04] Fabian Wleklinski: Suche im Semantic Web – Erweiterung des VPR um eine intuitive und RQL-basierte Anfrageschnittstelle, Master's Thesis, University of Frankfurt, 2004
- [WLS03] Timo Weithoener, Thorsten Liebig, and Guenther Specht: Storing and Querying Ontologies in Logic Databases, First International Workshop on Semantic Web and Databases, Sept. 7-8 2003