Querying Dynamic and Context-Sensitive Metadata in Semantic Web

Sergiy Nikitin, Vagan Terziyan, Yaroslav Tsaruk, Andriy Zharko

Industrial Ontologies Group, Agora Center, University of Jyväskylä
P.O. Box 35, FIN-40014 Jyväskylä, FINLAND
e-mail: seniki@cc.jyu.fi

Abstract. RDF (core Semantic Web standard) is not originally appropriate for context representation, because of its initial focus on the ordinary Web resources, such as web pages, files, databases, services, etc., which structure and content are more or less stable. However, on the other hand, emerging industrial applications consider e.g. machines, processes, personnel, services for condition monitoring, remote diagnostics and maintenance, etc. to be specific classes of Web resources and thus a subject for semantic annotation. Such resources are naturally dynamic, not only from the point of view of changing values for some attributes (state of resource), but also from the point of view of changing "status-labels" (condition of the resource). Thus, context-awareness and dynamism appear to be new requirements to the existing RDF. This paper discusses the issues of representing the contexts in RDF and constructions coming with context representation. We discover certain representation patterns and their classification towards development of the general approach of querying dynamic and context-sensitive metadata in Semantic Web by autonomous agents.

1 Introduction

Emerging Semantic Web technology offers a Resource Description Framework (RDF) as a standard for semantic annotation of Web resources. It is expected that Web content with RDF-based metadata layer and ontological basis for it will be enough to enable interoperable and automated processing of Web data by various applications. RDF-based tools, e.g. Hewlett-Packard's Jena [14] and Stanford's Protégé [15] provide a base for reasoning about metadata and about situated data (entities situated in time and space) that is superior to alternatives such as relational databases or object-oriented databases. However, according e.g. to [10] essential representational ability is missing from the current generation of Semantic Web tools and languages. When that ability is added, the resulting capabilities offer a combination of novelty and flexibility that may usher in a wave of commercial Semantic Web tool-based applications. Evidently the existing RDF tools should be extended to support contexts to enable querying a set of RDF statements having common temporal, spatial or other metadata attributes. In [10] it was concluded that the "clear winners" for possible solution can be quads (i.e. adding a fourth field of type 'context' to each RDF triple) and a context mechanism that references individuals instead of statements. Another attempt has been made recently to add C-OWL (Context OWL), an

extended language with an enriched semantics which allows us to contextualize ontologies, namely, to localize their contents (and, therefore, to make them not visible to the outside) and to allow for explicit mappings (bridge rules). The core open issue is the tension between how much knowledge should be shared and globalized (via ontologies) and how much should be localized with limited and controlled forms of globalization (via contexts) [11]. In [12] the usage of context- and content-based trust mechanisms have been proposed and the cRDF trust architecture was presented which allows the formulation of subjective and task-specific trust policies as a combination of reputation-, context- and content-based trust mechanisms. There exist different ways how to understand and use context information for RDF data. In [13] these different ways have been summarized and the RDF-Source related Storage System (RDF-S3) has been proposed. RDF-S3 aimed to keep track of the source information for each stored RDF triple. On top the RDF-S3 has 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. Two main arguments are stated in [13] 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 additional context nodes can be easily distinct from RDF triples. Therefore context nodes were used instead of context parts (quads).

There is not yet clear vision, which way is better (triples or quads) for representing contextual metadata in RDF. Another issue is for what kind of resources such descriptions will be required. On one hand the ordinary Web resources, such as web pages, files, databases, services, etc., which structure and content are more or less stable, probably do not need a specific way of context representation. However, on the other hand, emerging industrial applications consider e.g. machines, processes, personnel, services for condition monitoring, remote diagnostics and maintenance, etc. represent specific classes of Web resources and thus a subject for semantic annotation. Such resources are naturally dynamic, not only from the point of view of changing values for some attributes (state of resource) but also from the point of view of changing “status-labels” (condition of the resource). In our former effort within SmartResource project [16] we presented Resource State/Condition Description Framework (RscDF), as an extension to RDF, which introduces upper-ontology for describing such characteristics of resources as states and corresponding conditions, dynamics of state changes, target conditions and historical data about previous states. These descriptions are supposed to be used by external Web-services (e.g. condition monitoring, remote diagnostics and predictive maintenance of the resources). We presented RscDF as temporal and contextual extensions of RDF and discussed a State-Symptom-Diagnosis-Decision-Maintenance model as the basis for RscDF schema.

RSCDF is a unified representation format for resource state and condition description (encoding). RscDF-language formalizes context definition structure. RscDF-Schema defines main concepts and structure of the language. The structure is highly flexible, thus allowing definition of different complex constructions over the basic statements. Different definitions being used for resource description must refer to or define instances of classes from Industrial Maintenance Ontology. Detailed descrip-

tion of RscDF-language is not in a scope of this paper, so we refer to [17]. Figure 1 shows the key element of RscDF – "SR_Statement".

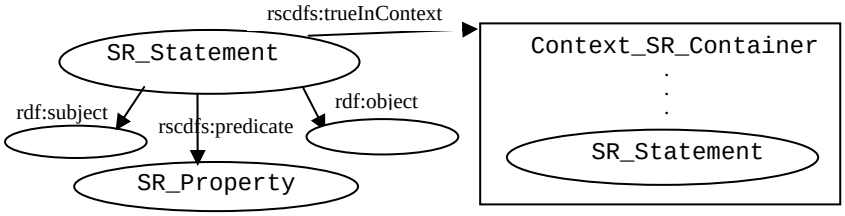

Fig. 1. SR_Statement structure

The SR_Statement defines the basic structure of statements being used in RscDF. The combinations of statements and references to statements and statement containers may form highly structured semantic description. The important semantics are represented by SR_Property class and its subproperties. The property in the rscdfs:predicate container defines the type and structure of rdf:object of current SR_Statement. However, the property specification defines only domain and range. So to know the structure of the statement, we have to attach some pattern description to SR_Property.

The RscDF language was designed to serve the concept of a Global Understanding Environment [1]. GUN concept utilises Semantic Web approach for resource annotation and ontology-based semantic representation and describes communities of interacting Smart Resources. GUN provides a framework for making resources smart, for interaction, collaboration, coordination of these resources and resource discovery support. Types of resources are not restricted to traditional web content, but can be physical resources from real world, such as humans and devices (see Figure 2).

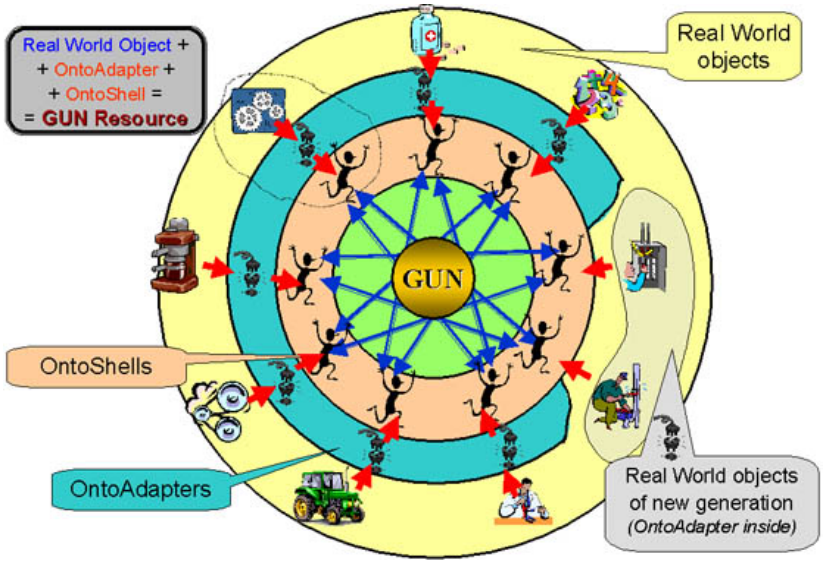

Fig. 2. GUN concept illustrated (adopted from [1])

GUN paradigm provides every participant with common structured data representation format, allowing explicit and unambiguous knowledge sharing. In order to become GUN participant certain steps of adaptation should be taken. In GUN development our research group focuses on industrial case study that is concerned with large-scale platforms for automated management of industrial objects. The adaptation process to GUN environment is described in General Adaptation Framework [18]. "General adaptation" assumes a design of a sufficient framework for an integration of different (by structure and nature) resources into the GUN environment. This environment will provide a mutual interaction between heterogeneous resources. Adaptation assumes elaboration of a common mechanism for new resource integration, and its provision with a unified way of interaction. The main idea of adaptation is based on a concept of "adapter", which plays role of a bridge between an internal representation of resource and a unified environment.

Adapter is a software component, which provides a bidirectional link between a resource interface and an interface of the environment. GUN assumes interoperability of Smart Resources. Smart Resource is a conjunction of Real World Resource (RWR), Adapter and Agent. By extending RWR within Adapter and Agent we make it GUN compatible. General Adaptation includes development of Adapter for RWR. Adaptation to GUN is not just syntactical transformation from one representation format to another. The key element of adaptation is mapping of concepts being used by "Real-World-Resource" to Industrial Maintenance Ontology (IMO) elements. The role of IMO lies in unification and structuring of data being represented in such way that every resource description taking part in GUN must refer to it.

Semantic Web standards are not yet supporting semantic descriptions of resources with proactive behavior. However, as the research within the SmartResource project shows [16], to enable effective and predictive maintenance of an industrial device in distributed and open environment, it will be necessary to have autonomous agent based monitoring over device state and condition and also support from remote diagnostics Web-Services (see Figure 3).

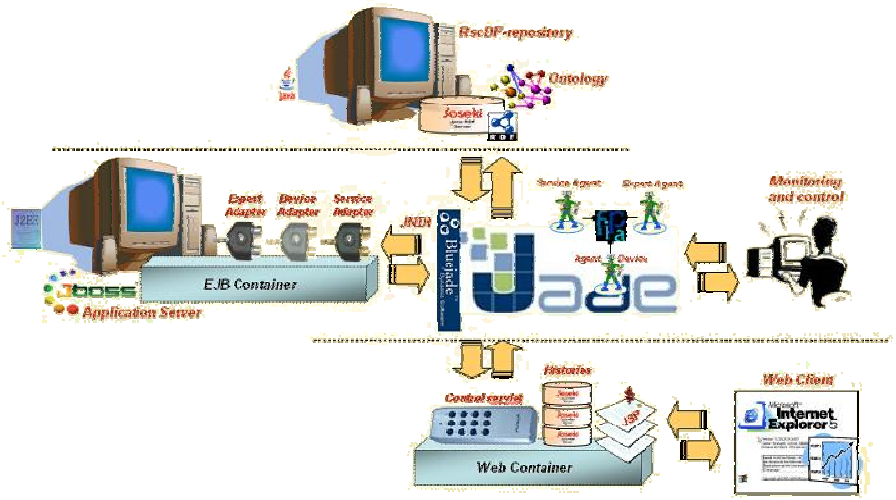

Fig. 3. SmartResource as a Multi-Agent System

This means that the description of a device as a resource will require also the description of proactive behavior of autonomous condition monitoring applications (agents, services) towards effective and predictive maintenance of the device. For that we plan to develop in 2005 another extension of RDF, which is Resource Goal/Behavior Description Framework (RGBDF) to enable explicit specification of maintenance goals and possible actions towards faults monitoring, diagnostics and maintenance. Based on RSCDF and RGBDF and appropriate ontological support, we also plan to design RSCDF/RGBDF platforms for smart resources (devices, Web-services and human experts) equipped by adapters and agents for proactivity, and then to apply several scenarios of communication between the platforms towards learning Web-services based on device data and expert diagnostics to enable automated remote diagnostics of devices by Web-services.

In this paper we present our solution how to manage (according to the structure of the paper Section 2 describes about storing and Section 3 is dedicated to querying) the context-sensitive metadata for applications compatible with Semantic Web and GUN concepts by utilising existing technologies and tools. Some examples with industrial metadata are also provided.

2 Storing RDF-Based Metadata

Nowadays there are a lot of proposals related to storing RDF data in RDF databases, each with different client-server protocols and different client APIs. For our purposes we surveyed a number of most popular RDF-storages (Kowari[1], Sesame[2], Joseki[3]) and selected Joseki storage as most suitable allowing access to RDF-data through HTTP.

2.1 Joseki

Joseki has been proposed and maintained by Semantic Web group at HP Labs. Joseki is a web application for publishing RDF models on the web and realized useful access to models through HTTP protocol. This allows getting easy access to model from anywhere you want. It is built on Jena and, via its flexible configuration, allows a Model to be made available on a specified URL and queried using a number of languages. Results can be returned as RDF/XML, RDF/N3, or NTriples. The query languages, result formats, and model sources can be extended to produce new alternatives tailored to the user's needs.

2.2 Storing and Extracting Data in Joseki

Information stored in Joseki are presented in a format of models. The client application has an access to a specified model and executes operation on this model. Operations that can be done upon the remote model:

- add new model or statement

1 http://www.kowari.org/

2 http://www.openrdf.org/

3 http://www.joseki.org/

- remove model or statement
- extract data from storage

New model can be appended to already existing model on the Joseki server. This operation also allows appending new statement to the predefined model. Each model or statement can be removed from the storage by using the remove operation.

Data extraction from Joseki storage can be implemented by using different mechanisms:

- fetch the whole model
- SPO query (single triple match language)
- RDQL query

Information from the storage can be extracted partly or as a whole model. To extract the whole model the fetch mechanism is used. For extracting just specified information, SPO and RDQL queries are available. SPO (also known as "Triples") is an experimental minimal query language. An SPO query is a single triple pattern, with optional subject (parameter "s"), predicate (parameter "p"), and object (parameter "o", if an URIref or parameter "v" for a string literal). Absence of the parameter implies "any" for matching that slot of the triple pattern.

RDQL is a query language, which is similar to SQL (Structured Query Language) and allows specifying the set of conditions, which should suite the extracted set of statements.

The architecture of Joseki is presented in Figure 4.

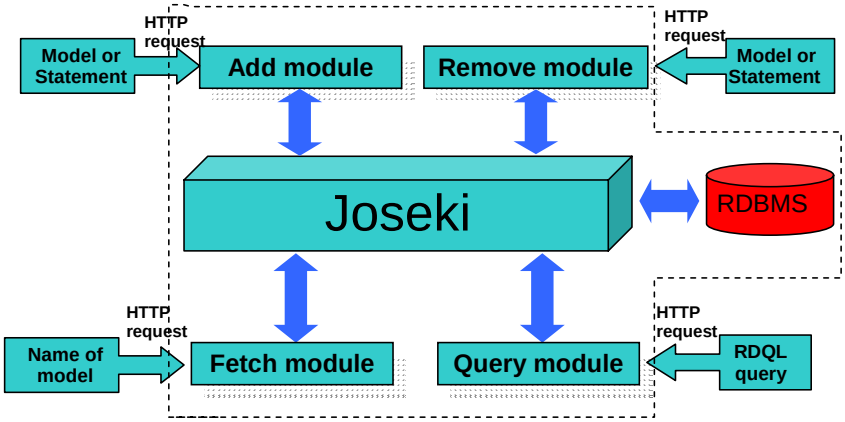

Fig. 4. Architecture of Joseki storage

It consists of the core module and modules, which execute specialized functions on the remote model (fetching, adding, removing, querying). Interaction between the client and the Joseki server is implemented through HTTP query. The type of the query depends on the length of the query. It could be GET if query is not longer then 200 characters, otherwise POST method is used. Each model in Joseki server has a predefined set of operations, which could be executed upon it. When server gets query to one of the defined models, it checks the list of operations which could be executed and if the operation is not specified it responds by a fail message. Each operation is

executed by a specified module. As an input each module requires special parameters. For example, Addition and Remove modules need as an input model or statement, which have to be added or removed. As a response Joseki sends empty model, if the operation was successful.

One more optional component is RDBMS assigned for storing models in a persistent storage. The models in Joseki can be saved in two ways (See Figure 5):

- to a file
- to a RDBMS.

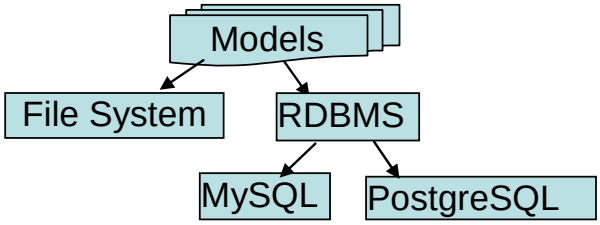

Fig. 5. Types of storing models in Joseki

The target RDBMS is specified in the configuration file joseki-db.n3.

2.3 RDQL

Resource Description Query Language (RDQL) is a query language for RDF. RDQL is an implementation of the SquishQL RDF query language and is similar to SQL. It borrows basic set of words for specifying the set of data, which should be returned (e.g. SELECT, WHERE, FROM, etc). As a condition for extracting, RDQL provides, the "WHERE" clause followed by a list of triples (subject, predicate, object). These triples define the pattern for a search. RDQL has one more key word for defining a space of URI identifiers. It allows avoiding long names.

In the sample query presented below (SELECT-query), as a result, two values will be returned: Matthew and Jones. At the beginning of query we specify the values, which should be returned: "`?family`" and "`?given`". The first condition determines a statement, which has `vcard:FN` property value "Matt Jones". Then we extract data from property `vcard:N` to the variable "name". Basing on this information, we extract values of the property `vcard:Family` and `vcard:Given`.

```
SELECT ?family, ?given
WHERE (?vcard vcard:FN "Matt Jones")
      (?vcard vcard:N ?name)
      (?name vcard:Family ?family)
      (?name vcard:Given ?given)
USING vcard FOR <http://www.w3.org/2001/vcard-rdf/3.0#>
```

Scheme on Figure 6 shows the steps of the query execution. The names of nodes are presented as names of variables to make picture clearer. The values of variables "`vcard`" and "`name`" are used as an input to the next condition statements.

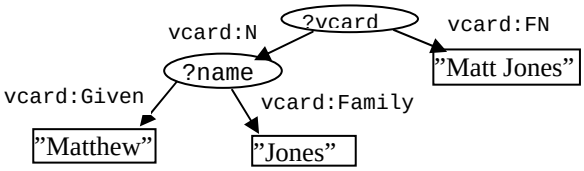

Fig. 6. Query description scheme

3 RscDF Data Management in GUN

The capabilities GUN provides rely on the common data representation format (RSCDF) and the common understanding of domain (Industrial Maintenance Ontology). As far as RSCDF is RDF-compatible, we reuse already existing RDF-databases to store RSCDF data.

3.1 Joseki RDF storage in GUN

Joseki server plays the role of a permanent storage in GUN. It stores all the information, which come from all adapters and provides convenient access for extracting data from it. The storage can be centralized or embedded into the adapter. In the scenario we implemented (Figure 7), it plays the role of a common shared storage.

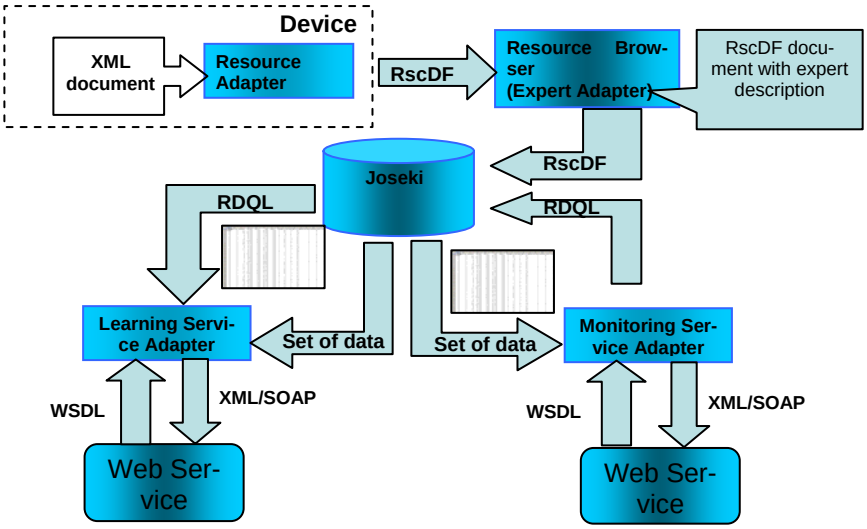

Fig. 7. Presentation scenario

According to the needs of our task the special module for interaction with Joseki server has been developed. This module allows following operations on model and

statements, such as: adding, removing and clearing the whole model. To implement model functionality the class JosekiStorage was created. Property `modelURI` pointing to model is initialized in a constructor of the class. As an input to the constructor parameters “HostName” and “ModelName” are passed.

In addition, there are classes from packages com.hp.hpl.jena.joseki and com.hp.hpl.jena.rdf.model. Classes from these packages simplify work with RDF data. Figure 8 shows the class diagram of the module.

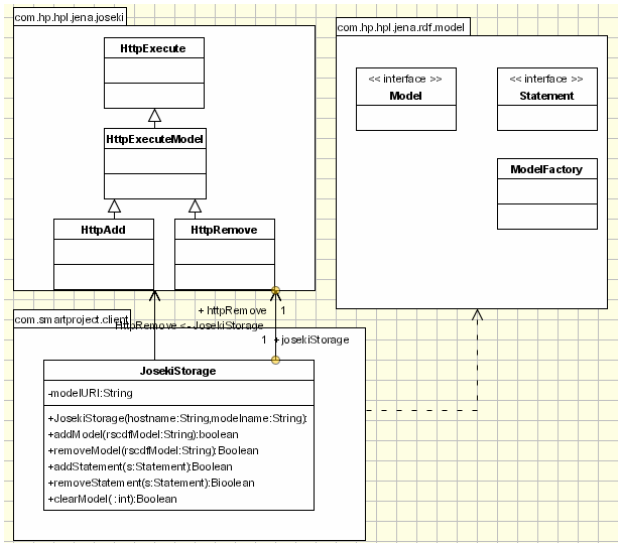

Fig. 8. Class diagram showing classes needed for interaction with the Joseki server

3.2 Applying RDQL to RscDF Querying

When querying RscDF data we deal with Statement objects that has the additional property `rscdfs:trueInContext`. When selecting a Statement about an object having certain property, we have to consequently apply queries, specifying the rdf:object, rscdfs:predicate or rdf:subject property values, so the query may look like:

```
SELECT ?stmts
WHERE
(?stmts,<rdf:subject>,<papmDescr:123456XZ24>),
(?stmts,<rscdfs:predicate>,<measureOnt:surfacelevel>)
USING
papmDescr  FOR
<http://www.cc.jyu.fi/~olkhriye/rscdfs/resource/resourc
eInstanceDescription#>,
rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#>,
rscdfs FOR
<http://www.cc.jyu.fi/~olkhriye/rscdfs/0.3/rscdfs#>,
measureOnt FOR
<http://www.cc.jyu.fi/~olkhriye/rscdfs/0.3/ontologies/m
easurementOntology#>
```

The resulting variable `stmts` will contain the set of Statements, whose subject and predicate properties satisfy the condition presented in Figure 9 in a form of a graph.

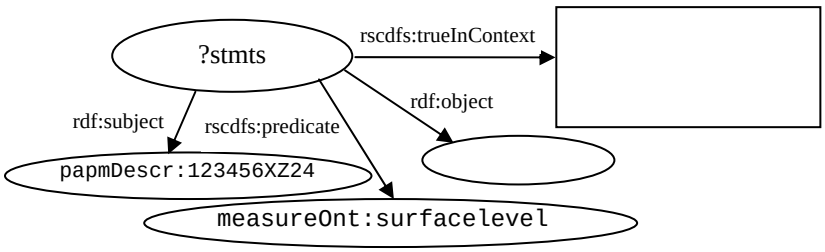

Fig. 9. Visual query representation

However, when the query contains context-related parts, we meet a problem of representing it in the RDQL language. The query becomes difficult to read, because of additional constructions. For example, when the statements describing different object properties at the certain moment of time, should be selected, we have to specify the value of time-statement lying in the context container.

3.3 Querying Patterns

RscDF language provides a facility to select Statements by a certain template. The template Statement is put to the context container of Statement, wrapping the Statements selected according to the template. Figure 10 shows the structure of Statement, being created as a result of data collection according to a certain template.

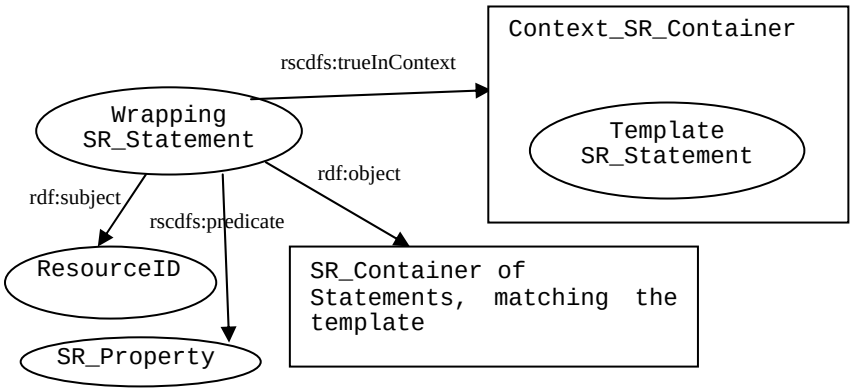

Fig. 10. Data Collection Statement selecting data according to a template

The most vivid example of the template context-dependent data collection is State-template data collection. For example, we have a certain resource, logging a track of its states. Different Statements about resource states are marked with time. So, the Statements will contain Statement about time in context container (Figure 11).

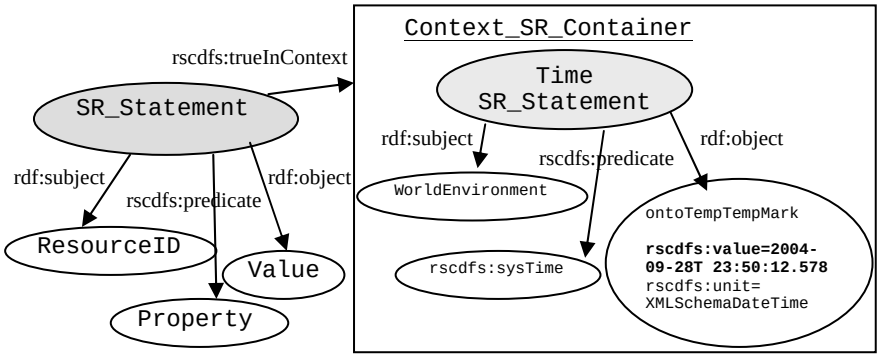

Fig. 11. Statement with time context

Figure 12 shows the data collection (subhistory) statement. The `rdf:object` property contains reference to container with Statements, matching the data collection template. The data collection template Statement is placed to the Context_SR_Container of the State Statement.

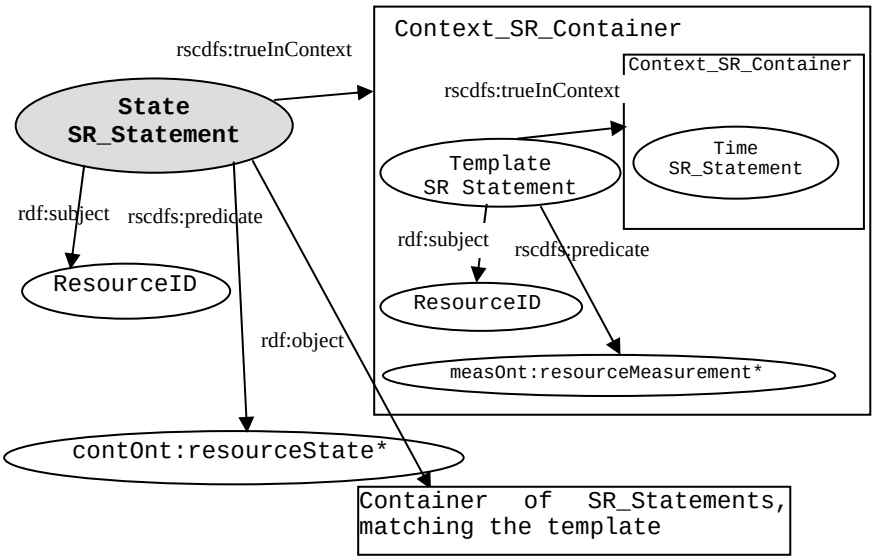

Fig. 12. Collecting Statement of State template

All the names marked with (*) are not actually present in Industrial Maintenance Ontology, but mean more generic classes or properties.

To apply the query, to data collection structures residing in RDF storage via RDQL language we have to write a number of routine triple queries, so it is reasonable to discover certain RDQL templates combining the operations into blocks and asking only start input data for further query execution. In case of State template we have discovered following routines:

– State RscDF-Statement To “attribute-value pairs” Routine:

Input	Output
Pointer to State Statement	Set of attribute-value pairs of one state

– Subhistory Statement to “Set of State Records”

Input	Output
Pointer to Subhistory Statement	Set of attribute-value pairs of correspondent states

Further on we omit namespaces definition and `USING` clause. For the first case the RDQL query looks like:

```
SELECT ?ValueStatements, ?NumUnits, ?NumValues
WHERE
(<StateStmtID>, <rdf:object>, ?StateContainer),
(?StateContainer, <rscdfs:member>, ?ValueStatements),
(?ValueStatements, <rdf:object>, ?NumValueInstances),
(?NumValueInstances, <rscdfs:value>,?NumValues),
(?NumValueInstances, <rscdfs:unit>, ?NumUnits)
```

The output of the query is a plain 3-column table with a set of rows. It is implied that every record in the table belongs to State, hence here we have 3 output variables, but in cases, when this routine is used as a subroutine, we have to return also the State Statement identifiers in order to be able to identify then relationships of values to states. Table 1 illustrates a possible output of the query:

Statement ID	Units	Value
somens:valueStatementID_1	measureUnitsOnt:temperatureCelsius	70
somens:valueStatementID_2	measureUnitsOnt:roundsPerMinute	1500

Table 1. RDQL Query output

We put Statement ID to query output, because it uniquely identifies the belonging of values and units and allows further inference upon received results. As far as RDQL query result is displayed in one non-normalized table, we store redundant data, but save the semantics.

The routine logic can be wrapped as a method, whose input is the name of Statement and output - RDQL subroutine. Example in Figure 13 shows more complex logic. It reuses previous example of State data selection, but provides a Set of States.

Below is the RDQL query:

```
SELECT?StateStmts,?ValueStatements,?NumUnits, ?NumVaues
WHERE
(<HistoryStmtID>, <rdf:object>, ?StatesCont),
(?StatesCont, <rscdfs:member>, ?StateStmts),
(?StateStmts, <rdf:object>, ?StateContainers),
(?StateContainers, <rscdfs:member>, ?ValueStatements),
(?ValueStatements, <rdf:object>, ?NumValueInsts),
(?NumValueInsts, <rscdfs:value>,?NumValues),
(?NumValueInsts, <rscdfs:unit>, ?NumUnits)
```

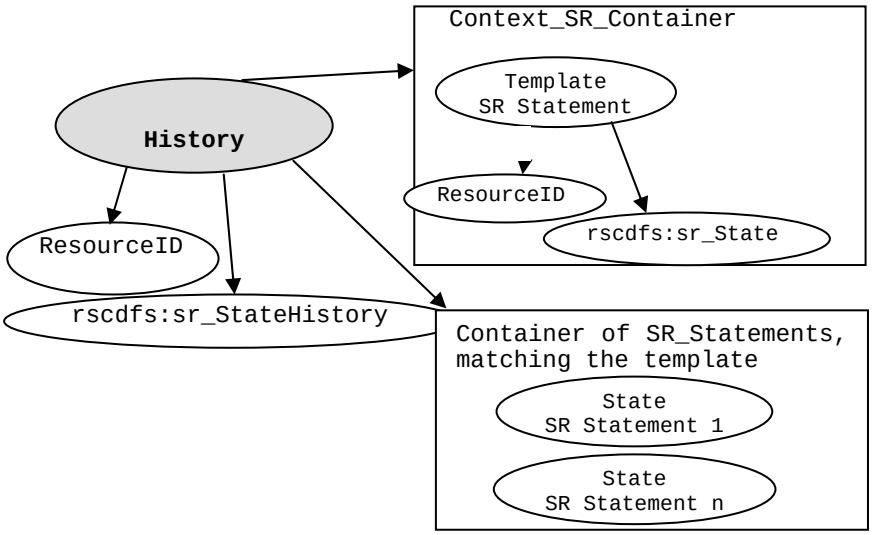

Fig. 13. History Statement

The five last strings of the query above are almost equivalent to the State query template. The only difference is presence of variable `?StateStmts` instead of static given value `StateStmtID`. Query doesn't contain any references to types of properties being used in statements because we know beforehand what kind of data we deal with. In general, when the statement's ID is not known, we should first look for it, specifying as a search criteria Resource's ID and property type, for example:

```
SELECT ?stmts
WHERE
(?stmts,<rdf:subject>,<resourceID>),
(?stmts,<rscdfs:predicate>,<rscdfs:sr_StateHistory>)
```

Basically, the SR_Property being pointed by `rscdfs:predicate`, specifies the data template. So it makes sense to develop the ontology of data templates and associate it with SR_Properties.

Conclusions

In this paper we tried to analyze the problems of storing and managing context-enabled data via RDF storages. Finally, Joseki RDF storage and querying engine has been chosen as the most appropriate for integration to the prototype platform for adaptation of industrial resources to Semantic Web – pilot system, result of the Adaptation Stage of the SmartResource project. The approach based on the RDQL-patterns has been applied in the logic of the part of General Semantic Adapter, responsible for querying RscDF storages – dynamic and context-sensitive histories of industrial resources (experts, web services and devices). The flexibility of the RDQL-patterns has

allowed to design a unified semantic adapter – a mediator between software agents (which implement proactive goal-driven behavior of originally passive industrial resources) and RDF-based storage of the history data of the corresponding industrial resources.

Further, it is planned to apply the developed method based on the RDQL-patterns in the design of querying mechanism for goal/behavior rule storages, which will utilize RGBDF – Resource Goal/Behavior Description Framework. The latter will be designed during the Proactivity Stage of the SmartResource activities as a part of the Pro-GAF – General Proactivity Framework.

Acknowledgements

This research has been performed as part of the SmartResource ("Proactive Self-Maintained Resources in Semantic Web") project in Agora Center (University of Jyväskylä, Finland) and funded by TEKES and industrial consortium of following companies: Metso Automation, TeliaSonera, TietoEnator and Science Park of Jyväskylä.

References

1. Terziyan V., Semantic Web Services for Smart Devices in a "Global Understanding Environment", In: R. Meersman and Z. Tari (eds.), *On the Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*, Lecture Notes in Computer Science, Vol. 2889, Springer-Verlag, 2003, pp.279-291.
2. Online Jena API tutorial by B. McBride, "An Introduction to RDF and the Jena RDF API", August 2003, http://jena.sourceforge.net/tutorial/RDF_API/.
3. Online Jena tutorial by A. Seaborne, Hewlett Packard, "Jena Tutorial. A Programmer's Introduction to RDQL", April 2002, http://www.hpl.hp.com/semweb/doc/tutorial/RDQL/.
4. Kevin Wilkinson, Craig Sayers, Harumi A. Kuno, and Dave Reynolds, Efficient RDF Storage and Retrieval in Jena2, In: I. F. Cruz, V. Kashyap, S. Decker, R. Eckstein (Eds.): Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Humboldt-Universität, Berlin, Germany, September 7-8, 2003: 131-150.
5. A. Barnell, RDF Objects, Technical Report, Semantic Web Applications Group, Hewlett Packard Laboratories Bristol, Avon, England, November 2002.
6. Webpage of Jena on the official website of B. McBride, Hewlett Packard, "Jena, An RDF API in Java", http://www.uk.hpl.hp.com/people/bwm/rdf/jena.
7. R. Lee, Scalability Report on Triple Store Applications, Technical Report, SIMILE project, 2004.
8. D. Beckett, Semantic Web scalability and storage: survey of free software/open source RDF storage systems, Deliverable 10.1 report, SWAD-Europe project (IST-2001-34732), 2002.

9. B. McBride, Jena: Implementing the RDF Model and Syntax Specification, HP Labs in proceedings of the Second International Workshop on the Semantic Web, WWW10, Hong Kong, 1st May 2001.
10. R. MacGregor, In-Young Ko, Representing Contextualized Data using Semantic Web Tools, In Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems, ISWC 2003, October 2003, Sanibal Island, Florida, USA.
11. P. Bouquet, F. Giunchiglia, F. Harmelen, L. Serafini, and H. Stuckenschmidt, Contextualizing Ontologies, *Journal of Web Semantics*, vol. 26, 2004: 1-19 pp.
12. C. Bizer and R. Oldakowski. Using Context- and Content-Based Trust Policies on the Semantic Web. In 13th World Wide Web Conference, WWW2004 (Poster), 2004.
13. Official website of RDF-S3 - RDF Source related Storage System, http://www.dbis.informatik.uni-frankfurt.de/~tolle/RDF/RDFS3/.
14. Official website of JENA – a Semantic Web Framework for Java, http://jena.sourceforge.net/.
15. Official website of the Protégé ontology management tool, http://protege.stanford.edu/.
16. Webpage of the SmartResource project, http://www.cs.jyu.fi/ai/OntoGroup/SmartResource_details.htm.
17. Kaykova O., Khriyenko O., Naumenko A., Terziyan V., Zharko A., "RSCDF: Resource State/Condition Description Framework", Deliverable 1.1 report, September 2004, SmartResource project, (http://www.cs.jyu.fi/ai/IJWGS-2004_v2.doc).
18. Kaykova O., Khriyenko O., Kovtun D., Naumenko A., Terziyan V., Zharko A., "GAF: General Adaptation Framework", Deliverable 1.2 report, October 2004, SmartResource project (http://www.cs.jyu.fi/ai/SJIS-2005.doc).