Towards a Modification Exchange Language for Distributed RDF Repositories

Wolfgang Nejdl[1], Wolf Siberski[1], Bernd Simon[2], and Julien Tane[1, 3]

[1]Learning Lab Lower Saxony, Expo Plaza 1, 30539 Hannover, Germany
{nejdl, siberski, tane}@learninglab.de
[2]Abteilung für Wirtschaftsinformatik, Neue Medien, Wirtschaftsuniversität Wien,
Augasse 2-6, A-1090 Vienna, Austria
bernd.simon@wu-wien.ac.at
[3]Universität Karlsruhe, Institut AIFB, Karlsruhe, Englerstr. 11, 76131 Karlsruhe, Germany

Abstract. Many RDF repositories have already been implemented with various access languages and mechanisms. The aim of the EDUTELLA framework is to allow communication between different RDF repository implementations. Part of EDUTELLA is a Query Exchange Language (QEL) which can be used as lingua franca to retrieve information from RDF repositories. This work shows why we also need standardization of distributed modification capabilities. We describe use case scenarios for annotation and replication services and use them as guideline for our approach towards a Modification Exchange Language (MEL) for distributed RDF repositories.

1 Introduction

In order to realize the Semantic Web, repositories storing metadata on information and services need to become interoperable [0]. While a lot of query mechanisms and languages currently do exist, the realization of the Semantic Web still requires a lingua franca allowing interactions between repositories for the purpose of managing metadata in a distributed manner.

The EDUTELLA framework aims to provide an RDF-based infrastructure which allows services to exchange metadata via a peer-to-peer network [0]. A peer-to-peer architecture goes beyond the boundaries of a classical client-server architecture. Each node can act as a provider or consumer of information and services. The network as a whole provides a discovery mechanism for finding relevant information and service providers. This approach increases the flexibility of system design and contributes to a more effective infrastructure for discovery, delivery and processing of information and service [0]. We envision a peer-to-peer infrastructure as the primary infrastructure for the Semantic Web, due to the increased heterogeneity of interoperable, high-level services we expect on the Semantic Web.

Currently peer-to-peer networks are based on proprietary protocols. In order to make heterogeneous peer-to-peer networks interoperable, gateways have to be designed, which are based on open protocols with a well-defined semantic [0]. EDUTELLA already offers the possibility to refine and optimize information search via protocols for querying metadata from RDF repositories. Hence, a first step

towards the interoperability of metadata repositories has been achieved by the definition of a Query Exchange Language. However, this is insufficient when it comes to annotation and replication within a network of distributed metadata repositories, where also a standardized mechanism to communicate metadata changes is needed.

In this paper we present a basic language designed for communicating metadata changes between distributed RDF repositories. The paper is organized as follows: In Section 2 use cases identifying our functional requirements are described. In Section 3 we discuss initial considerations, which are used as the basis for the design of our proposed language. This language, the Modification Exchange Language, is described in Section 4 as a possible means to standardize modification requests to RDF repositories. Section 5 addresses related work, and Section 6 presents concluding remarks.

2   Use Cases

To show why we need a standardized modification interface to RDF repositories we present two exemplifying use cases [0]. The first use case illustrates the need for replicating RDF repositories, a special instance of a general modification use case. The second use case describes the need for a Modification Exchange Language in the context of collaborative metadata authoring.

2.1 Integrated Systems for Teaching and Learning

In this use case two types of peers are involved: a learning management system (LMS), which supports instructors in the process of delivering learning, and a brokerage system (BS), which provides facilities for the exchange of learning resources.

A BS for learning resources supports instructors preparing their courses, by making educational content such as electronic textbooks, lecture notes, exercises, case studies, etc. stored at dispersed content repositories available at single virtual node. The idea behind brokerage systems is to support the re-use of learning resources and the collaborative development of it. Examples for brokerage systems for learning resources are: UNIVERSAL (http://www.ist-universal.org), Merlot (http://www.merlot.org), and GEM - The Gateway to Educational Materials (http://www.thegateway.org). Where as systems such as GEM and Merlot provide a loose integration of the various content sources via hyperlinks, UNIVERSAL aims at providing a tighter integration allowing the BS to grant and withdraw access rights at remote delivery systems based on learning resource metadata stored on a central node.

An LMS typically holds various learning resources in a repository. Instructors combine those resources to courses, which are then presented to their learners according to a course curriculum. Some learning management systems, for example Hyperwave's E-learning Suite, enables the sharing of individual learning resources among all instructors registered at a single system installation. Instructors can query the repository of a single system installation in order to search for an appropriate resource of one of their peers, which they would like to re-use in their own course.

However, up to now an open exchange of learning resources beyond the boundaries of single system installation is not available due to the lack of an appropriate infrastructure. One requirement for such an infrastructure would be the ability to replicate metadata describing learning resources of one LMS to a BS, so that it can be cached there and queried directly by all users of the BS. Metadata replication is a key element in such a usage scenario, which requires modification commands such as insert, update, and delete to be executed at remote copies of an RDF repository.

A survey [0] has shown that instructors have a clear preference towards opening already existing learning resource repositories selectively compared to redundantly uploading and managing their learning resources onto a central server. As a result, brokerage systems such as UNIVERSAL are aiming at making the metadata of dispersed learning resources available without requiring instructors to upload resources to a central server.

Integrating an LMS with a BS creates a peer-to-peer network, where the combination of both types of peers creates a new system with an added value. Whereas an LMS provides basic functionality for managing learning resources, a BS enhances this functionality by providing means for specifying usage conditions of the learning resources offered. In order to realize such a scenario of integrated services, brokerage systems are required to include the metadata describing learning resources stored at distributed LMS, so that it can provide customized offers of learning resources to remote instructors [0, 0].
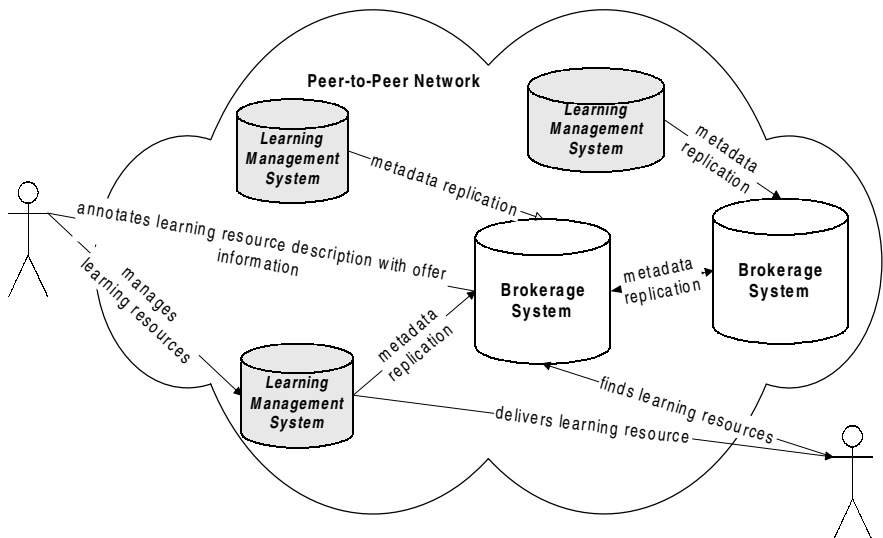
**Fig. 1.** Integrating LMS and BS by means of metadata replication

Figure 1 shows users interacting with an LMS and a BS using the LMS for providing general metadata on learning resources, for example title or educational objective, whereas the BS is used for specifying offer-related information such as price or usage restrictions. One can envision an extension scenario, where multiple brokerage

systems use replication in order to enhance their repositories with content descriptions of allied systems. In a similar way replicating metadata from a BS to an LMS would be required, when an LMS aims to provide facilities for querying and presenting metadata of remote resources via its own user interface.

2.2 Collaborative Annotation

One of the core components of the Semantic Web is to have metadata available in a machine and human-understandable format. As part of the KAON Framework [0], the Ontomat annotizer [0] has been developed to tackle this need. It provides facilities for annotation and annotation-enhanced authoring KAON uses the same format for metadata and ontology, namely RDF, whose advantages have been agreed on by a large community of users [0].

On the one hand, realizing high quality markup is perceived to be a crucial aspect in the context of the Semantic Web [0]. On the other hand, annotation is a time consuming effort. As a result a collaborative approach for sharing, both, existing metadata and the annotation work has been proposed [0], which contributes to a reduction in costs.
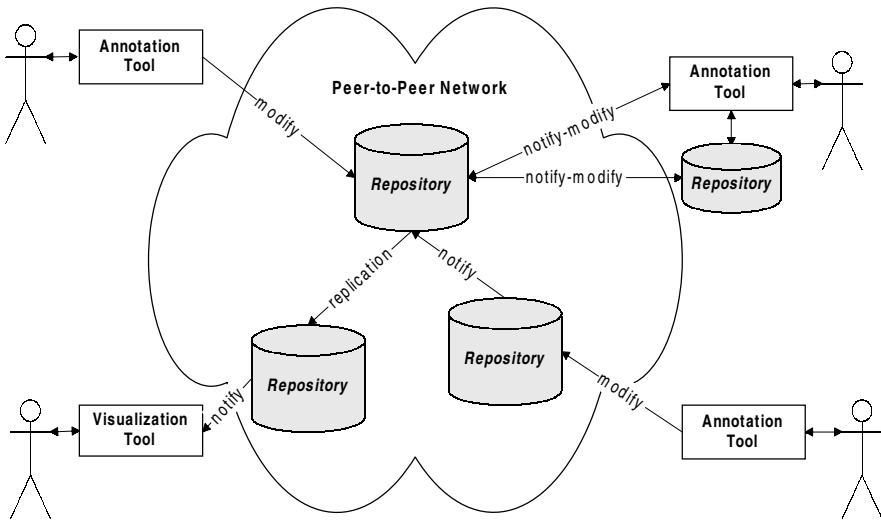
Fig. 2. Collaborative metadata authoring by means of notification and modification mechanisms

In order to preserve coherence between institutions, collaboration support for metadata authoring has to consider decentralization and a high level of heterogeneity. Indeed, each annotator may have different goals, use different tools and belong to diverse institutions. The important point is then to allow interaction between annotating applications and storage components without imposing the need for central control entities or a specific annotation framework.

Building on the query mechanism of the EDUTELLA project, the possibility to retrieve, both, metadata and ontologies has been added to the KAON framework in order to address collaborative aspects of metadata annotation. In addition, due to the distributed storage, performing collaborative work requires two important functionalities: notification and modification. First, other annotators might want to be notified of any recent change. Second, the metadata may be stored in a dispersed manner and accessed by annotators. This means that a modification protocol has to be designed to address the different needs imposed by collaborative annotation.

Collaborative metadata authoring tools can make great use of replication mechanisms. There are at least two reasons for this: First, performing queries for dispersed annotations may take too long. Hence, a caching mechanism relying on metadata replication can improve the overall system performance. Second, in a peer-to-peer network, peers are not expected to be constantly accessible. Replication would allow the annotator to have continuing access to previous states of annotation, which are updated as soon as the source repository becomes accessible again.

3   Design Considerations

Like the Query Exchange Language (QEL) the Modification Exchange Language (MEL) proposed in this paper is based on RDF. This has several advantages:

- In the spirit of the Semantic Web, messages are self-describing in a format suitable to be processed by all kinds of RDF tools;
- When stored persistently the messages build a journal of all modifications of a repository. As such a journal also consists of RDF statements, it can be queried using QEL queries;
- Existing approaches to describe statements and select RDF subgraphs can be used;
- By encoding the commands[1] in the message we avoid the need for a standardized repository API with operations for each command type.

The drawback of RDF-encoded messages is that the messages become quite bloated, as reification is needed with its circumstantial syntax.

3.1 Granularity Levels of Modification Commands

In an RDF-based environment several granularity levels of the minimum amount of metadata which can be addressed by a modification request can be distinguished: statement, resource with properties (either all properties or restricted by scope) or subgraphs. Each granularity has certain advantages and drawbacks:

- *Statement-centered*: Addressing RDF statements is the simplest solution. However, when updating statements (which will probably be the most frequent action compared to insert and delete) the lack of statement identifier in RDF causes

---

[1] In the context of RDBMS or other storage systems typically the terms 'insert/update/delete *statement*' are used. To avoid confusion with RDF statements, we use the term '*command*' instead.

difficulties. Essentially only insert and delete commands are available, and the complete triples have to be sent whenever a statement has to be deleted.

- **Resource-centered**: If change actions are grouped by resource, the set of all statements having the same resource as subject becomes the smallest modification unit. For inserts this leads to the same behavior as with the statement-centered approach. Delete operations can be performed by just submitting the URI of the resource. When updating a resource all statements regarding this resource have to be sent even when only some properties have changed. This is again unavoidable because of the missing statement identifier. Imagine a repository has stored (myCourse contributor A), (myCourse contributor B). Then an update statement arrives stating (myCourse contributor C). How should the resulting contributor set look like? AC, BC, ABC or C are possible choices, but to enable deletions the last choice is the only feasible. To avoid sending too much redundant information an update command could be scoped by a reference to a schema or schema element.
- **Subgraph-centered**: RDF query languages can deliver the query result as a subgraph of the repository. Therefore we can design modification commands as a combination of a query to specify the affected statements and a specification of the changes to these statements. For example, an update would consist of a query specifying the changed statement(s) and the description of the new statement(s). The repository can then change the selected statements accordingly.

For MEL we chose the subgraph-centered approach for the following reasons:

- This approach can handle variables in the modification specifications; while other approaches require explicit specification of all statements to be changed, this approach also supports change patterns.
- It enables replacing the object part of a statement without knowing its actual value.
- It integrates nicely with the existing query language QEL, which can be used to specify the subgraph selection.

In order to avoid the occurrence of inconsistent states of RDF repositories caused by remote modification commands, atomic modification commands have to be grouped into transactions. The handling of such transactions is often solved by sending the modification commands one by one, followed by a commit command. Such a design requires a stateful communication protocol, which is more complex and requires more resources than a stateless solution. We prefer the latter, and therefore allow modification messages to contain multiple commands which possibly form a logical unit. The repository can process such a message in one chunk, thereby avoiding the need to store a communication state.

3.2 Replication Design

Replication is a widely discussed topic in computer science and information systems research. Replication of data is required to increase the performance of a global information system [0] or enhance the reliability of a storage media [0]. Caching is a special form of data replication where the emphasize lies on improving the response time of systems for the most frequently accessed data [0]. It has been shown [0] that converting passive caches into replicated servers improves transmission times and results in a more evenly distributed bandwidth usage (because the replicas can be updated during low-traffic hours). In the context of the Semantic Web replication is

an important mechanism for establishing value chains of integrated peers. Metadata needs to be forwarded from peer A to peer B, because peer B may be capable of providing a special service (e.g. brokerage of learning resources) peer A (e.g. a learning management system) is not able to provide.

The following list summarizes design considerations of replication mechanisms:
- Primary design objective: increased availability and reliability;
- Traceability of data providers: traceable vs. anonymous;
- Communication mode: synchronous vs. asynchronous, also called eager replication vs. lazy replication [0];
- Degree of initial modification distribution: update everywhere vs. primary copy, also called active vs. passive replication;
- Degree of consistency: strong consistency vs. weak consistency [0].

Our primary design objective of the replication protocol is to increase the availability of data in order to create value chains of integrated peers.

Currently we assume that the primary copies know their replicas and vice versa. Providers will be traceable by system; creating a (semi-) anonymous replication protocol is not a design goal here. Other, more complex, approaches would be:
- The primary copy publishes its changes to replication hubs which distribute them to the replicas.
- Replicas fetch changes from their primary copies on a scheduled basis.

Synchronous replication requires locking since an update transaction must update copies before it commits [0]. In a peer-to-peer environment synchronous replication is not feasible, because of temporary (un)availability of peers. This also supports the primary copy approach, where the metadata is updated at the repository holding the primary copy first, and is then distributed to the replicas. To avoid complex reconciliation procedures, modification commands must be sent to the primary copy first.

An RDF repository holding replicated metadata from more than one location will have to preserve the originating context with the metadata for the following reasons:
- Statements from different origins may be contradictory. Merging such statements into one statement would invalidate the complete repository content. When the context is preserved, one way to handle such cases would be to return separate results for each context. A more sophisticated solution could assign trust levels to replicated repositories and filter statements from less trusted repositories when contradictions occur.
- When merging repositories without considering their origin, delete and update actions may lead to undesired effects. One can imagine the following scenario: Professor X changes from university A to university B. Both universities are providing meta data about their staff, which are replicated by peer C. As X is now member of the staff of B, B inserts (among others) the statement (X teaches Economics) into its repository. This statement is replicated to C. Some time later, A deletes all statements about X from its repository, among them the statement (X teaches Economics). This must not result in the deletion of this statement at C because the statement is also asserted by B. C can handle this case correctly only if it stores the origins of all statements.

It is also advisable that the replicas know were the primary copy is stored when tight consistency is needed [0]. For example, a user at the BS intends to book a resource,

the BS has to check back whether this resource is still available and provide the latest offer terms. In this case referring back to the primary copy is advisable.

3.3 Annotation Design

As described in the use case above, collaborative metadata authoring requires to be supported by an distributed environment and without a central control entity. Peer-to-peer networks address this need.

The heterogeneity problem of the annotation applications and storages can then be addressed by defining a set of application independent protocols for the exchange of metadata. However, we saw that exchanging metadata is not sufficient for a collaborative annotation scenario. A modification protocol should also be designed in order to allow:

*Change notification***:** Annotators need to be informed of changes which could influence their annotation work. Basically, they need to know what has been inserted, updated or deleted. Moreover, the notifications should be as comprehensive and expressive as possible. Therefore, using the *subgraph-centered* approach should help to make modifications more easily visualizable.

*Change request***:** Different annotators using a set of different repositories need a neutral way to request changes in the metadata that they store. If all use and support the same modification protocol, the actual task can be left to the implementation of the repository.

Some modifications might not require that you have specific information about which object you want to modify. For example, the set of all pages written by a given author might be marked as "regularly-updated".

4 The Modification Exchange Language (MEL)

4.1 Introduction

MEL is based on QEL, which is an RDF representation for Datalog queries. Datalog is a non-procedural query language based on Horn clauses. In this language a query consists of literals (predicates expressions describing relations between variables and constants) and a set of rules [0].

As in SQL we provide the commands *insert*, *delete* and *update*. All commands use a statement specification to describe the affected statements.
We describe the syntax in an informal notation similar to EBNF (Extended Backus-Naur Form)[2].

    statementSpec = subjectSpec propertySpec objectSpec
    subjectSpec = *subject* resourceSpec
    propertySpec = *property* resourceSpec
    objectSpec = *object* (resourceSpec | literalSpec)
    resourceSpec = URI

---

[2] EBNF is not well suited for specifying RDF messages formally, because no order of the statements can and should be prescribed, but it allows a concise description.

    literalSpec = STRING

A special type of resourceSpec is a variable, which must be declared in the command:
    variableDeclaration = *hasVariable* resourceSpec

4.2 Format of Modification Commands

The *Delete* command consists of a statement specification and optionally a query
constraint. All statements in the repository matching the specification are deleted. A
constraintSpec can be any QEL query.
    deleteCommand = *Delete* statementSpec {variableDeclaration} [constraintSpec]

The following example deletes all statements with property *dc:comment* and a subject
of *rdf:type …#Book* from the repository:

```
<edu:Delete rdf:about="#delete_cmd">
  <edu:oldStatement rdf:resource="#del_stmt"/>
  <edu:hasVariable rdf:resource="#U"/>
  <edu:hasVariable rdf:resource="#V"/>
  <edu:hasConstraint rdf:resource="#del_constraint"/>
</edu:Delete>

<edu:DeletedStatement rdf:about="#del_stmt">
  <rdf:subject rdf:resource="#U"/>
  <rdf:predicate rdf:resource="&dcq;comment"/>
  <rdf:object rdf:resource="#V"/>
</edu:QueryStatement>

<!-- QEL-1 query -->
<edu:Query rdf:about="#del_constraint">
  <edu:hasVariable rdf:resource="#U"/>
</edu:Query>

<edu:Variable rdf:about="#U">
  <rdf:type rdf:resource="http://www.lit.edu/types#Book"/>
</edu:Variable>
```

The *Insert* command syntax is similar to the delete syntax. Here the statement
specification describes the new statements.
    insertCommand = *Insert* statementSpec {variableDeclaration} [constraintSpec]

The simplest case is an insert of one RDF statement:

```
<edu:Insert rdf:about="#insert_cmd1">
  <edu:newStatement rdf:resource="#insert1_stmt"/>
</edu:Insert>

<edu:InsertedStatement rdf:about="#insert1_stmt">
  <rdf:subject
      rdf:resource="http://www.mylib.org/books#Book37"/>
  <rdf:predicate rdf:resource="&dc;title"/>
  <rdf:object rdf:resource="The Magic of RDF"/>
</edu:QueryStatement>
```

It is also possible to insert more than one statement with a single command. Suppose you want to add a book collection to a library. The following command inserts a new property `lendingState` for all resources of type `Book`, preparing all books for library business with one statement:

```
<edu:Insert rdf:about="insert_cmd2">
  <edu:newStatement rdf:resource="#insert2_stmt"/>
  <edu:hasConstraint rdf:resource="#insert2_constraint"/>
  <edu:hasVariable rdf:resource="#W"/>
</edu:Insert>

<edu:InsertedStatement rdf:about="#insert2_stmt">
  <rdf:subject rdf:resource="#W"/>
  <rdf:predicate rdf:resource="&lib;lendingState"/>
  <rdf:object rdf:resource="&lib;available"/>
</edu:QueryStatement>

<edu:Query rdf:about="#insert2_constraint">
  <edu:hasVariable rdf:resource="#W"/>
</edu:Query>

<edu:Variable rdf:about="#W">
  <rdf:type rdf:resource="http://www.lit.edu/types#Book"/>
</edu:Variable>
```

Update commands require two statement specifications, one for the replaced statements and one for the replacing statements:

  updateCommand = *Update* 2*statementSpec {variableDeclaration} [constraintSpec]

The following example updates the modification date of the resource with the title 'Sample':

```
<edu:Update rdf:about="#update_cmd">
  <edu:newStatement rdf:resource="#new_stmt"/>
  <edu:oldStatement rdf:resource="#old_stmt"/>
  <edu:hasConstraint rdf:resource="#update_constraint"/>
  <edu:hasVariable rdf:resource="#X"/>
  <edu:hasVariable rdf:resource="#Y"/>
</edu:Update>

<edu:OriginalStatement rdf:about="#old_stmt">
  <rdf:subject rdf:resource="#X"/>
  <rdf:predicate rdf:resource="&dcq;modified"/>
  <rdf:object rdf:resource="#Y"/>
</edu:QueryStatement>

<edu:InsertedStatement rdf:about="#new_stmt">
  <rdf:subject rdf:resource="#X"/>
  <rdf:predicate rdf:resource="&dcq;modified"/>
  <rdf:object>
    <dcq:W3CDTF>
      <rdf:value>2002-02-03T:15:34:16+01:00</rdf:value>
    </dcq:W3CDTF>
  </rdf:object>
</edu:QueryStatement>
```

```
<edu:Query rdf:about="#update_constraint">
  <edu:hasVariable rdf:resource="#X"/>
</edu:Query>
<edu:Variable rdf:about="#X">
  <dc:title>Sample</dc:title>
</edu:Variable>
```

4.3 Format of Modification Messages

Each modification message is identified by a unique message identifier, which ensures the correct ordering of messages. This identifier is formed of at least two components (time, identifier of the modification originator) and an optional third one (request count). The originator identifier is a Universal Unique Identifier (UUID). A mechanism has to guarantee that UUIDs are unique, for example by combining hardware addresses, and random seeds. Time is coded using W3C's version of the date and time format (http://www.w3.org/TR/NOTE-datetime) with complete date plus hours, minutes, seconds and time zone designator. If multiple modification messages are created within a second, a request count can be used to uniquely identify the request.

messageID = originator timestamp [number]
originator = *messageOriginator* UUID
timestamp = *messageTimestamp* W3CDTF
number = *messageNumber* DIGIT

A modification message can hold multiple synchronization commands, which can be either an insert, delete or update command. All commands (and other necessary resources) are identified by a unique local fragment. The commands are contained in a sequence to preserve the order.

Additional message information can be added, for example when the message was created and modified for the last time, i.e. closed and prepared for sending it to the replicating peer. The name of the peer placing the request can also be attached.

message = messageID messageInformation commandList
messageInformation= {originator} {creationTime} {modificationTime}
commandList = {command}
command = insertCommand | updateCommand | deleteCommand
creationTime = W3CDTF
modificationTime = W3CDTF

An example is presented below:

```
<edu:ModificationMessage rdf:about="#msg1">
  <edu:messageOriginator>
    urn:jxta:uuid-BEFAF79B91504F2FA39FAEFE9C7A4602
  </edu:messageOriginator>
  <edu:messageTimestamp>
    <dcq:W3CDTF>
      <rdf:value>2002-02-03T15:34:42+01:00</rdf:value>
    </dcq:W3CDTF>
```

```
    </edu:messageTimestamp>
    <edu:hasCommands>
      <rdf:Seq>
        <rdf:_1 rdf:resource="#cmd1"/>
        <rdf:_2 rdf:resource="#cmd2"/>
        <rdf:_3 rdf:resource="#cmd3"/>
      </rdf:Seq>
    </edu:hasCommands>
  </edu:ModificationMessage>
```

The receiving peer responds to the modification message by sending a response message which contains information about the update success.

5 Related Work

Several Web initiatives are currently extended with replication or modification protocols.

The Replication Architecture of the Lightweight Directory Access Protocol (LDAP) distinguishes between different replica types [0]. Each replica type has a certain set of operations assigned, which it is allowed to carry out. For example, the primary replica provides a full copy of the replica, to which all applications that require tight consistency direct their operations. On the contrary fractional replica accept only read-only LDAP operations. Introducing a hierarchy of replica peer types is worthwhile to consider in future versions of MEL.

The Universal Description, Discovery and Integration (UDDI) architecture [0] specifies the data replication process and interface required to achieve data replication between UDDI operators. The replication process makes use of XML. UDDI relies on SOAP, which provides the mechanism for using XML in simple message-based exchanges. UDDI operators sent controlled XML messages in order to communicate change records requests. The underlying message architecture is rather simple, as for example compared to LDAP, and does not support any semantically rich, self-containing messages.

The rdfDB Query Language [0] is a high level query language with a query syntax similar to SQLas far as selects are concerned. rdfDB provides modification commands according to the *statement-centered* approach: insert and delete commands which take lists of statements as an argument are available. Variables cannot be used within these commands.

Several other query languages are derived from rdfDB, e.g. RDQL [0] which is part of the Jena framework [0] and Inkling [0]. Interestingly, all of them have abandoned insert and delete and provide query capabilities only.

ANNOTEA is a client/server system for the creation of annotations [0]. All commands are sent to the server via HTTP. Commands to insert, update and delete annotations are provided, and a separate query language (Algae) is available. All messages are represented in RDF, enclosed in a HTTP PUT request. ANNOTEA uses the *resource-centered* approach. For insert as well as for update, the client sends all statements describing one resource in one chunk to the server. Update deletes all existing statements regarding the resource before inserting the sent statements. A

delete message contains just the resource URI; the server deletes all statements where this resource is subject.

RQL [0] is a highly developed RDF query language used in RDFSuite [0] and Sesame [0]. It provides no modification commands, because in these systems repository modification is done through a special API.

TRIPLE [0] is an RDF query and transformation language based on frame logic, also without modification support.

6   Concluding Remarks

In this paper, we have discussed replication and annotation in a peer-to-peer network and extended QEL, the query language specified for EDUTELLA, with additional modification capabilities. We believe that standardizing a modification exchange language, such as the one proposed in this paper, will contribute to the evolution of the Semantic Web idea. Our work is a first step in this direction; we have not yet treated all necessary aspects for these services. For example, the question of how to authorize modification commands is an issue, which still has to be addressed. In addition a full validation of the use cases sketched still has to be carried out.

References

1. J. Helfin and J. Hendler. A Portrait of the Semantic Web in Action. IEEE Intelligent Systems, 16 (2), 54-59, 2001.
2. Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér and Tore Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. Accepted for WWW2002, 2002.
3. L. Gong. Project JXTA: A Technology Overview. Sun Microsystems, Palo Alto, 2001.
4. B. Wiley. Interoperability Through Gateways. In: A. Oram (ed.), Peer-to-Peer - Harnessing the Power of Disruptive Technologies. O'Reilly, 2001.
5. I. Jacobsen and M. Christensen. Object-Oriented Software Engineering: A Use-Case Driven Approach. Addison-Wesley, Reading, 1992.
6. B. Simon. Faculty Members Meeting Electronic Education Markets - Determinants for Project Success. Working Paper, Department of Information Systems, Wirtschaftsuniversität Wien, Vienna, 2001.
7. S. Guth, G. Neumann and B. Simon. UNIVERSAL - Design Spaces for Learning Media. In: R. H. Sprague (ed.). Proceedings of the 34th Hawaii International Conference on System Sciences, 2001.
8. S. Brantner, T. Enzi, S. Guth, G. Neumann and B. Simon. UNIVERSAL - Design and Implementation of a Highly Flexible E-Market Place of Learning Resources. In: R. Hartley, Kinshuk, T. Okamoto and J. P. Klus (ed.). Proceedings of the IEEE International Conference on Advanced Learning Technologies, 2001.
9. The Karlsruhe Ontology and Semantic Web Tool Suite. http://kaon.aifb.uni-karlsruhe.de, 2001.
10. S. Handschuh, S. Staab and A. Mädche: CREAM - Creating relational metadata with a component-based, ontology-driven annotation framework. In: ACM K-CAP 2001. October, Vancouver, 2001.
11. S. Handschuh and S. Staab. Authoring and Annotation of Web Pages in CREAM. Accepted for WWW2002, 2002.

12. J. Hendler. Agents and the Semantic Web. IEEE Intelligent Systems, 16 (2), 30-37, 2001.
13. L. Qiu, V. N. Padmanabham, and G. M. Voelker. On the placement of web server replicas. In Proc. 20th IEEE INFOCOM, 2001.
14. B. Liskov, S. Ghemawat, R. Gruber, P. Johns, L. Shrira and M. Williams. Replication in the Harp file system. In: Proceedings of the 13th ACM Symposium on Operating Systems Principles, 1991.
15. R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In Proceedings of the Twentieth International Conference on Distributed Computing Systems, 1999.
16. M. Baentsch, L. Baum, G. Molter, S. Rothkugel and P. Sturm. Enhancing the Web's Infrastructure – From Caching to Replication. IEEE Internet Computing, 1(2):18--27, Mar. 1997.
17. J. Gray, P. Helland, P. O'Neil and D. Shasha. The dangers of replication and a solution. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 1996.
18. M. Wiesmann, F. Pedone, A. Schiper, B. Kemme and G. Alonso. Understanding replication in databases and distributed systems. In: Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'2000), Taipei, Taiwan, R.O.C. IEEE Computer Society Los Alamitos California, 2000.
19. B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In: Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS), Amsterdam, The Netherlands, 1998.
20. J. Merrells, E, Reed, U. Srinivasan. LDAP Replication Architecture. IETF Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-ldup-model-06.txt, 2000.
21. R. Atkinson and J. Munter (eds.).UDDI Version 2.0 Replication Specification. uddi.org (2001). Available at http://www.uddi.org/pubs/Replication-V2.00-Open-20010608.pdf
22. R. V. Guha. RDFDB QL. http://web1.guha.com/rdfdb/query.html.
23. Andy Seaborne. RDQL - RDF Data Query Language. http://hpl.hp.com/semweb/rdql.html, 2001.
24. Brian McBride. Jena: Implementing the RDF Model and Syntax Specification. http://www-uk.hpl.hp.com/people/bwm/papers/20001221-paper/, 2000.
25. L. Miller. Inkling: RDF query using SquishQL. http://swordfish.rdfweb.org/rdfquery/ 2001.
26. J. Kahan, M. Koivunen, E. Prud'Hommeaux and R. Swick. Annotea: An Open RDF Infrastructure for Shared Web Annotations. In Proc. of the WWW10 International Conference. Hong Kong, 2001.
27. G. Karvounarakis, V. Christophides, D. Plexousakis and S. Alexaki. Querying CommunityWeb Portals. In: Proc. 17^{ièmes} Journees Bases de Donnees Avancees (BDA'01), Agadir, Maroc, 2001.
28. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis and K. Tolle. The RDFSuite: Managing Voluminous RDF Description Bases. In: Proc. of the 2nd Int. Workshop on the Semantic Web, Hong-Kong, 2001.
29. J. Broekstra, A. Kampman and F. van Harmelen. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In: D. Fensel, J. Hendler, H. Lieberman and W. Wahlster (eds.). Semantics for the WWW. MIT Press, 2001.
30. Michael Sintek and Stefan Decker. TRIPLE – An RDF Query, Inference, and Transformation Language. Deductive Databases and Knowledge Management Workshop (DDLP'2001), Japan, 2001.