EDUTELLA:
A P2P Networking Infrastructure Based on RDF

Wolfgang Nejdl, Boris Wolf, Changtao Qu[*], Stefan Decker[†], Michael Sintek[‡]
Ambjörn Naeve, Mikael Nilsson, Matthias Palmér[§], Tore Risch[¶]

ABSTRACT

Metadata for the World Wide Web is important, but metadata for Peer-to-Peer (P2P) networks is absolutely crucial. In this paper we discuss the open source project Edutella which builds upon metadata standards defined for the WWW and aims to provide an RDF-based metadata infrastructure for P2P applications, building on the recently announced JXTA Framework. We describe the goals and main services this infrastructure will provide and the architecture to connect Edutella Peers based on exchange of RDF metadata. As the query service is one of the core services of Edutella, upon which other services are built, we specify in detail the Edutella Common Data Model (ECDM) as basis for the Edutella query exchange language (RDF-QEL-i) and format implementing distributed queries over the Edutella network. Finally, we shortly discuss registration and mediation services, and introduce the prototype and application scenario for our current Edutella aware peers.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query Languages*; C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed Applications, Distributed Databases*

General Terms

Languages

Keywords

Semantic Web, Peer-to-Peer, Query Languages, E-Learning

1. INTRODUCTION

While in the server/client-based environment of the World Wide Web metadata are useful and important, for *Peer-to-Peer (P2P)* environments metadata are absolutely crucial. Information Resources in P2P networks are no longer organized in hypertext like structures, which can be navigated, but are stored on numerous peers waiting to be queried for these resources if we know what we want to retrieve and which peer is able to provide that information. Querying peers requires metadata describing the resources managed by these peers, which is easy to provide for specialized cases, but non-trivial for general applications.

P2P applications have been successful for special cases like exchanging music files. However, retrieving "all recent songs by Madonna" does not need complex query languages nor complex metadata, so special purpose formats for these P2P applications have been sufficient. In other scenarios, like exchanging educational resources, queries are more complex, and have to build upon standards like IEEE-LOM/IMS [5, 13] metadata with up to 100 metadata entries, which might even be complemented by domain specific extensions.

Furthermore, by concentrating on domain specific formats, current P2P implementations appear to be fragmenting into niche markets instead of developing unifying mechanisms for future P2P applications. There is indeed a great danger (as already discussed in [8]), that unifying interfaces and protocols introduced by the World Wide Web get lost in the forthcoming P2P arena.

The Edutella project [9] addresses these shortcomings of current P2P applications by building on the W3C metadata standard RDF [22, 2]. The project is a multi-staged effort to scope, specify, architect and implement an RDF-based metadata infrastructure for P2P-networks based on the recently announced JXTA framework [10]. The initial Edutella services will be *Query Service* (standardized query and retrieval of RDF metadata), *Replication Service* (providing data persistence / availability and workload balancing while maintaining data integrity and consistency), *Mapping Service* (translating between different metadata vocabularies to enable interoperability between different peers), *Mediation Service* (define views that join data from different meta-data sources and reconcile conflicting and overlapping information) and *Annotation Service* (annotate materials stored anywhere within the Edutella Network).

Our vision is to provide the metadata services needed to enable interoperability between heterogeneous JXTA applications. Our first application will focus on a P2P network for the exchange of educational resources (using schemas like IEEE LOM, IMS, and ADL SCORM [37] to describe course materials), other application areas will follow.

In Sections 2 and 3 we describe the background and framework of the Edutella architecture and our educational application scenario. Then, as the query service is one of the core services of Edutella, upon which other services are built, we specify in detail in Section 4 the Edutella common data model (ECDM) as basis for the Edutella query exchange language and format imple-

[*]Learning Lab Lower Saxony, University of Hannover, 30060 Hannover, Germany, {nejdl,wolf,qu}@learninglab.de

[†]Database Group, Stanford University, USA, stefan@db.stanford.edu

[‡]DFKI GmbH, Kaiserslautern, Germany, sintek@dfki.de

[§]Centre for user oriented IT Design, Royal Institute of Technology, Stockholm, Sweden, {amb,mini,matthias}@nada.kth.se

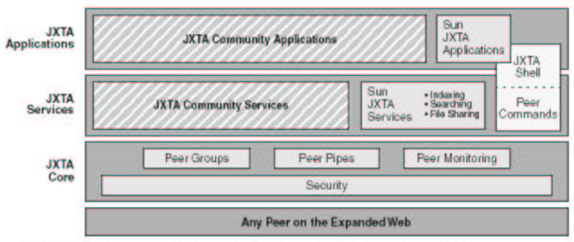
[¶]Department of Information Science, Uppsala University, Sweden, tore.risch@dis.uu.se

menting distributed queries over the Edutella network. Finally, we sketch translations from the Edutella CDM to different query languages (Section 5), shortly discuss registration and mediation services (Section 6), and introduce the prototype and application scenario for our current Edutella aware peers (Section 7).

2. BACKGROUND

2.1 The JXTA P2P Framework

JXTA is an Open Source project [18, 10] supported and managed by Sun Microsystems. In essence, JXTA is a set of XML based protocols [36] to cover typical P2P functionality. It provides a Java binding offering a layered approach for creating P2P applications (core, services, applications, see Figure 1, reproduced from [10]). In addition to remote service access (such as offered by SOAP), JXTA provides additional P2P protocols and services, including peer discovery, peer groups, peer pipes, and peer monitors. Therefore JXTA is a very useful framework for prototyping and developing P2P applications.



JXTA software is layered with core functionality developed through an open, collaborative effort, and higher-level services developed by P2P community developers.

Figure 1: JXTA Layers

This layered approach fits very nicely into our application scenarios defined for Edutella:

Edutella Services (described in web service languages like DAML-S or WSDL, etc.) complement the Jxta Service Layer, building upon the JXTA Core Layer, and

Edutella Peers live on the Application Layer, using the functionality provided by these Edutella services as well as possibly other JXTA services.

On the Edutella Service layer, we define data exchange formats and protocols (how to exchange queries, query results and other metadata between Edutella Peers), as well as APIs for advanced functionality in a library-like manner. Applications like repositories, annotation tools or GUI interfaces connected to and accessing the Edutella network are implemented on the application layer.

2.2 Educational Context

Every single university usually has already a large pool of educational resources distributed over its institutions. These are under control of the single entities or individuals, and it is unlikely that these entities will give up their control, which explains why all approaches for the distribution of educational media based on central repositories have failed so far. Furthermore, setting up and maintaining central servers is costly. The costs are hardly justifiable, since a server distributing educational material would not directly benefit the sponsoring university. We believe, that in order to really facilitate the exchange of educational media, approaches based on metadata-enhanced peer-to-peer (P2P) networks are necessary.

In a typical P2P-based e-learning scenario, each university acts not only as content provider but also as content consumer, including local annotation of resources produced at other sites. As content provider in a P2P network they will not lose their control over their learning resources but still provide them for use within the network. As a content consumer both teachers and students benefit from having access not only to a local repository, but to a whole network, using queries over the metadata distributed within the network to retrieve required resources.

P2P networks have already been quite successful for exchanging data in heterogeneous environments, and have been brought into focus with services like Napster and Gnutella, providing access to distributed resources like MP3 coded audio data. However, pure Napster and Gnutella like approaches are not suitable for the exchange of educational media. For example, the metadata in Gnutella is limited to a file name and a path. While this might work for files with titles like “Madonna - Like a Virgin”, it certainly does not work for “Introduction to Algebra - Lecture 23”. Furthermore, these special purpose services lead to fragmented communities which use special purpose clients to access their service.

The educational domain is in need of a much richer metadata markup of resources, a markup that is often highly domain and resource type specific. In order to facilitate interoperability and reusability of educational resources, we need to build a system supporting a wide range of such resources. This places high demands on the interchange protocols and metadata schemata used in such a system, as well as on the overall technical structure. Also, we do not want to create yet another special purpose solution which is outdated as soon as metadata requirements and definitions change.

Our metadata based peer to peer system has therefore to be able to integrate heterogeneous peers (using different repositories, query languages and functionalities) as well as different kinds of metadata schemas. We find common ground in the essential assumption that all resources maintained in the Edutella network can be described in RDF, and all functionality in the Edutella network is mediated through RDF statements and queries on them. For the local user, the Edutella network transparently provides access to distributed information resources, and different clients/peers can be used to access these resources. Each peer will be required to offer a number of basic services and may offer additional advanced services.

3. EDUTELLA SERVICES

Edutella connects highly heterogeneous peers (heterogeneous in their uptime, performance, storage size, functionality, number of users etc.). However, each Edutella peer can make its metadata information available as a set of RDF statements. Our goal is to make the distributed nature of the individual RDF peers connected to the Edutella network completely transparent by specifying and implementing a set of Edutella services. Each peer will be characterized by the set of services it offers.

Query Service. The Edutella query service is the most basic service within the Edutella network and is described in more detail in the second part of this paper. Peers register the queries they may be asked through the query service (i.e., by specifying supported metadata schemas (e.g., “this peer provides metadata according to the LOM 6.1 or DCMI standards”) or by specifying individual properties or even values for these properties (e.g., “this peer provides metadata of the form dc_title(X,Y)” or “this peer provides metadata of the form dc_title(X,’Artificial Intelligence’)”). Queries are sent through the Edutella network to the subset of peers who have registered with the service to be interested in this kind of query. The resulting RDF statements are sent back to the requesting peer.

Edutella Replication. This service is complementing local storage by replicating data in additional peers to achieve data persistence / availability and workload balancing while maintaining data integrity and consistency. Since Edutella is mainly concerned with metadata, replication of metadata is our initial focus. Replication of data might be an additional possibility (though this complicates synchronization of updates).

Edutella Mapping, Mediation, Clustering. While groups of peers will usually agree on using a common schema (e.g., SCORM or IMS/LOM for educational resources), extensions or variations might be needed in some locations. The Edutella Mapping service will be able to manage mappings between different schemata and use these mappings to translate queries over one schema X to queries over another schema Y. Mapping services will also provide interoperation between RDF- and XML-based repositories. Mediation services actively mediate access between different services, clustering services use semantic information to set up semantic routing and semantic clusters.

4. EDUTELLA QUERY SERVICE

The Edutella Query Service is intended to be a standardized query exchange mechanism for RDF metadata stored in distributed RDF repositories and is meant to serve as both query interface for individual RDF repositories located at single Edutella peers as well as query interface for distributed queries spanning multiple RDF repositories. An RDF repository (or knowledge base) consists of RDF statements (or facts) and describes metadata according to arbitrary RDFS schemas.

One of the main purposes is to abstract from various possible RDF storage layer query languages (e.g., SQL) and from different user level query languages (e.g., RQL, TRIPLE): The Edutella Query Exchange Language and the Edutella common data model provide the syntax and semantics for an overall standard query interface across heterogeneous peer repositories for any kind of RDF metadata. The Edutella network uses the query exchange language family RDF-QEL-i (based on Datalog semantics and subsets thereof) as standardized query exchange language format which is transmitted in an RDF/XML-format.

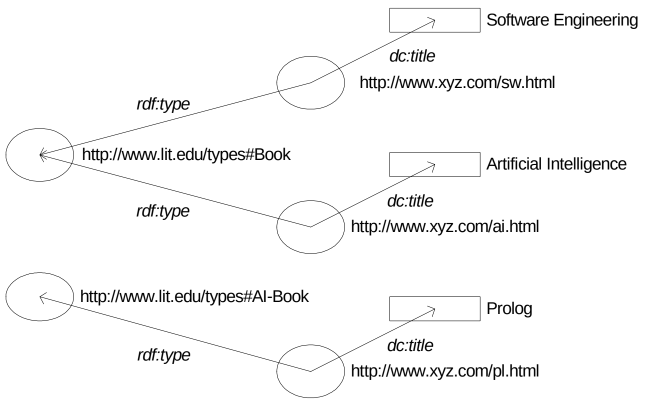

Software Engineering
dc:title
http://www.xyz.com/sw.html
rdf:type
http://www.lit.edu/types#Book
Artificial Intelligence
dc:title
rdf:type
http://www.xyz.com/ai.html
http://www.lit.edu/types#AI-Book
Prolog
dc:title
rdf:type
http://www.xyz.com/pl.html

Figure 2: Knowledge Base as RDF Graph

We will start with a simple RDF knowledge base and a simple query on this knowledge base depicted in Figure 2, with the following RDF XML Serialization:

```
<lib:Book about="http://www.xyz.com/sw.html">
  <dc:title>Software Engineering</dc:title>
</lib:Book>

<lib:Book about="http://www.xyz.com/ai.html">
  <dc:title>Artificial Intelligence</dc:title>
</lib:Book>
```

```
<lib:AI-Book about="http://www.xyz.com/pl.html">
  <dc:title>Prolog</dc:title>
</lib:AI-Book>
```

Evaluating the following query (plain English)

> “Return all resources that are a book having the title ’Artificial Intelligence’ or that are an AI book.”

we get the query results shown in Figure 3, depicted as RDF-graph.

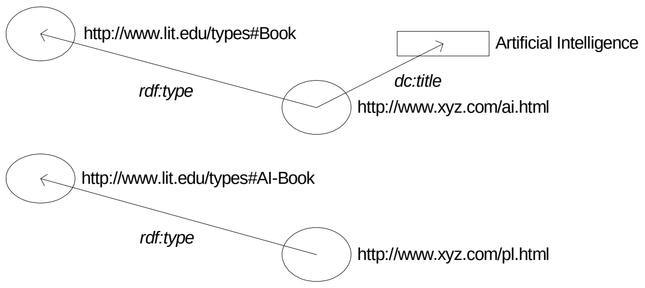

http://www.lit.edu/types#Book
Artificial Intelligence
dc:title
rdf:type
http://www.xyz.com/ai.html
http://www.lit.edu/types#AI-Book
rdf:type
http://www.xyz.com/pl.html

Figure 3: Query Results as RDF Graph

4.1 Query Exchange Architecture

Edutella peers are highly heterogeneous in terms of the functionality (i.e., services) they offer. A simple peer has RDF storage capability only. The peer has some kind of local storage for RDF triples (e.g., a relational database) as well as some kind of local query language (e.g., SQL). In addition the peer might offer more complex services such as annotation, mediation or mapping.

To enable the peer to participate in the Edutella network, Edutella wrappers are used to translate queries and results from the Edutella query and result exchange format to the local format of the peer and vice versa, and to connect the peer to the Edutella network by a JXTA-based P2P library. To handle queries the wrapper uses the common Edutella query exchange format and data model for query and result representation. For communication with the Edutella network the wrapper translates the local data model into the Edutella common data model ECDM described in this paper and vice versa, and connects to the Edutella Network using the JXTA P2P primitives, transmitting the queries based on the common data model ECDM in RDF/XML form.

In order to handle different query capabilities, we define several RDF-QEL-i exchange language levels, describing which kind of queries a peer can handle (conjunctive queries, relational algebra, transitive closure, etc.) The same internal data model is used for all levels.

4.2 Datalog Semantics for the Edutella Common Data Model (ECDM)

Datalog is a non-procedural query language based on Horn clauses without function symbols. A Horn clause is a disjunction of literals where there is at most one positive (non-negated) literal. A Datalog program can be expressed as a set of rules/implications (where each rule consists of one positive literal in the consequent of the rule (the head), and one or more negative literals in the antecedent of the rule (the body)), a set of facts (single positive literals) and the actual query literals (a rule without head, i.e., one or more negative literals). Additionally, we can use negation as failure in the antecedent of a rule, with the semantics that such a literal cannot be proven from the knowledge base (see, e.g., [34]).

Literals are predicates expressions describing relations between any combination of variables and constants such as title(http://www.xyz.com/book.html, ’Artificial Intelligence’). Each rule is divided into head and body with the head being a single literal and the body being a conjunction of any number of positive literals (including conditions on variables). Disjunction is expressed as a set of rules with identical head. A Datalog query then is a conjunction of query literals plus a possibly empty set of rules.

Datalog shares with relational databases and with RDF the central feature, that data are conceptually grouped around properties (in contrast to object oriented systems, which group information within objects usually having object identity).[1] Therefore Datalog queries easily map to relations and relational query languages like relational algebra or SQL. In terms of relational algebra Datalog is capable of expressing selection, union, join and projection and hence is a relationally complete query language. Additional features include transitive closure and other recursive definitions.

The example knowledge base in Datalog reads

```
title(http://www.xyz.com/ai.html,’Artificial
                                   Intelligence’).
type(http://www.xyz.com/ai.html,Book).
title(http://www.xyz.com/sw.html,’Software
                                   Engineering’).
type(http://www.xyz.com/sw.html,Book).
title(http://www.xyz.com/pl.html,’Prolog’).
type(http://www.xyz.com/pl.html,AI-Book).
```

In RDF any statement is considered to be an assertion. Therefore we can view an RDF repository as a set of ground assertions either using binary predicates as shown above, or as ternary statements “s(S,P,O)”, if we include the predicate as an additional argument. In the following examples, we use the binary surface representation, whenever our query does not span more than one abstraction level[2].

Example Query in (binary) Datalog notation.

```
aibook(X) :- title(X, ’Artificial Intelligence’),
             type(X, Book).
aibook(X) :- type(X, AI-Book).
?- aibook(X).
```

Since our query is a disjunction of two (purely conjunctive) subqueries, its Datalog representation is composed of two rules with identical heads. The literals in the rules’ bodies directly reflect RDF statements with their subjects being the variable X and their objects being bound to constant values such as ’Artificial Intelligence’. Literals used in the head of rules denote derived predicates (not necessarily binary ones). In our example, the query expression “aibook(X)” asks for all bindings of X, which conform to the given Datalog rules and our knowledgebase, with the results:

```
aibook(http://www.xyz.com/ai.html)
aibook(http://www.xyz.com/pl.html)
```

4.3 Edutella Common Data and Query Exchange Model

Internally Edutella Peers use a Datalog based model to represent queries and their results. Figure 4 visualizes this data model as UML class diagram. All classes beginning with *RDF* are standard RDF concepts and reflect their usage in the Stanford RDF API [28].

Each query is represented as an instance of `EduQuery` which aggregates an arbitrary number of `EduRule` and `EduLiteral` objects. `EduLiterals` are either `RDFReifiedStatements` (binary predicates / ternary statement literals, corresponding to reified RDF statements), `EduStatementLiterals` (non-ternary statement literals, that cannot be expressed as ordinary RDF statements) or `EduConditionLiterals` (a condition expression on variables such as $X > 5$). In our examples we use different surface notations of this data model, and switch to a predicate as argument view, whenever our query spans more than one abstraction level (see Section 4.7). Technically, it is sufficient to define a single instance of `EduLiteral` as query literal. However, by using a set of `EduLiteral` objects, all query literals together can be interpreted as the RDF result graph of the `EduQuery`, as long as the query literals are all instances of RDFReifiedStatement.

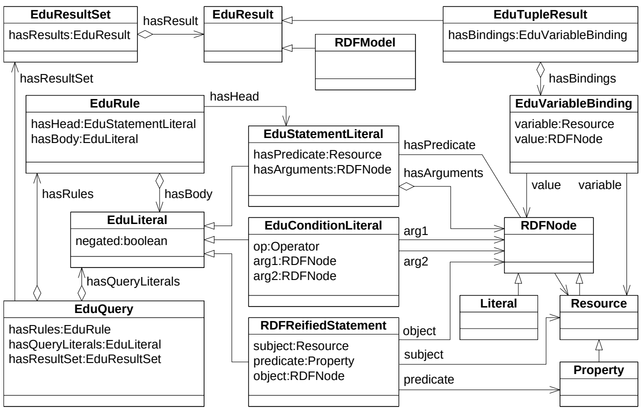


Figure 4: Edutella Common Data and Query Exchange Model (ECDM)

An `EduRule` consists of an `EduStatementLiteral` as its head and an arbitrary number of `EduLiterals` as its body. `EduStatementLiterals` can occur within a rule’s body as well to allow reuse of other rules and recursion.[3] In database terms, `EduStatementLiterals` are intensional predicates, and are defined through the head of rules. `RDFReifiedStatements` are extensional predicates, and are stored explicitly in the RDF database. Therefore, `RDFReifiedStatements` can be expressed by binary predicates / ternary predicate statements, while `EduStatementLiterals` can have more than two arguments for the predicate.

Results are represented either as a set of `RDFModel` or `EduTupleResult` objects depending on whether the results are requested to be in RDF graph or tuple format. In the latter case each `EduTupleResult` aggregates a number of `EduVariableBinding` objects - one for each variable within the query.

4.4 Edutella Wrapper API

The following sketches the current prototypical Edutella Wrapper API, version 0.8, used as a blueprint in our current Edutella wrappers, to enable Edutella peers to handle our Edutella common data model in a coherent manner. The API will most likely change in subsequent versions, but its structure gives a good overview over the functionalities this API has to provide.

The Java binding (available from the Edutella Project Page[4]) is composed of the following packages:

[1] These views can be combined, though, see, e.g., [20] and[14], and to some extent RDFS, which specifies classes in an object oriented way, even though it does not introduce object identity (though it can easily be extended with it, see [29]).

[2] see also the discussion in Section 4.7

[3] Note, that as input format we can even allow arbitrary first order logic formulas in the body of rules, which then can be transformed into a set of rules using the Lloyd-Topor transformation [24].

[4] http://edutella.jxta.org/

`net.jxta.edutella.util.datamodel`: Contains all classes for the Edutella common data model as described in Figure 4. This common model is used for transmitting queries within the Edutella network.

`net.jxta.edutella.util`: Contains classes RDF-QEL-1, RDF-QEL-2, etc. for importing queries given in the respective formats into the internal query model or in turn export queries from the internal model into other syntaxes and representations.

`net.jxta.edutella.peer`: Contains general service implementations for Edutella peers, e.g., Edutella provider service, consumer service, and hub service, etc. Also the general interfaces for Edutella provider adapter, hub adapter, and consumer adapter are defined in this package.

`net.jxta.edutella.provider`: Contains a general Edutella Provider implementation which runs an Edutella provider service. Various Edutella providers can realize different adapters, which correspond to different back-end repositories, and embed these adapters into the general implementation as plug-ins.

`net.jxta.edutella.hub`: Contains a general Edutella hub implementation which runs an Edutella hub service. Various Edutella hubs can adopt different mechanisms to handle distributed query and implement various query mediators in the form of plug-ins to the general Edutella hub implementation.

`net.jxta.edutella.consumer`: Contains a general Edutella consumer implementation which runs an Edutella consumer service. Various Edutella consumers can realize different adapters to provide different presentations of query results.

4.5 RDF-QEL-i Language Levels

In the definition of the Edutella query exchange language, several important design criteria have been formulated:

Standard Semantics of query exchange language, as well as a sound RDF serialization. Simple and standard semantics of the query exchange language is important, as transformations to and from this language have to be performed within the Edutella peer wrappers, which have to preserve the semantics of the query in the original query language. A sound encoding of the queries in RDF to be shipped around between Edutella peers has to be provided.

Expressiveness of the language. We want to interface with both simple graph based query engines as well as SQL query engines and even with inference engines. It is important that the language allows expressing simple queries in a form that simple query providers can directly use, while allowing for advanced peers to fully use their expressiveness.

Adaptability to different formalisms. The query language has to be neutral to different representation semantics, it should be able to use any predicates with predefined semantics (like `rdfs:subclassOf`), but not have their semantics built in, in order to be applicable to different formalisms used by the Edutella peers. It should easily connect to simple RDFS repositories, relational databases and object-relation ones, as well as to inference systems, with all their different base semantics and capabilities.

Transformability of the query language. The basic query exchange language model must be easy to translate to many different query languages (both for importing and exporting), allowing easy implementation of Edutella peer wrappers.

Edutella follows a layered approach for defining the query exchange language. Currently we have defined language levels RDF-QEL-1, -2, -3, -4 and -5, differing in expressiveness. The most simple language (RDF-QEL-1) can be expressed as unreified RDF graph, the complex ones are more expressive than RDF itself and therefore have to be expressed using reified RDF statements. All language levels can be represented through the same internal data model.

4.5.1 RDF-QEL-1

The RDF-QEL-1 syntax design is driven by its simplicity and readability: Following a QBE (Query By Example) paradigm queries are represented using ordinary RDF graphs having exactly the same structure as the answer graph, with additional annotations to denote variables and constraints on them. Any RDF graph query can be interpreted as a logical (conjunctive) formula that is to be proven from a knowledge base.

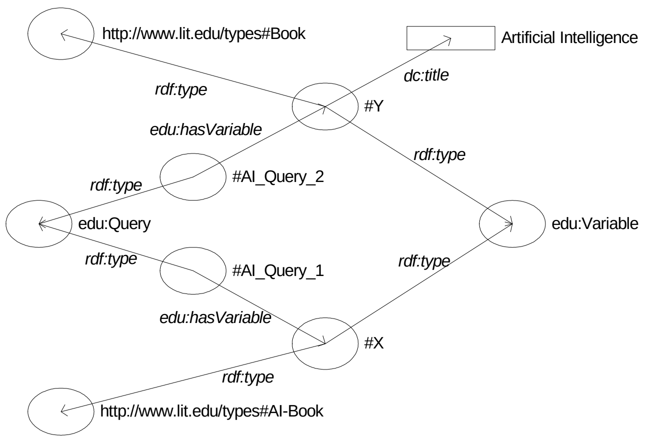


Figure 5: Example Query in RDF-QEL-1, Unreified Format

Since disjunction cannot be expressed in RDF-QEL-1 our example query has to be split into two separate sub queries (Figure 5).

```
<edu:QEL1Query rdf:ID="AI_Query_1">
  <edu:hasVariable rdf:resource="#X"/>
</edu:QEL1Query>

<edu:Variable rdf:ID="X" rdfs:label="X">
  <rdf:type
   rdf:resource="http://www.lit.edu/types#AIBook"/>
</edu:Variable>

<edu:QEL1Query rdf:ID="AI_Query_2">
  <edu:hasVariable rdf:resource="#Y"/>
</edu:QEL1Query>

<edu:Variable rdf:ID="Y" rdfs:label="X">
  <rdf:type
   rdf:resource="http://www.lit.edu/types#Book"/>
  <dc:title>Artificial Intelligence</dc:title>
</edu:Variable>
```

4.5.2 RDF-QEL-2

Extending RDF-QEL-1 with disjunction leads to RDF-QEL-2. As this language is no longer purely assertional, it cannot be expressed directly in RDF without talking *about* RDF triples in order to combine them logically. For this purpose, we utilize the RDF construct called *reification*. Reifying an RDF statement involves creating a model of the RDF triple in the form of an RDF resource of type Statement. This resource has as properties the subject, the predicate and the object of the modeled RDF triple. Such reified statements are the building blocks for each query and can, in RDF-QEL-2, linked together by an AND-OR tree. In RDF-QEL-2 the example query reads like

```
<edu:Variable rdf:about="#X" rdfs:label="X"/>

<edu:And rdf:about="#andbagID">
  <rdf:_2 rdf:resource="#st2"/>
  <rdf:_1 rdf:resource="#st3"/>
</edu:And>

<edu:Or rdf:about="#orbagID">
  <rdf:_1 rdf:resource="#andbagID"/>
  <rdf:_2 rdf:resource="#st1"/>
</edu:Or>

<edu:QueryStatement rdf:about="#st1">
  <rdf:subject rdf:resource="#X"/>
  <rdf:object
   rdf:resource="http://www.lit.edu/types#AIBook"/>
  <rdf:predicate rdf:resource="rdf:type"/>
</edu:QueryStatement>

<edu:QueryStatement rdf:about="#st2">
  <rdf:subject rdf:resource="#X"/>
  <rdf:object
   rdf:resource="http://www.lit.edu/types#Book"/>
  <rdf:predicate rdf:resource="rdf:type"/>
</edu:QueryStatement>

<edu:QueryStatement rdf:about="#st3">
  <rdf:object>Artificial Intelligence</rdf:object>
  <rdf:subject rdf:resource="#X"/>
  <rdf:predicate rdf:resource="dc:title"/>
</edu:QueryStatement>
```

The advantage of the RDF-QEL-2 form is that queries can easily be visualized using a query graph. The Conzilla query interface [30] is based on a subset of UML, using the UML specialization relationship for logical OR and the UML aggregation relationship for logical AND. As shown in Figure 6, our current prototype uses a graph-view, which is displayed as ordinary RDF with the exception that the triplets searched for (which are reified in RDF-QEL-i, where $n > 1$) are displayed as dashed arrows indicating that they are searched for. The logical view is displayed as a parse tree. This is the logical combination of the primitive statements, showing which combinations that should be matched at the same time in order for the query to succeed. The connections between the different views are displayed by highlighting the corresponding parts.

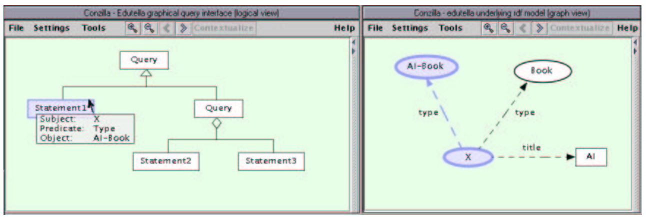

Conzilla - Edutella graphical query interface (logical view)
File Settings Tools Contextualize Help
Query
Statement1
Subject: X
Predicate: Type
Object: AI-Book
Query
Statement2
Statement3
Conzilla - edutella underlying rdf model (graph view)
File Settings Tools Contextualize Help
AI-Book
Book
type
type
X
title
AI

Figure 6: Edutella Graph Query Interface

Queries can be stored and reused later, thus we can work with a library of queries that can be combined to new queries. Those queries can either be used as is or as templates, where sub-strings, numerical values, etc are filled in. Details of sub-queries can be suppressed by hiding them in detailed maps that can be presented hierarchically.

4.5.3 RDF-QEL-3

Going a step further, we might actually choose to skip RDF-QEL-2 in favor of RDF-QEL-3, which allows conjunction, disjunction and negation of literals. RDF-QEL-3 is essentially Datalog. Hence, the query is a set of Datalog rules, which can be encoded easily using reified statements (as for RDF-QEL-2), introducing additional constructs for negation and implication. As long as queries are non-recursive this approach is relationally complete. The example query expressed in RDF-QEL-3 resembles the internal Datalog model described above.

```
<edu:QEL3Query rdf:ID="AI_Book_Query">
 <edu:hasQueryLiteral rdf:resource="st0"/>
 <edu:hasRule rdf:resource="r1"/>
 <edu:hasRule rdf:resource="r2"/>
</edu:QEL3Query>

<edu:Variable rdf:ID="X" rdfs:label="X"/>

<edu:Rule rdf:ID="r1">
 <edu:hasHead rdf:resource="st0"/>
 <edu:hasBody rdf:resource="st2"/>
 <edu:hasBody rdf:resource="st3"/>
</edu:Rule>

<edu:Rule rdf:ID="r2">
 <edu:hasHead rdf:resource="st0"/>
 <edu:hasBody rdf:resource="st1"/>
</edu:Rule>

<edu:QueryStatement rdf:ID="st0">
 <edu:predicate rdf:resource="aibook"/>
 <edu:arguments>
  <rdf:Seq>
   <rdf:_1 rdf:resource="#X"/>
  </rdf:Seq>
 </edu:arguments>
</edu:QueryStatement>

<edu:QueryStatement rdf:ID="st1">
 <rdf:subject rdf:resource="#X"/>
 <rdf:object
  rdf:resource="http://www.lit.edu/types#AIBook"/>
 <rdf:predicate rdf:resource="rdf:type"/>
</edu:QueryStatement>

<edu:QueryStatement rdf:ID="st2">
 <rdf:subject rdf:resource="#X"/>
 <rdf:object
  rdf:resource="http://www.lit.edu/types#Book"/>
 <rdf:predicate rdf:resource="rdf:type"/>
</edu:QueryStatement>

<edu:QueryStatement rdf:ID="st3">
 <rdf:object>Artificial Intelligence</rdf:object>
 <rdf:subject rdf:resource="#X"/>
 <rdf:predicate rdf:resource="dc:title"/>
</edu:QueryStatement>
```

4.5.4 Further RDF-QEL-i Levels

RDF-QEL-4: RDF-QEL-4 allows recursion to express transitive closure and linear recursive query definitions, compatible with the SQL3 capabilities. So a relational query engine with full conformance to the SQL3 standard will be able to support the RDF-QEL-4 query level.

RDF-QEL-5: Further levels allow arbitrary recursive definitions in stratified or dynamically stratified Datalog, guaranteeing one single minimal model and thus unambiguous query results ([31]).[5]

[5]Technically, when using negation, recursion and the ternary representation of statements, static stratification can never be guaranteed

RDF-QEL-i-A: Support for the usual aggregation functions as defined by SQL2 (e.g., COUNT, AVG, MIN, MAX) will be denoted by appending "-A" to the query language level, i.e., RDF-QEL-1-A, RDF-QEL-2-A, etc. RDF-QEL-i-A includes these aggregation functions as edu:count, edu:avg, edu:min, etc. Additional "foreign" functions like edu:substring etc. to be used in conditions might be useful as well, but have not been included yet in RDF-QEL-i-A.

4.6 Representing Complex Property Semantics

RDFS already comes with predefined semantics for certain properties (i.e., transitiveness of rdfs:subclassof, inheritance for rdf:type). Whenever the query includes these pre-defined predicates, we presume these to have the pre-defined semantics. This is valid for DAML-OIL pre-defined predicates and their semantics as well, i.e., if we use definitions like

```
<daml:TransitiveProperty rdf:ID='hasAncestor/>
```

then transitivity of `hasAncestor` is assumed

```
hasAncestor(X,Y) :- hasAncestor(X,Z),
                    hasAncestor(Z,Y).
```

without having to be specified explicitly in the query.

If we want to specify something else, we have in principle to specify its semantics as Datalog rule, and ship it with the query. However, we can add special annotations like `edu:transitive_closure_of` (denoting transitive closure of properties), `edu:inherited_version_of` (inheritance of properties along the subclassOf-hierarchy), `edu:reflexive_version_of` (reflexive version of a property) to properties, which can be used directly by the Edutella peer wrapper (whenever it knows what these `edu:properties mean`). This has the advantage, that the wrapper does not have to infer the correct semantics from the corresponding Datalog rules, but can use the predefined semantics for these `edu:properties` directly.[6] This keeps the clear semantics for RDF-QEL-i, but allow abbreviations which make it easier to write Edutella peer wrappers.

4.7 Querying Schema Information

As apparent already from the RDFS schema definition [2], and discussed in more detail in the recent RDF model theory [12] (see also the axiomatic definition of an extension of RDFS we called O-Telos-RDF [29]), RDFS does not distinguish between data and schema level, and represents all information in a uniform way as a graph. Indeed, as discussed in [29] in some more details, there is no principal difference between entities at different modeling levels (i.e., objects, classes and meta-classes are represented in a uniform way), and queries over an RDFS schema should not be more difficult than queries over RDFS data.

Therefore our internal query exchange model as shown in Figure 4 treats entities on all levels in a uniform way (as RDFNodes), and the attributes of `EduStatementLiterals` can be entities on different levels (objects, classes or even predicates). Therefore, representing queries at different levels does not pose problems.

In order to express Datalog like queries ranging over different abstraction levels, instead of writing properties as binary predicates, we have to switch to a triple syntax using a ternary predicate "s", i.e., instead of writing "book(X,'Artificial Intelligence')" we write "s(X,book,'Artificial Intelligence')". If we enforce the restriction, that the predicate symbol "s" always denotes this special ternary predicate, we can also mix this notation with the binary predicate notation we used so far in our examples. Generalizing the query from our running example a bit, we now want to ask for any additional property our AI books might have, getting the query:

```
aibook(X) :- title(X, 'Artificial Intelligence'),
             type(X, Book).
aibook(X) :- type(X, AI-Book).

book_property(P) :- s(P, rdfs:domain, Book).
ai_book_property(P) :- s(P, rdfs:domain, AI-Book).

ai_book_attribute(X,P,V) :-
    aibook(X), book_property(P), s(X,P,V).
ai_book_attribute(X,P,V) :-
    aibook(X), ai_book_property(P), s(X,P,V).

?- ai_book_attribute(X,P,V)
```

4.8 Result Formats

4.8.1 *Standard Result Set Syntax*

As a default, we represent query results as a set of tuples of variables with their bindings. Referring to our example there are two bindings for a single variable. This is also shown in Figure 4, and closely follows the convention of returning substitutions for variables occurring in queries to logic programs.

```
<edu:ResultSet rdf:ID="AI_Results">
  <edu:hasResult rdf:parseType="Resource">
    <rdf:type rdf:resource="edu:TupleResult"/>
    <edu:hasVariable rdf:parseType="Resource">
      <rdf:type rdf:resource="edu:VariableBinding"/>
      <edu:bindsVariable rdf:resource="#X"/>
      <rdf:value
       rdf:resource="http://www.xyz.com/ai.html"/>
    </edu:hasVariable>
  </edu:hasResult>
  <edu:hasResult rdf:parseType="Resource">
    <rdf:type rdf:resource="edu:TupleResult"/>
    <edu:hasVariable rdf:parseType="Resource">
      <rdf:type rdf:resource="edu:VariableBinding"/>
      <edu:bindsVariable rdf:resource="#X"/>
      <rdf:value
       rdf:resource="http://www.xyz.com/pl.html"/>
    </edu:hasVariable>
  </edu:hasResult>
</edu:ResultSet>
```

4.8.2 *RDF Graph Answers*

Another possibility, which has been explored recently in Web related languages focusing on querying semistructured data (for an overview see, e.g., [1]), is the ability to create objects as query results. In the simple case of RDF-QEL-1, we can return as answer objects the graph representing the RDF-QEL-1 query itself with all Edutella specific statements removed and all variables instantiated. The results can be interpreted as the relevant sub graph of the RDF graph we are running our queries against (see Figure 3). In other words, the answer graph contains sufficient information, so that running the query using only the data in the answer graph returns the same result as running the query against the original database.

```
<lib:Book about="http://www.xyz.com/ai.html">
  <dc:title>Artificial Intelligence</dc:title>
</lib:Book>
<lib:AI-Book about="http://www.xyz.com/pl.html"/>
```

(because we only use one ternary predicate "s(S,P,O)"), so we have to rely on dynamic stratification (which depends on the actual instantiation of literals) or switch to well-founded semantics.

[6] These predefined `edu:properties` can be seen as macros, which take the corresponding properties and generate the corresponding Datalog definitions (see also the discussion in Section 5.1).

When we use general RDF-QEL-i queries, we assume the structure of the answer graph to be defined by the query literals (provided they are all binary predicates). Note, that all variables used in the query literals are assumed to be existentially quantified, so if they are not instantiated during the query evaluation, they are represented as anonymous nodes in the RDF answer graph (as discussed in [12]).[7]

If we also allow skolem functions in the head literals of our rules, we are able to generate arbitrary complex objects, which use these skolem values as object IDs (see also [1]). This use of skolem functions was introduced in proposals unifying logic and object oriented formalisms, first informally discussed in [26] and refined in later proposals, e.g., [20]. Even though RDF does not allow skolem functions, we can return the answer objects as RDF graphs in such cases with anonymous nodes substituted for each skolem value (different anonymous nodes for different skolem values).

If we use existing resources to group the properties of an object, we can use the resource URI directly instead of introducing a new skolem value, as in our previous example, with a slight modification in the query literals:

```
?- ai_book_attribute(X,P,V), s(X,P,V).
```

In this case, we can group around the resources of type book all their properties, and therefore return as answer objects the sets of s-triples corresponding to the resource-property-value triples each book found through the query (i.e., the "hedgehogs" centered around the URIs of the books).

5. WRAPPING DIFFERENT PEER QUERY LANGUAGES

The following sections are not meant to be complete characterizations of the mappings but rather sketch these mappings and translate our example query into the local query language. Further details will be found in a forthcoming report.

5.1 RQL

RQL is an RDF query language described in [19] and [3], and used within the EU-IST project On-To-Knowledge. RQL focuses on SQL like query expressions, exploiting path expressions, implicit and explicit joins, and the usual comparison operators. All examples in [19] and [3] can be expressed using conjunctive queries, though the formal RQL specification also includes all set operations (union, intersect and minus), making it relationally complete.

The default for queries including `typeOf` and `subclassOf` is to use transitivity of `subclassOf`, as defined by RDFS. These queries would be translated using simple `typeof(X,Y)` and `subclassof(X,Y)` binary predicates, as the transitivity of `subclassof` is reflected in the query engine, not in the query language.

Additionally, RQL specifies the variants `typeOf^` and `subclassOf^` ("direct" `typeOf` and `subclassOf`), which can be defined in Datalog as follows (assuming `subclassOf` not to be reflexive, as advocated in [2]):

```
typeof^(X,Y) :- typeof(X,Y),
                not(typeof(X,Z), subclassof(Z,Y)).

subclassof^(X,Y) :-
                subclassof(X,Y),
                not(subclassof(X,Z), subclass(Z,Y)).
```

[7]Anonymous nodes, i.e., existential variables in the RDF graph itself, can be handled by the usual Lloyd-Topor transformation [24].

or the other way around, if we assume that the local peer stores only the "typeof^ " and "subclassof^ " facts

```
subclassof(X,Y) :- subclassof^(X,Y).
subclassof(X,Y) :- subclassof(X,Z), subclassof^(Z,Y).

typeof(X,Y) :- typeof^(X,Y).
typeof(X,Y) :- subclassof(Z,Y), typeof^(X,Z).
```

and alternatively using the predefined properties "edu:inherited_version_of" and "edu:transitive_closure_of"

```
<rdf:Property rdf:ID="typeOf">
    <edu:inherited_version_of
     rdf:resource="#typeOf^"/>
</rdf:Property>

<rdf:Property rdf:ID="subclassOf">
    <edu:transitive_closure_of
     rdf:resource="#subclassOf^"/>
</rdf:Property>
```

which assumes the corresponding Datalog definitions above. Translating these special constructs "typeof^ " and "subclassof^ " needs additional recursive definitions in the query exchange language (i.e., RDF-QEL-5 capability). However, if the Edutella wrapper for an RQL peer understands "edu:inherited_version_of" and "edu:transitive_closure_of", these definitions can be automatically generated, so we can avoid shipped them along with the query, and stay with RDF-QEL-3 (non-recursive Datalog) queries.[8]

Example query in RQL:

```
select X
from Book{X}.title{Y}
where Y = "Artificial Intelligence"
UNION
select X
from AI-Book{X}
```

Structurally and syntactically the query looks similar to its SQL counterpart. RQL uses its own syntax and does not come with any RDF XML serialization. The RDF type statements do not need to be made explicit since the RDFS class concept is an inherent part of RQL. Both sub queries use linear path expressions. In case the queries are based on more complex graph structures several linear path expressions are enumerated in the FROM clause and have to be joined explicitly by a WHERE clause. For querying schema information, which is also possible in RQL, we translate the RQL query expressions into Edutella Datalog programs using the ternary notation of triples, as discussed in Section 4.7.

5.2 TRIPLE

TRIPLE is an RDF query, inference, and transformation language [35] based on Horn logic F-Logic [20]. TRIPLE's architecture allows semantic features to be defined for various object-oriented and other RDF extensions like RDF Schema. TRIPLE provides a (human readable) Prolog-like syntax as well as an RDF-based syntax for exchanging queries and rules. In the Prolog-like syntax, RDF statements are written as *molecules*, i.e.,

$$subject[predicate -> object]$$

or, for multiple predicate-object pairs for one subject,

$$subject[pred_1 -> obj_1; pred_2 -> obj_2; ...]$$

[8]The peer actually does not need to use these recursive Datalog definitions directly, but instead simply can translate properties like "typeof^ " and "subclassof^ " into the local versions implementing the specified semantics.

Our sample knowledge base and query can be mapped as follows:

```
// namespace abbreviations
rdf := 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'.
dc := 'http://purl.org/dc/elements/1.0/'.
types := 'http://www.lit.edu/types#'.
xyz := 'http://www.xyz.com/'.
// sample knowledge base
xyz:'sw.html'[ rdf:type -> types:Book;
     dc:title -> 'Software Engineering' ].
xyz:'ai.html'[ rdf:type -> types:Book;
     dc:title -> 'Artificial Intelligence' ].
xyz:'pl.html'[ rdf:type -> types:'AI-Book';
     dc:title -> 'Prolog' ].
// sample query
FORALL X  aibook(X) <-
        X[rdf:type -> types:'AI-Book']
     OR X[rdf:type -> types:Book;
          dc:title -> 'Artificial Intelligence'].
```

TRIPLE regards RDF data not as one large heap, but partitions the set of RDF data in different subsets, called RDF models. Different subsets can come from different sources, have different semantics etc. TRIPLE supports models, parameterized models, model expressions, etc., which are useful extensions to RDF-QEL-i as well, when we want to concentrate more on data integration and transformation using different sources.

In general, at least RDF-QEL-3 is needed to capture TRIPLE programs, which are then very close to our internal data model (both being based on Horn logic). Currently unsupported in RDF-QEL-3 are some additional features of TRIPLE, e.g., functional terms and RDF models as sets of statements.

5.3 SQL

To keep the discussion simple, we assume a single STATEMENTS table (i.e., we use the triple-oriented view and ternary statements) storing all statements the repository is aware of in a relational database.[9] The table consists of three columns SUBJECT, PREDICATE and OBJECT of type character string with each column value being interpreted either as concatenation of namespace and resource name or as literal value. More sophisticated database schemas might provide a view according to this one-table schema.

The mapping of the example query is straightforward. In terms of its Datalog representation the query may be satisfied by either of its two rules with the first being a conjunction of two literals and the second only involving a single literal. The disjunction of the two rules is mapped to a `UNION` of two sub queries. This query structure directly resembles the RDF-QEL-3 syntax as well as the internal Datalog query model.

```
SELECT S1.SUBJECT
FROM STATEMENT S1, STATEMENT S2
WHERE S1.PREDICATE =
 'http://purl.org/dc/elements/1.1/#title'
AND S1.OBJECT = 'Artificial Intelligence'
AND S2.PREDICATE =
 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
AND S2.OBJECT =  'http://www.lit.edu/types#Book'
AND S1.SUBJECT = S2.SUBJECT)
UNION
SELECT S1.SUBJECT
FROM STATEMENT S1
WHERE S1.PREDICATE =
 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
AND S1.OBJECT =  'http://www.lit.edu/types#AIBook'
```

[9] Some RDF stores include additional information like models, etc., which we neglect in this section.

More complicated queries with rules referencing other rules in their bodies need to be modeled either as nested queries or as derived relations [34]. It is not possible to map queries containing recursive rules to traditional SQL (that is SQL92 or below), therefore SQL2 only maps queries up to RDF-QEL-3. With SQL3 supporting linear recursion we are also able to map RDF-QEL-4 queries. For those more familiar with relational algebra expressions, we also provide the query in relational algebra. In contrast to the statement centric view used in the SQL mapping, we use a predicate centric approach here, with one relation (or table) for each predicate.

$$\Pi_{\mathrm{TYPE.subject}} \\ ((\sigma_{\mathrm{object=Artificial\ Intelligence}}(\mathrm{TITLE}) \bowtie \sigma_{\mathrm{object=Book}}(\mathrm{TYPE})) \\ \cup (\sigma_{\mathrm{object=AIBook}}(\mathrm{TYPE})))$$

5.4 XPath/dbXML

The open source native XML database dbXML [6] provides a natural way to store XML-based learning resource meta-data. In the following we show an example knowledge base stored in our dbXML-based meta-data repository (using DCMI RDF/XML binding [21]):

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
 <rdf:Description
  rdf:about="http://www.xyz.org/ai.html">
  <dc:title>Artificial Intelligence</dc:title>
  <dc:type>Book</dc:type>
 </rdf:Description>

 <rdf:Description
  rdf:about="http://www.xyz.org/sw.html">
  <dc:title>Software Engineering</dc:title>
  <dc:type>Book</dc:type>
 </rdf:Description>

 <rdf:Description
  rdf:about="http://www.xyz.org/pl.html">
  <dc:title>Prolog</dc:title>
  <dc:type>AI-Book</dc:type>
 </rdf:Description>
</rdf:RDF>
```

Because dbXML employs the W3C XPath language [4] to accomplish its query service, the first task for dbXML content providers is to map RDF-QEL-1 to a query representation in XPath. XPath provides several XML-specific features (like retrieving hierarchical substructures from a whole XML document). If we abstract from these features and focus on the features comparable to a relational query language, XPath basically provides select statements identifying specific tags within XML documents. So conjunctive queries in RDF-QEL-1 can be mapped into conjunctions of XPath expressions.

In case the XML-database also supports disjunction between XPath expressions, we can also translate RDF-QEL-2 queries into appropriate XPath queries using both AND and OR Boolean connectives. The translation between XML- and RDF-schemas such as IMS or LOM for learning resources is especially easy, because these schemas basically define hierarchically structured metadata, where the differences between RDF and XML hardly matter. Our example query can be written in the syntax of XPath as

```
/rdf:RDF/rdf:Description/dc:title[text()=
 "Artificial Intelligence"]
AND
/rdf:RDF/rdf:Description/dc:type[text()="Book"]
```

```
OR
/rdf:RDF/rdf:Description/dc:type[text()="AIBook"]
```

and the results will be

```
<rdf:Description
 rdf:about="http://www.xyz.com/ai.html">
 <dc:title>Artificial Intelligence</dc:title>
 <rdf:type>Book</rdf:type>
</rdf:Description>

<rdf:Description
 rdf:about="http://www.xyz.com/pl.html">
 <dc:title>Prolog</dc:title>
 <rdf:type>AI-Book</rdf:type>
</rdf:Description>
```

Note, that the default behaviour for XPath result sets in dbXML is different from the result sets defined for our RDF-QEL-i queries. RDF-QEL-i queries return exactly the tuples which we mention in the query, while XPath queries in dbXML by default return the whole sub-document located below the element selected by the XPath expression, but might also identify the whole document matching this expression.

5.5 AmosQL and Mediators

Amos II [33] is a distributed mediator engine where views of data from several different data sources can be defined. The views are defined using the object-relational query language AmosQL. AmosQL is relationally complete.

It is relatively simple to translate RDF and RDF-Schema metadata descriptions to AmosQL schema manipulation statements. RDF-QEL-i queries can the also easily be translated to AmosQL statements, as will be shown. However, it is more challenging to represent the mediators themselves in RDF or RDF-Schema, the reasons being that Amos II has a rich object-oriented data model, e.g., many data types and mediation primitives, which are needed for mediation from many different kinds of data sources.

Types in Amos II correspond to *classes* in RDF-Schema. An Amos II object is classified into one or more *types* making the object an *instance* of those types. The set of all instances of a type is called the *extent* of the type. The types are organized in a multiple inheritance, supertype/subtype hierarchy. If an object is an instance of a type, then it is also an instance of all the supertypes of that type; conversely, the extent of a type is a subset of the extent of a supertype of that type (extent-subset semantics).

Meta-objects (system objects) in Amos II mediators, such as types and function, are first class and can be queried as any other objects. This makes it straight-forward to represent RDF-Schema classes as Amos II types. The transparent representation of meta-objects in the mediators allows powerful queries about the capabilities and structure of each mediator.

Object attributes, queries, methods, and relationships are modeled by *functions* in Amos II. Depending on their implementation the functions can be classified into several kinds including *stored functions* that represent facts and correspond to *properties* in RDF, *derived functions* that represent views and correspond to *rules* in Datalog, and *foreign functions* that implement algorithms external to the query language (e.g., in Java). When wrapping external data sources with Amos II the multi-directional foreign function facility [33] provides the primitive to specify access paths and capabilities of the sources. The general syntax for AmosQL queries is:

```
select <result>
   from <domain specifications>
   where <condition>
```

Each *domain specification* associates a query variable with a type where the variable is universally quantified over the extent of the type, including indefinite extents as integers with some restrictions. This is different to SQL where variables range over tuples in tables.

A query compiler translates AmosQL statements into object calculus and algebra expressions in an internal simple logic based language called ObjectLog [23], which is an OO dialect of Datalog where predicates are typed and can be overloaded.

Since ObjectLog is used internally by Amos II it is relatively easy to translate RDF-QEL-i into AmosQL. However, certain conventions need to be introduced to, e.g., RDF-QEL-1 in order for the translation to be straight-forward: `typesof` predicates where the second argument denotes types must be inserted first in rules, RDF properties cannot be overloaded, but name spaces provide a means to make them unique. It is thus no problem to represent RDF properties as stored functions in Amos II, and RDF-specific meta-resources, e.g., classes and properties are mapped to corresponding Amos II types. AmosQL does not permit recursive views; instead a transitive closure meta-function is provided to handle most situations requiring recursive views.

6. REGISTRATION SERVICE AND QUERY MEDIATORS

The *wrapper-mediator* approach introduced in [38], divides the functionality of a data integration system into two kinds of subsystems. The *wrappers* provide access to the data in the data sources using a *common data model* (CDM) and a *common query language*. The *mediators* provide coherent views of the data in the data sources by performing semantic reconciliation of the CDM data representations provided by the wrappers. Both common data model (ECDM) and common query language for the Edutella network have been defined in this paper.

To mediate distributed data sources we are using a two-layered approach: Simple 'wrapping' mediators distribute queries to the appropriate peer with the restriction that queries can be answered completely by one Edutella peer. Complex 'integrating' mediators are able to mediate distributed queries over multiple repositories. The query syntax to queries to both kinds of mediator will be identical in both cases.

6.1 Simple Wrapping Mediators

The first layer of functionality for distributed queries in the Edutella network will be based on simple query hubs and wrapping mediation. While query hubs might have some wrapping capability, our prototype peers will use them only as registration and query distribution peers using the Edutella common data and query model, and implement wrapping capability (to and from the common model) locally within the Edutella peer wrappers as discussed in Section 4.4. Thus, each Edutella peer offers a common query interface based on the common model (possibly at different levels as defined by RDF-QEL-i) to the network.

Registration of peer query capabilities is based on (instantiated) property statements and schema information, telling the network, which kind of schema the peer uses, with some possible value constraints (select conditions). These registration messages have the same syntax as RDF-QEL-1 queries, which are sent from the peer to the registration / query distribution hub. Additionally, the peer announces to the hub, which query level it can handel (RDF-QEL-1, RDF-QEL-2, etc.) Whenever the hub receives queries, it uses these registrations to forward queries to the appropriate peers, merges the results, and sends them back as one result set.

6.2 Mediator Peers Handle Distributed Queries

The second layer introduces query mediators or query hubs. These mediators bring in the extra intelligence required to assemble distributed and heterogeneous queries. These more complex mediators submit subqueries to different repositories that might be able to answer them, collect the sub-results, join and reconcile them, and again return the outcome to the client.

Several mediator servers will be available communicating through JXTA. Each mediator server has its own mediator meta-data schema and accesses meta-data from other mediators or data sources. The views provided through the integrating mediators are transparently queryable using RDF-QEL-i.

In Amos II each mediator server appears as a virtual database layer having object-oriented data abstractions and query language. Object-oriented views provide transparent access to the data sources from clients and other mediator servers. Conflicts and overlaps between similar real-world entities being modeled differently in different data sources are reconciled through the mediation primitives [16, 17] of Amos II which are translated to ObjectLog. The mediation services allow transparent access to similar object structures represented differently in different data sources.

The representation of integrating mediators in RDF requires a richer data model than what is currently available in RDF or RDF-Schema. Alternatively various conventions can be introduced in the RDF-based meta-data definitions, e.g., some convention is needed on how to represent type annotated and generic Datalog rules, since ObjectLog rules can be overloaded on types. A somewhat inelegant way would be to use different name spaces for this but type annotated properties seem more convenient. Second, RDF currently does not have views and can therefore not represent mediators that join data from different sources. Named RDF-QEL-i queries would provide a way to specify views. Derived Amos II functions would correspond to derived properties defined as named RDF-QEL-i queries. Third, the mediation primitives for reconciling overlapping and conflicting information in data sources need RDF bindings.

Mediators can cooperate by being defined in terms of other mediators, i.e., the mediators are *composable* [15, 33]. The composition of mediators allows for modularity and reuse of the view definitions while avoiding the administrative and performance bottleneck of having a single mediator system with a global schema. Different interconnecting topologies can be used to compose mediator servers depending on the integration requirements.

7. PROTOTYPE SCENARIO

Our current prototype setup features a set of (already existing) peers, which we have extended with the appropriate Edutella wrappers, and which connect to the Edutella framework with the following functionalities: *local query* (directly to repository), *distributed query* (mediated by a simple wrapper mediator and by an AMOS II mediator peer) and *update* (through annotation peer).

The following peers can be connected to the Edutella network using the Edutella wrapper libraries: *OLR Repository peer* [7], based on subset of IMS/LOM metadata, will be able to translate from RDF-QEL-3 into internal query language SQL, return results in specified result format, *DbXML peer* [32], as a prototype for an XML-DB, based on subset of subset of IMS/LOM metadata, using a simple mapping service to translate from RDF-QEL-1 queries to Xpath queries over the appropriate XML-LOM schema, *AMOS II peer* (with local repository) [33], translate from RDF-QEL-3 into AmosQL, *Simple query and registration hub*, distribute queries based on schema information and query capabilities, *Complex mediation peer*, mediate queries on AMOS II based mediation, uses one AMOS II peer and one OLR repository peer, *Graphical query interface peer* based on Conzilla [30], take a graph, and translate it to a query expression, which then can be pushed into the Edutella network, visualize results, with RDF-QEL-1 and RDF-QEL-2 functionality *JXTA shell peer* as well as textual interface implemented via servlet, for direct query input in RDF-QEL-3, *Annotation peer* based on Ontomat [11], query a repository with a query, update/annotate the results, write them back to the repository, *KAON-Server*[10], formerly OntoBroker, see also [25], *Storage and computation peer* with Datalog capabilities for RDFS and O-Telos-RDF ([29], based on ConceptBase Server [14]), and a *File based repository peer* based on the JENA toolkit, with the corresponding query language RDQL [27], which stores its RDF data in files.

All source code of the Edutella implementation can be downloaded from the Edutella Project Page.

8. SUMMARY AND ACKNOWLEDGEMENTS

While in the server/client-based environment of the World Wide Web metadata are useful and important, for P2P environments metadata are absolutely crucial, in order to describe the resources managed by these peers. So far, P2P applications have used domain specific formats and metadata schemas, leading to a fragmentation of the P2P worlds into niche markets.

In this paper, we have described the current status of the Edutella project, which addresses these shortcomings of current P2P applications by building on the W3C metadata standard RDF. The project is a multi-staged effort to scope, specify, architect and implement an RDF-based metadata infrastructure for P2P-networks based on the recently announced JXTA framework. We have described the main architecture and services provided by the Edutella framework, and have discussed in detail the Edutella query service, which defines a common query exchange model and language used to exchange queries and query results between Edutella peers. We have further discussed the basic registration and mediation services for distributed queries.

Our vision is to provide the metadata services needed to enable interoperability between heterogeneous JXTA applications. Our first application will focus on a P2P network for the exchange of educational resources, and we have described the prototype environment we are using to test the Edutella framework. This prototype environment will be up and running at the end of this year, further work will concentrate on refining the existing architecture and scalability of the Edutella network and add further kinds of peers and services to the network.

Acknowledgements. This paper is based on a lot of fruitful discussions with participants within the PADLR projects. We especially want to thank Steffen Staab and Raphael Volz from AIFB, Wolf Siberski, Martin Wolpers and Hadhami Dhraief from KBS, and Gustav Neumann and Bernd Simon from Vienna.

9. REFERENCES

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web*. Morgan Kaufmann Publishers, 2000.

[2] Dan Brickley and R. V. Guha. W3C Resource Description Framework (RDF) Schema Specification. http://www.w3.org/TR/1998/WD-rdf-schema/, March 2000. W3C Candidate Recommendation.

[10]http://kaon.aifb.uni-karlsruhe.de/

[3] Jeen Broekstra. *RQL Tutoria: Babysteps in Sesame RQL*. Aidministrator Nederland, September 2001. Version 0.5.

[4] James Clark and Steve deRose. XML Path Language (XPath), version 1.0. Technical report, W3C, November 1999. W3C Recommendation.

[5] IEEE Learning Technology Standards Committee. IEEE LOM Working Draft 6.1. http://ltsc.ieee.org/wg12/index.html, April 2001.

[6] dbXML Group. dbxml native database. http://www.dbXML.org/.

[7] Hadhami Dhraief, Wolfgang Nejdl, Boris Wolf, and Martin Wolpers. Open learning repositories and metadata modeling. In *International Semantic Web Working Symposium (SWWS)*, Stanford, CA, July 2001.

[8] Rael Dornfest and Dan Brickley. The power of metadata. http://www.openp2p.com/pub/a/p2p/2001/01/18/metadata.html, January 2001. excerpted from the book ”Peer-to-Peer: Harnessing the Power of Disruptive Technologies.

[9] The Edutella Project. http://edutella.jxta.org/.

[10] Li Gong. Project JXTA: A technology overview. Technical report, SUN Microsystems, April 2001. http://www.jxta.org/project/www/docs/TechOverview.pdf.

[11] Siegfried Handschuh and Steffen Staab. Authoring and annotation of web pages in cream. In *Proceedings of WWW-2002*. ACM Press, 2002.

[12] Pat Hayes. Rdf model theory. Technical report, W3C Working Draft, September 2001.

[13] IMS Global Learning Consortium Inc. IMS Learning Resource Metadata Specification v1.2.1. http://www.imsproject.org/metadata/index.html.

[14] M. Jarke, R. Gallersdörfer, M. Jeusfeld, M. Staudt, and S. Eherer. ConceptBase - a deductive object base for meta data management. *Journal on Intelligent Information Systems*, 4(2):167 – 192, 1995.

[15] V. Josifovski, T. Katchaounov, and T. Risch. Optimizing queries in distributed and composable mediators. In *4th Conference on Cooperative Information Systems, CoopIS'99*, pages 435 – 446, Edinburgh, Scotland, September 1999. http://www.dis.uu.se/ udbl/publ/coopis99.pdf.

[16] V. Josifovski and T. Risch. Functional query optimization over object-oriented views for data integration. *Journal of Intelligent Information Systems (JIIS)*, 12(2-3):165 – 190, 1999.

[17] V. Josifovski and T. Risch. Integrating heterogeneous overlapping databases through object-oriented transformations. In *25th Conf. on Very Large Databases (VLDB'99)*, pages 435 – 446, Edinburgh, Scotland, September 1999. http://www.dis.uu.se/ udbl/publ/vldb99.pdf.

[18] Project JXTA Homepage. http://www.jxta.org/.

[19] G. Karvounarakis, V. Christophides, D. Plexousakis, and S. Alexaki. Querying community web portals, 2001. Available at http://www.ics.forth.gr/proj/isst/RDF/RQL/, 2001. Submitted for publication.

[20] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[21] Stefan Kokkelink and Roland Schwänzl. *Expressing Qualified Dublin Core in RDF/XML*. DCMI, August 2001. http://dublincore.org/documents/2001/08/29/dcq-rdf-xml/.

[22] Ora Lassila and Ralph R. Swick. W3C Resource Description framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/REC-rdf-syntax/, February 1999. W3C Recommendation.

[23] W. Litwin and T. Risch. Main memory oriented optimization of oo queries using typed datalog with foreign predicates. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):517 – 528, 1992.

[24] J. W. Lloyd and R. W. Topor. Making prolog more expressive. *Journal of Logic Programming*, 3:225–240, 1984.

[25] A. Mädche, S. Staab, R. Studer, Y. Sure, and R. Volz. Seal - tying up information integration and web site management by ontologies. *IEEE Data Engineering Bulletin*, March 2002. http://www.aifb.uni-karlsruhe.de/ sst/Research/Publications/data-engineering-bulletin2002.pdf.

[26] D. Maier. A logic for objects. In *Workshop on Foundations of Deductive Databases and Logic Programming*, pages 6–26, 1986.

[27] Brian McBride. Jena: Implementing the rdf model and syntax specification. Technical report, Hewlett Packard Laboratories, Bristol, UK, 2000. http://www.hpl.hp.com/semweb/index.html.

[28] Sergey Melnik. *RDF API Draft*, January 2001. http://www-db.stanford.edu/ melnik/rdf/api.html.

[29] Wolfgang Nejdl, Hadhami Dhraief, and Martin Wolpers. O-telos-rdf: A resource description format with enhanced meta-modeling functionalities based on o-telos. In *Workshop on Knowledge Markup and Semantic Annotation at the First International Conference on Knowledge Capture (K-CAP'2001)*, Victoria, BC, Canada, October 2001.

[30] Mikael Nilsson and Matthias Palmér. Conzilla - towards a concept browser. Technical Report CID-53, TRITA-NA-D9911, Department of Numerical Analysis and Computing Science, KTH, Stockholm, 1999. http://kmr.nada.kth.se/papers/ConceptualBrowsing/cid_53.pdf.

[31] T.C. Przymusinski. Every logic program has a natural stratification and an iterated least fixed point model. In *Proceedings of the ACM Symposium on Principle of Database Systems (PODS)*, pages 11–21, 1989.

[32] Changtao Qu and Wolfgang Nejdl. Towards interoperability and reusability of learning resources: A SCORM-conformant courseware for computer science education. Technical report, Learning Lab Lower Saxony, University of Hannover, October 2001.

[33] T. Risch and V. Josifovski. Distributed data integration by object-oriented mediator servers. *Concurrency and Computation: Practice and Experience*, 13(11):933 – 953, 2001.

[34] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database Systems Concepts*. McGraw-Hill Higher Education, 4 edition, 2001.

[35] Michael Sintek and Stefan Decker. TRIPLE—An RDF query, inference, and transformation language. In *Deductive Databases and Knowledge Management (DDLP'2001)*, October 2001. http://www.dfki.uni-kl.de/frodo/triple/TripleReport.pdf.

[36] SUN Microsystems. *JXTA v1.0 Protocols Specification*, 2001. http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html.

[37] ADL Technical Team. SCORM Specification v1.2. http://www.adlnet.org/Scorm/scorm.cfm, October 2001.

[38] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38 – 49, 1992.