Computing and Informatics, Vol. 31, 2012, 189–223

INTEGRATION OF HETEROGENEOUS DATA SOURCES IN AN ONTOLOGICAL KNOWLEDGE BASE

Antoni Myłka, Alina Myłka

AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics, IT and Electronics
Department of Computer Science
Al. A. Mickiewicza 30, 30-059 Kraków

Bartosz Kryza, Jacek Kitowski

AGH University of Science and Technology
Academic Computer Centre CYFRONET
Ul. Nawojki 11, 30-950 Kraków
e-mail: `bkryza@agh.edu.pl`

Communicated by Renata Słota

Abstract. In this paper we present X2R, a system for integrating heterogeneous data sources in an ontological knowledge base. The main goal of the system is to create a unified view of information stored in relational, XML and LDAP data sources within an organization, expressed in RDF using a common ontology and valid according to a prescribed set of integrity constraints. X2R supports a wide range of source schemas and target ontologies by allowing the user to define potentially complex transformations of data between the original data source and the unified knowledge base. A rich set of integrity constraint primitives has been provided to ensure the quality of the unified data set. They are also leveraged in a novel approach towards semantic optimization of SPARQL queries.

Keywords: Data management, data integration, semantics, ontologies, conversion, RDF, integrity constraints, semantic optimization, SPARQL

Mathematics Subject Classification 2000: 68P15, 68T30, 68U35

1 INTRODUCTION

We would like to present X2R, a system that embodies the semantic approach towards the data integration problem. Its aim is to allow the user to get a view on all the data without having to care about the heterogeneity of the underlying data sources. It approaches the problem in two stages. First the ideas in the problem domain need to be captured in common ontologies – formal specifications of concepts [19]. When the common ontologies are ready X2R can be used to extract the data from the source databases, convert it to sets of RDF triples and store them in a single RDF database exposing them to user queries.

The architecture of X2R stemmed from the requirement for flexibility. The tool had to be able to work with a given set of source database schemas and a given ontology. Transformation of data between source schemas and target ontologies was therefore a core problem to solve. Additional complexity arose from the need to support three basic kinds of databases: relational databases, XML databases and directory services accessible with the LDAP protocol.

The second goal of the system was to provide the user with tools that would ensure the integrity of the unified data set. All three kinds of source databases provide a rich set of possible integrity constraints. This aspect has been neglected in the context of semantic databases, with only a handful of works available (e.g. [24]). We wanted to explore this idea and test its viability in practice. Integrity constraints also opened new exciting avenues for SPARQL query optimization, which had hitherto been purely theoretical [36, Section 5].

X2R combines multiple earlier research ideas in a single system. Section 2 contains a brief overview of the related work in two core aspects of X2R: conversion of data from various sources to RDF, and the usage of constraints to enforce the integrity of RDF graphs and optimize SPARQL queries.

Later, in Section 3.1 we outline the X2R approach towards data conversion, how we combine D2RQ [11], a well-known and freely available relational-to-RDF converter, with our own solutions for XML and LDAP. The subsequent sections describe our proposal for a set of integrity constraint types for RDF, which extends the one from [24] (Section 4), and our implementation of a semantic optimizer for SPARQL queries described in theory in [36] (Section 5).

A set of experiments outlined in Section 6 show that the X2R converters are capable of working with large amounts of data using limited memory. They also give arguments in support for our claim that integrity constraints are potentially useful in RDF and show which problems have to be solved before the constraint validation and semantic optimization of SPARQL queries can be made applicable on an industrial scale.

Section 7 gathers the conclusions and gives pointers for future work.

All examples of RDF data in this paper use the Turtle syntax, recently submitted for standardization to the World Wide Web Consortium [5]. All the declarations used by the RDF listings in this paper are summarized in Listing 1 and ommitted from the examples themselves to aid readability.

```
@prefix ont: <http://example.org/ontology#> .
@prefix data: <http://example.org/data#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix xml2r: <http://fivo.cyf-kr.edu.pl/xmlmapping#> .
@prefix ldap2r: <http://fivo.cyf-kr.edu.pl/ldapmapping#> .
@prefix rmap: <http://cms.org/mapping/rdbms#> .
@prefix xmlmap: <http://cms.org/mapping/xml#> .
@prefix ldapmap: <http://cms.org/mapping/ldap#> .
@prefix cmsont: <http://cms.org/ontology#> .
```

Listing 1. Turtle namespace declarations

2 RELATED WORK

Ontology-based data integration is a part of the broad research field of data integration [25]. In the last decade much of the activity in this area has been centered around the set of technologies developed within the Worldwide Web Consortium Semantic Web Activity: the Resource Description Framework (RDF, [21]), the Web Ontology Language (OWL, [41]) and the SPARQL query language for RDF [30]. These standards form the basic technological framework to build systems which make use of semantically integrated data, with the ultimate goal to turn the Web itself into "Semantic Web" [9], where all information is easily available in a machine-processable form. These technologies are also the foundation of X2R.

Our tool provides solutions to two problems, inherent in any system which aims to realize the added value of an integrated dataset expressed in RDF with a common ontology: the conversion of data, and the maintenance of data integrity. This section gives an overview of the state of the art in both areas.

2.1 Conversion of Data to RDF

The need to convert existing data to RDF has been recognized early by the researchers in the Semantic Web community. Given the multitude of legacy data formats numerous tools have been proposed which make the information available in the form of RDF triples. In subsequent sections we present an overview of the ones most relevant to X2R, i.e. those which deal with relational, XML and LDAP data models.

We would like to note though, that to our knowledge no solution combines the capabilities to convert data from all three data models in a single system.

2.1.1 Relational Data

The relational databases are the cornerstone of most computer systems in operation today. Tim Berners-Lee wrote about the need to expose information from relational databases in RDF triples as early as in 1998 [7]. Many solutions have appeared since then. A recent, detailed survey can be found in [33]. The most relevant to this paper are the ones which provide a customizable mapping between the relational schema and the ontology.

METAMorphoses [40] has been designed to allow for a single portion of data from a relational database to be formatted both as a human-readable HTML page and as a machine-readable RDF document. As such it was confined to small portions of data and could not process large datasets in their entirety which made it impossible to use in our scenario. The mapping languages of SquirrelRDF [37] and Triplify [3] were not expressive enough and we did not need query rewriting since X2R follows the idea of materialized integration. Virtuoso Open Source Edition[1] from Openlink Software was designed as database server, not a data converter. Moreover it was written in C++. Both of these facts made it very difficult to integrate it in X2R. A proprietary solution – Anzo Data Collaboration Server[2] from Cambridge Semantics – was interesting since it was the only solution which allowed for integration of data from relational and LDAP databases. At the time of writing it did not support XML databases, though its modular structure advertised on the website allowed for such a possibility in future. As such the functionality of this solution had the most overlap with that of X2R, yet it was proprietary and we wanted to make our tool freely available.

We settled on D2RQ [11], a relational-to-RDF conversion tool developed within the team of Christian Bizer at the Free University of Berlin. The mapping language of D2RQ [10] was the most expressive one among the freely available tools. Moreover it was written in Java – our language of choice – and exposed all of its functionality as a single Java class – `D2RQModel`, thus making it trivial for us to build our conversion component as a thin wrapper around the `D2RQModel` class and integrate it within the X2R architecture.

2.1.2 XML Data

Conversion of XML data to RDF has also been a popular research topic. Since the most popular serialization of RDF – RDF/XML – uses the XML syntax, both models are often confused, even though they are fundamentally different. XML is a tree, while RDF is a graph in a general sense. The discussion of differences between XML and RDF is almost as old as RDF itself, dating as far as 1998 [8].

Initially we tried to find a solution similar to D2RQ, which we could integrate in X2R. None of the solutions we found met our requirements. Some of them were difficult or impossible to customize, the user could not easily alter the structure of the target data to make it conform to a predefined ontology. These included Krextor from DFKI in Bremen [23], based on XSLT, Gloze from HP Labs in Bristol [4] and the tools from the Spanish University of Lleida – XML2RDF and XSD2OWL [18].

[1] `http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/`
[2] `http://www.cambridgesemantics.com`

We also found two customizable tools, each providing its own mapping language: XML2OWL from the University of Aachen [42]; while promising and functional, it was written in C++, which made it very difficult to integrate with our system, and JXML2OWL from the University of Madeira [32] which provided an easy to use GUI tool to create mappings but its expressivity was severely limited; the most important drawback was its inability to generate resources with custom URIs, which made any integration very difficult.

The most interesting solution was the WEESA framework developed by Gerald Reif within the context of his PhD thesis at the Technical University of Vienna. The functionality described in [31] and [17] looked very promising. WEESA did not make any assumptions on the source XML schema, or the target ontology. It provided an expressive mapping language (based on XML) for defining sophisticated transformations between the source XML and target RDF data. Thus it gave the user considerable freedom to work with arbitrary XML schemas and ontologies, fulfilling the core goal of X2R. Unfortunately it relied on the entire dataset being available in memory. It was not designed for streaming operation. Its core use case was very similar to the one of METAmorphoses, mentioned in Section 2.1.1, to generate both an HTML page for human consumption and an RDF description for automated agents from a single, small piece of source XML data.

Therefore, an entirely new conversion-engine had to be developed for X2R.

2.1.3 LDAP Data

In comparison with relational and XML models, the LDAP data model did not enjoy as much popularity among RDF enthusiasts in recent years. The Sören Auer team at the University of Leipzig have proposed LDAP2OWL [15], a set of two tools, one converting an LDAP Schema to an OWL ontology and another one converting LDAP data to RDF which conforms to the said ontology. The mapping is not customizable though, therefore unusable in X2R. Two tools already mentioned in Section 2.1.1 are also relevant here. SquirellRDF provides the ability to issue very simple SPARQL queries against an LDAP directory, unfortunately too simple to be usable in X2R. Anzo Data Collaboration Server also advertises its LDAP-to-RDF conversion functionality.

It is interesting to note that an opposite problem, namely the problem of accessing RDF data stores with the LDAP protocol has also been explored with an LDAP 2 SPARQL tool described in [16], even though the RDF data model is more generic.

Having dismissed all the freely available tools we decided to implement our own tool for X2R.

2.2 RDF Data Integrity

RDF data integrity has not been receiving the level of attention comparable to the integrity of data in other data models. RDF has been developed as a technology to support semantic interoperability on the web scale (Semantic Web). One of the core concepts in Semantic Web is the so called Open World Assumption, which states that lack of information does not imply negation; in other words if a statement is not explicitly stated to be true, it cannot be considered false.

This assumption has found its way into the definitions of the inference semantics of ontology languages such as RDFS [12] or OWL [41]. Most of the constructs in ontology definitions are not interpreted as constraints on the data, but as premises to infer additional facts from. This may be counterintuitive. We could present it on a simple example.

Let us consider a very simple RDFS ontology consisting of three classes: Prince, Princess and Frog. We also define one property "kissed". We would like to express the assumption that a prince can only kiss a princess, therefore we define the domain and range of the "kissed" property to be Prince and Princess, respectively. A formal definition is given below:

```
ont:Prince a rdfs:Class .
ont:Princess a rdfs:Class .
ont:Frog a rdfs:Class .
ont:kissed a rdf:Property ;
  rdfs:domain ont:Prince ;
  rdfs:range ont:Princess .
```

Listing 2. Prince Princes and Frog ontology

Having defined this ontology, let us consider a following piece of RDF data.

```
data:John a ont:Prince .
data:Jane a ont:Frog .
data:John ont:kissed data:Jane .
```

Listing 3. John and Jane data

The data states that there exists a prince and a frog. Moreover it asserts that the prince has kissed the frog. We have said before that a prince can only kiss a princess. Is this a constraint violation? No it is not. The definition of the RDFS semantics contains a following rule ("rdfs3" in [12, Section 7.3]), which is an exact consequence of the open world assumption: "if a graph contains two triples: $(?p, \texttt{rdfs:range}, ?C)$ and $(?x, ?p, ?y)$ then add a following triple to the graph: $(?y, \texttt{rdf:type}, ?C)$".

When we combine the ontology and the data in a single graph and apply the "rdfs3" inference rule, we will infer a triple which says:

```
data:Jane a ont:Princess .
```

Listing 4. An inferred triple

Even though the fact that `data:Jane` is a princess is not explicitly stated anywhere, we cannot assume that it is false. The ontology says that a prince can only kiss a princess, so if he kissed a frog, this means that the frog must be a princess at the same time. The inference engine will infer this fact for us.

This phenomenon shows that ontologies do not actually define the structure of RDF data the way schemas in other data models do. They define the inference semantics of data. They do not say which RDF graphs are compliant or not compliant. They say what additional facts can be inferred from a given RDF graph.

This is why the proposals for integrity constraints in RDF appeared outside the main standardization process in the World Wide Web Consortium. The SPARQL Inferencing Notation (SPIN – [22]), allows the user to express constraints as arbitrary SPARQL ASK queries, without classifying them or putting them in any formal framework.

The work done within Clark and Parsia LLC, concentrates on defining an alternative OWL inference semantics, which would allow OWL itself to be used as a constraint definition language. It has been described by Sirin et al. in [38] and later implemented in their product Pellet Constraint Validator which interprets their OWL constraints and checks them with SPARQL queries.

The third approach, the one most relevant to this work has been proposed by Lausen, Meier and Schmidt in [24]. It is similar to the two above in that it uses SPARQL to validate constraints, yet it chooses to build the constraint primitive classification on a different formal framework than so called embedded dependencies known from the relational database literature (e.g. [1, part C]). As such, the constraints are defined at the data level without taking the inference semantics into account. In this approach an RDF graph is just a set of triples which either conforms to a set of constraints or not. The use of an inferencer may add additional triples to the graph which in turn may fix a constraint violation or introduce a new one, but the constraint validator does not need to be aware of the inference semantics of the ontology language. This simplifies the implementation and allows such a validator to be used in cases when no inference is used at all, which is especially important when large datasets are used, as large-scale inference is usually very costly.

Moreover, specifying constraints as embedded dependencies allows us to reuse another technique from the relational database literature – the Chase algorithm [1, Section 8.4], which was proven in [36] to be applicable to constraint-based semantic optimization of SPARQL queries. Sections 4 and 5 of this paper provide a more detailed overview.

It is worth noting that even though the field of RDF data integrity did not appear popular among researchers in the beginning of the Semantic Web movement it is now becoming ever more important, with motivation from commercial sector (as is the case in SPIN) and funding from public funding agencies like NIST (work of Evren Sirin at Clark and Parsia LLC) or DFG (the CORSOS project at the University of Freiburg, in the group of professor Lausen).

3 DATA INTEGRATION

In this section we would like to present the X2R approach towards the data integration problem, based on materialized conversion of source data to RDF triples configured via mappings. An important limitation is presented – in its basic form it lacks any tools for sophisticated data deduplication and fusion, which are nevertheless possible to implement on top of the current architecture. Later, an informal overview of all three mapping languages is presented, together with examples.

3.1 X2R Approach and Its Limitations

In the current X2R prototype the unified dataset emerges as a result of extracting the data from each data source, transforming it to RDF triples and loading it into a single RDF graph. The "unification" process is defined by the mappings for each data source. The mappings need to fulfil two tasks:

- Unify the vocabulary, i.e. make sure that the result RDF triples use the vocabulary from a common ontology and that the structure of data from all data sources conforms to the ontology
- Deduplicate the resources, i.e. make sure that if a single object from a real world is referred to in multiple data sources it is always assigned the same URI, so that two descriptions of a single resource from two data sources can be simply added to each other in the result RDF graph to form a single, richer description.

It is obvious that the deduplication approach outlined above is simplistic. During the conversion process, when a piece of data from a source database describing a real-world object is converted to RDF, a URI for that object is generated. This URI can then serve as the subject of the generated RDF triples. In the current prototype the URI generation can only take information from the source database into account though.

In order to highlight the implications of this limitation, we would like to present an example of an information integration scenario. We will also use it in sections on mapping languages. Let us consider a simple example of a content management system, which stores articles, user accounts and a log of all accesses of users to the articles. We would like to integrate data from a relational database, and LDAP server and an XML database. In the LDAP database we store information about the users of a system; a sample LDAP entry might look as follows:

```
dn: uid=mylka,ou=Students,dc=example,dc=org
uid: mylka
cn: Antoni Mylka
objectClass: account
objectClass: posixAccount
objectClass: top
uidNumber: 25088
```

Listing 5. LDAP data about a single user (LDIF)

The XML database would contain an XML document representing a list of articles, whose structure would be similar to:

```
<articles>
  <article id="6345">
    <topic>example topic</topic>
    <title>An example article/title>
    <content>
      The content of the article.
    </content>
  </article>
</articles>
```

Listing 6. XML document with a single article

The access log is stored in a relational database in a single table called "LOG" with four columns: "entry_id", "user_id", "article_id", "read_date". The information that Antoni Mylka accessed the example article No. 6345 on September 14th, 8:32 PM, would be expressed as a tuple:

```
----------------------------------------------------------------
| entry_id | user_id | article_id | read_date                  |
----------------------------------------------------------------
| 46234    | 25088   | 6345       | 2010-09-14 20:32           |
----------------------------------------------------------------
```

Listing 7. Log entry. User 25088 read article 6345. (relational database row)

In this example all three data sources use the same integer identifiers of articles and users. A description of a user generated from the LDAP entry could use a URI with the textual identifier as its part: `25088`. This identifier can be used as an object of an RDF triple which comprises the RDF version of the user description:

```
<http://cms.org/people/25088> cmsont:name "Antoni Mylka" .
```

Listing 8. User data from Listing 5 converted to RDF

The same can be said about articles in the XML database, each can be assigned a URI which contains the article identifier – the value of the "id" attribute of the "article" element, for instance for the article in Listing 6 it could be expressed in RDF as an URI: `http://cms.org/articles/6345`.

An RDF version of the XML article information (Listing 6) could then look like:

```
<http://cms.org/articles/6345>
    cmsont:topic "example topic" ;
    cmsont:title "An example article" ;
    cmsont:content "Example article content" .
```

Listing 9. Article information from Listing 6 converted to RDF

These identifiers can also be generated during the conversion of the relational database so that when the converter processes a single log entry; like the one from Listing 7 it is able to generate the following triples.

```
<http://cms.org/log/46234>
    cmsont:article <http://cms.org/articles/6345> ;
    cmsont:user <http://cms.org/users/25088> ;
    cmsont:readDate "2010-09-14T20:32"^^xsd:dateTime .
```

Listing 10. Log entry from Listing 7 converted to RDF

The RDF descriptions of the user, the article and the log entry can now be placed in a single RDF graph to enable queries like "Show me the topics of articles read by Antoni Mylka in the last month". This query encompasses information from all three databases. First the identifier of the user needs to be obtained on the basis of his name, then all the relevant log entries from the last month need to be gathered. The log entries are used to obtain identifiers of the articles, which allow us to obtain the topics.

In the integrated RDF dataset this can be done with a single SPARQL query. This will work though only because it was possible to generate the same URIs from all data sources. A system user could be assigned the same URI in the RDF triples generated from the LDAP server and the relational database because both data sources contained the same integer: 25088. A similar situation occurred in case of an article. The integer 6345 appeared in both data sources so during both conversion processes the article could be assigned the URI.

Such an assumption about the data sources allows us to reduce the problem of duplicate detection to appropriate URI generation, while the data fusion is performed simply by inserting the result of all conversion processes in a single RDF graph. In a general case though such a situation rarely occurs in practice. Both problems (duplicate detection and data fusion) are research fields in their own right and can boast their own extensive literature (see [25, Chapter 8]). They have been considered out of scope of the current X2R prototype, yet the X2R architecture allows for relatively easy implementation more sophisticated deduplication and fusion approaches on top of the output of the converters.

3.2 Relational Mapping Language

As already stated in Section 2.1.1 the relational mapping language in X2R is the mapping language of D2RQ. Its authoritative specification is given in [10]. We have used it as a point of reference for the design and evaluation of our own languages for XML and LDAP: both are defined as RDF vocabularies, both have constructs equivalent to `d2rq:ClassMap` and `d2rq:PropertyBridge` i.e. a construct that defines a set of resources and a construct which defines a predicate–object pair to be added to each resource to form a result RDF triple.

Readers unfamiliar with D2RQ mapping language are advised to acquaint themselves with [10] as we will refer to this work extensively in the following sections.

A mapping for the access log database from the example scenario (Section 3.1) would look as follows:

```
rmap:LOG a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:uriPattern "http://cms.org/log/@@LOG.id@@";
  d2rq:class cmsont:LogEntry .

rmap:LOG_UserID a d2rq:PropertyBridge;
  d2rq:belongsToClassMap rmap:LOG;
  d2rq:property cmsont:user;
  d2rq:uriPattern "http://cms.org/users/@@LOG.user_id@@" .

rmap:LOG_ArticleID a d2rq:PropertyBridge;
  d2rq:belongsToClassMap rmap:LOG;
  d2rq:property cmsont:article;
  d2rq:uriPattern "http://cms.org/articles/@@LOG.article_id@@" .

rmap:LOG_ReadDate a d2rq:PropertyBridge;
  d2rq:belongsToClassMap rmap:LOG;
  d2rq:property cmsont:readDate;
  d2rq:column "LOG.read_date" .
```

Listing 11. Relational conversion mapping which converts Listing 7 into Listing 10

3.3 XML Mapping Language

We would like to start the description by presenting an example mapping for the XML database in our test scenario from Section 3.1, Listing 6.

```
xmlmap:entryMap a xml2r:ClassMap;
    xml2r:nodeXPath "./articles/*" ;
    xml2r:uriPattern "http://cms.org/articles/${data(@id)} ;
    xml2r:class cmsont:Publication .

xmlmap:titleBridge a xml2r:PropertyBridge ;
    xml2r:belongsToProperty xmlmap:entryMap ;
    xml2r:property cmsont:title ;
    xml2r:pattern "${title/text()}" .

xmlmap:topicBridge a xml2r:PropertyBridge ;
    xml2r:belongsToProperty xmlmap:entryMap ;
    xml2r:property cmsont:topic ;
    xml2r:pattern "${topic/text()}" .

xmlmap:contentBridge a xml2r:PropertyBridge ;
    xml2r:belongsToProperty xmlmap:entrymap ;
    xml2r:property cmsont:content ;
    xml2r:pattern "${content/text()}" .
```

Listing 12. XML conversion mapping which converts Listing 6 into Listing 9

The core construct for the X2R XML mapping language is the `xml2r:ClassMap`, which is a counterpart of the `d2rq:ClassMap` element from d2rq (and <resource> element from WEESA). It is used for defining new resources, whose type, given by the `xml2r:class` property, is a class from the target ontology. RDF Resources are generated from a sequence of nodes from the source XML document. The sequence is defined with an XPath expression given with the property `xml2r:nodeXPath`. The basic idea of the conversion process is that for each resource in the sequence it is possible to get data via new XPath expressions relative to that resource. For instance, in the example scenario we define the `xml2r:nodeXPath` as `./articles/*`. In the document from Listing 6 this XPath expression will return a single XML node, the `<article>` element. Within the context of that element, the relative XPath query `data(@id)` will return the value of the "id" attribute of the `<article>` element which in this case is 6345.

This value will be used within the URI, as defined in `xml2r:uriPattern`. The URI pattern value may contain arbitrary relative XPath expressions, thus giving the user considerable freedom to generate URIs for resources.

Each resource may have an unlimited number of properties. They are described via `xml2r:PropertyBridge` which corresponds to the `d2rq:PropertyBridge` (or WEESA <triple>). Three most essential elements of a property definition:

- `xml2r:belongsToClassMap` – determines the resource to which this property should belong, this resource ought to be defined via `xml2r:ClassMap`, equivalent to `d2rq:belongsToClassMap`.
- `xml2r:property` – the URI of the predicate. It would usually denote a property from the target ontology, equivalent to `d2rq:property`.
- `xml2r:pattern` – describes how the value of the property should be generated. Like the `xml2r:uriPattern` it is built of an arbitrary sequence of characters and relative XPath expressions. Equivalent to `d2rq:pattern`. The difference is that the XML version contains relative XPath queries in the pattern, while the D2RQ version contains column names.

3.4 LDAP Mapping Language

The LDAP mapping language is less expressive than the other two mapping languages presented earlier in this chapter. This is related mostly to the nature of the LDAP data model and the limited expressivity of LDAP filters utilized for the retrieval of data from the LDAP server. An example is given below; it corresponds with the LDAP user directory from Listing 5.

```
ldapmap:Person a ldap2r:Resource ;
    ldap2r:query "(objectClass=posixAccount)" ;
    ldap2r:uriPattern "http://cms.org/people/${uidNumber}" .

ldapmap:PersonName a ldap2r:Triple ;
    ldap2r:subject ldapmap:Person ;
    ldap2r:predicate cmsont:name ;
    ldap2r:object [ a ldap2r:AttributeValue ;
                    ldap2r:attributeName "cn" ]
```

Listing 13. LDAP conversion mapping which converts Listing 5 into 8

The mapping is concentrated around the `ldap2r:Resource` that is the bridge between an RDF resource and an LDAP object. It maps certain LDAP entries retrieved from the LDAP server via a query specified in `ldap2r:query` to RDF resources given as a `ldap2r:uriPattern`. Having defined the subject we may attach to it predicates and objects via the `ldap2r:Triple` constructs.

Every node of type `ldap2r:Triple` may have the following predicates:

- `ldap2r:subject` – it corresponds to a node that should be already defined within the mapping, the type of this node must be `ldap2r:Resource`
- `ldap2r:predicate` – it defines new predicate that will be attached to the resource
- `ldap2r:object` – defines the object of the triple, it may be either an attribute value, taken individually from each entry, or a constant, common to all resources.

4 INTEGRITY CONSTRAINTS

The approach towards integrity constraints in RDF has been taken from the "SPARQLing Constraints in RDF" paper by Schmidt, Meier and Lausen [24]. It is based on the observation that a set of RDF triples could be treated as a single ternary relation. This allows us to use the terminology well-known from literature about relational databases [1] and apply it to RDF.

X2R contains an implementation of concepts presented in [24]. We would like to avoid repeating them here, so readers unfamiliar with conjunctive queries and embedded dependencies are advised to consult that paper and relevant sections from [1, part B].

4.1 Classification

Constraints in X2R (and in [24]) belong to two classes: equality-generating dependencies (EGD) and tuple-generating dependencies (TGD). An EGD states that if certain conditions are met, two pieces of data must be equal. For instance, the uniqueness constraint states that if two resources have a property with the same value, then these resources must be equal. A TGD on the other hand states that certain conditions imply the existence of other triples. For instance the domain constraint states that existence of a triple with a certain predicate might imply that the subject of that triple has a certain type.

From the constraints described in [24] we have implemented two: Totality and Uniqueness. We have also proposed four additional tuple-generating dependencies which were not present in [24]. They are all related to domains and ranges of predicates:

Range. The domain of a predicate is defined in RDFS with the `rdfs:range` construct. It states that the objects of all triples with that predicate have a specified type. In RDFS this is a premise for inference, as already outlined in the example in Section 2.2. We would like to specify this as a constraint. Notation below uses a as a shortcut for `rdf:type`. A constraint $\textsc{Range}(p, C_r)$ reads: all objects of triples with the predicate p have the type C_r.

$$\textsc{Range}(p, C_r) : \forall_{(x,y)}\ T(x, p, y) \Rightarrow T(y, a, C_r)$$

The above applies to all occurences of the predicate p. We have found that sometimes it is practical to limit the constraint only to subjects of a specific type (usually different from the range type). Therefore we have implemented constraint we termed "ClassDomain".

$$\textsc{ClassRange}(C_d, p, C_r) : \forall_{(x,y)}\ T(x, a, C_d) \wedge T(x, p, y) \Rightarrow T(y, a, C_r)$$

Domain. The two domain constraints are analogous to the range constraints, but they enforce the type of the subjects of triples. $\textsc{Domain}(C_d, p)$ reads: all subjects of triples with the predicate p have the type C_r.

$$\begin{aligned} \textsc{Domain}(C_d, p) : &\quad \forall_{(x,y)}\ T(x, p, y) \Rightarrow T(x, a, C_d) \\ \textsc{ClassDomain}(C_d, p, C_r) : &\quad \forall_{(x,y)}\ T(y, a, C_r) \wedge T(x, p, y) \Rightarrow T(x, a, C_d) \end{aligned}$$

4.2 Representation in RDF

All the constraints defined in this chapter are stored in the RDF database. Their RDF representation uses types and predicates from a constraint ontology created by us. It differs from the one described in [24].

Firstly, we introduce a class `x2rc:Constraint` denoting a constraint and two properties that are needed for connecting constraints with the target ontology (target ontology is a final ontology for integrated data):

- `x2rc:hasConstraint` – this property attaches a particular constraint to a class from the target ontolgoy
- `x2rc:hasPropertyConstraint` – this property attaches a particular constraint to a property from the target ontology.

Each constraint type (from those described in Section 4.1) we define a subclass of `x2rc:Constraint`. The classification includes class constraints like `x2rc:Totality` and `x2rc:Unique` which are to be attached to a class from target ontology via the `x2rc:hasConstraint` predicate. Property-related constraints, like `x2rc:Domain`, `x2rc:Range`, `x2rc:ClassDomain` or `x2rc:ClassRange`, are attached to a property from the target ontology with `x2rc:hasPropertyConstraint`.

4.3 Validation

The constraint validation process has been inspired by the one from [24]. We made an important change. Instead of `ASK` queries we use `CONSTRUCT` queries. They create instances of subclasses of `x2rc:ConstraintViolation`. Thus as a result of the validation process we get detailed information about all resources that violate constraints. In the original approach from the paper, we would only get information whether any constraint violations exist or not. For instance, a validation query for a DOMAIN(`ont:Class`, `ont:property`) consraint, identified in the RDF database via a `data:DomainConstraint` URI might look like this:

```
PREFIX x2rr: <http://agh.edu.pl/2009/06/x2r-reports#>
PREFIX ont: <http://example.org/ontology#>
PREFIX data: <http://example.org/data#>
CONSTRUCT { _:b a x2rr:ClassDomainConstraintViolation .
            _:b x2rr:offendingInstance ?x .
            _:b x2rr:violatedConstraint
                      data:DomainConstraint . }
WHERE {
  ?x ont:property ?y .
  OPTIONAL {
    ?z ont:property ?y .
    ?z a ont:Class .
    FILTER(?x = ?z)
  } FILTER (!bound(?z))
}
```

Listing 14. Domain constraint validation query

5 QUERY OPTIMIZATION

Optimization process in X2R is an implementation of the algorithms described in [34, Chapter 5]. The optimizer is initialized with all constraints known to be valid for the current state of the RDF database. Then, the tree-based representation of the query can be rewritten. The rewriting process is implemented with the aid of the Visitor pattern. There are multiple visitors necessary to conduct the optimization. Each optimizer traverses the query tree and performs a single well-defined transformation. The optimization is implemented as a sequence of such transformations.

- `BGPOptmizer` – Implements the process described in [34, Section 5.4.3]. It traverses the query tree and finds lists of statements that are connected via the join operation. Each such list of statements is a Basic Graph Pattern (BGP) and may be optimized by the `X2RChaseAndBackchaseEngine`.

  The `BGPOptimizer` converts each BGP into a list of simpler `TriplePattern` instances, and passes it to `X2RChaseAndBackchaseEngine`. When an optimized list of `TriplePatterns` is returned, it is converted back to the SPARQL algebra format and reattached to the query tree. This process is an exact implementation of the AND-only optimization algorithm described in [34, Section 5.4.3].

- `FilterPushingOptimizer` – it implements so-called filter pushing – an algebraic equivalence in the SPARQL algebra (Group V on Table 4.1 [34, Chapter 4, p. 110]) and two rules related to the semantic optimization. Under certain conditions it is possible to remove the filter or remove the variable from the projection (see Lemma 5.4 within [34, Section 5.4.4])
- `OPTOptimizer` implements the optimization rule for the OPTIONAL (OSI and OSII defined on p. 142 of [34]). Under certain circumstances it is possible to substitute the OPTIONAL operator with a normal join, which could potentially drastically reduce the complexity of a query.
- `ProjectionPusher` implements the algebraic projection pushing rules [34, Section 4.2.4]. We found them necessary to ensure that the BGPOptimizer brings maximal results.

`X2RChaseAndBackchaseEngine` is the heart of the entire optimization. It is an implementation of the Chase and Backchase algorithm, first described in [6, 28]. This algorithm gets a conjunctive query and a set of constraints as input, and outputs a potentially shorter query, equivalent under the constraints. Its application to the optimization of SPARQL queries is described in [34, Section 5.4.3].

We used MARS [14], a tool developed by the pioneers of constraint-based optimization from the team of Alin Deutsch. It has been developed to explore ways to combine data from relational and XML storage (hence the name: Mixed and Redundant Storage). We have been able to work with it due to courtesy of Nicola Onose from University of California in San Diego, who provided us with the source code.

The `X2RMarsChaseAndBackchaseEngine` is a thin wrapper around MARS, which exposes the subset of its functionality as an interface that is easy to use other parts of the optimization subsystem. The interface (`ChaseAndBackchaseEngine`) has been separated from the implementation which makes it possible to swap the implementation should a better one become available.

6 EVALUATION

6.1 Introduction

When we compare X2R with other existing solutions in the realm of semantic information integration, it shows two important advantages.

The first one lies in the fact that it provides a unified framework for integrating data from database utilizing three different data models. Each individual model has been a subject of broad research and numerous integration solutions exist that deal with each model separately (some examples have been presented in Section 2). Generic frameworks have also been proposed which allow such a solution to be built on top of some basic infrastructure (e.g. as a Service-Oriented Architecture). Still to our knowledge no single solution exists that would provide a relatively simple way to integrate data from relational, XML and LDAP databases.

The second advantage lies in integrity constraints. X2R provides one of the first usable implementations of integrity constraints for RDF data. It is possible to express most of the constraints from source models in RDF model by using built-in RDF constraint types. Not only can they be used for mapping constraints retrieved directly from each data source but also to express business rules that span data from multiple data sources. When the constraints hold, they can be used to remove "obvious" parts from SPARQL queries which in certain cases can make them run faster.

Having gone through a description of the system implementation and the ideas behind it we would like to present the results of experiments that show how our prototype performs in practice. They are divided into two parts. The first part (Section 6.2) is devoted to the conversion components – the Data Source Adapters. The second part (Section 6.3) covers the validation of integrity constraints and the semantic optimization of queries.

We have expressed the goals of the system as a series of simple claims and designed the experiments to validate those claims under realistic workloads, with large datasets. The rest of this section presents our claims in more detail. The following sections describe the experiments, present the results, and provide a commentary.

6.1.1 Conversion Hypotheses

The tests of the RDF conversion process have been designed to confirm three hypotheses.

- H1.1 It is possible to convert a non-trivial, realistic dataset to RDF using a custom mapping.
- H1.2 The conversion supports relational, XML and LDAP databases.
- H1.3 It is possible to perform the conversion if the size of the source or the result dataset is larger than the size of the available random access memory.

6.1.2 Validation and Optimization Hypotheses and Assumptions

In the second part (Section 6.3), devoted to the integrity constraints we used the well-known LUBM benchmark [20]. It proved possible to define 32 constraints on the dataset generated by the LUBM generator. Using those constraints our optimizer could minimize 8 out of 14 queries. We tested the time it took to answer both the optimized and unoptimized ones on datasets of various sizes. Moreover we also measured the time of validation. The validation is necessary to ensure that the optimized queries are equivalent to the unoptimized ones. Those measurements were conducted twice for two sets of constraints: the full one with 32 constraints and a limited one with only nine constraints. The limited set of constraints contains only those which are required for LUBM queries optimization, so that the result queries are the same but the optimization is less time-consuming.

In measuring the query performance we distinguish between the evaluation time and the total time. The evaluation time does not include the optimization, while the total time does (details in Section 6.3.3). Three assumptions and two hypotheses were constructed for the validation and optimization tests.

The assumptions pertain to five important properties of the optimization process: the dataset size, the constraint set size, the total time, the evaluation time and the effect of optimization on the query, i.e. if the query stays the same after the optimization or if it is changed.

A2.1 If the optimization does not change the query and the dataset is fixed then the evaluation time does not change between "optimized" and "unoptimized" queries (because they are the same) regardless of the number of constraints involved (because the optimization time is not included in the evaluation time).

A2.2 If the query is minimized by the optimization process with some constraint set, then adding new constraints which do not influence the result of optimization for that query does not change the evaluation time (because no further improvements in the query are made, and the optimization time is not included in the evaluation time).

A2.3 The cost of the optimization usually rises with the number of constraints involved. Therefore, if the result of the query optimization remains the same and adding new constraints does not change it, then the total time rises with the constraint set size. (For example, in our experiment the total time with 32 constraints is longer than the total time with 9 constraints because we know that the queries are optimized in exactly the same way.)

The hypotheses that we wanted to explore are fundamental to the viability of the semantic optimization approach. They are summarized below.

H2.1 If the optimization engine finds a shorter equivalent query, then the evaluation time of the rewritten query is faster than the original one.

H2.2 For each query there should exist a dataset size above which the semantic optimization is beneficial. On smaller datasets the overhead incurred by the semantic optimization exceeds the benefits of shorter evaluation time. For larger datasets though, as the overhead does not depend on the dataset size, while the evaluation time does, the gain in evaluation time outweighs the costs of semantic optimization.

6.2 Conversion

The aim of the conversion experiments was to gain an overview of the capabilities of all three data source adapters in a realistic usage scenario. The remaining subsections of this section describe all aspects of the experiment in more detail: the hardware used, the source data, mappings, result data and measurement results. The section concludes with a brief summary of all findings.

6.2.1 Hardware

The experiments have been conducted on two desktop computers connected with a 100 MB Ethernet network via a D-Link DES 1005-D switch. The first machine (henceforth called the "source" machine) housed the databases with the source data. It was a an AMD64 2 800+ machine with 512 megabytes of RAM running under Ubuntu 9.10 Server. The second machine ("target" machine) ran the Java programs that initiated the conversion process and passed the results to the target RDF storage. The target machine was a little more powerful, boasting an AMD64 3 500+ CPU, 2 gigabytes of RAM. It ran under the same Ubuntu 9.10.

6.2.2 Source Data

The source data has been adapted from the DBLP dataset. DBLP is a well-known Computer Science bibliography portal operated by Michael Ley from the University of Trier in Germany[3]. The technical, scientific and social aspects of operating DBLP have been described in numerous publications, most recently in [26] and [27].

We chose DBLP because the data is available free of charge. Moreover it is widely used, well-known, actively maintained and pretty large (the DBLP website makes it available as XML files of about 500MB).

The data that powers DBLP is made available for download as a single large XML file. We used this file to test the XML Data Source Adapter. Sedna, an XML database, served as the XML storage implementation. The FacetedDBLP portal uses a modified version of the DBLP data. It is prepared by the portal maintainers. From our point of view, the most interesting fact is that they make the data available as an SQL file ready to be imported into a MySQL database – thus a perfect testbed for our Relational Data Source Adapter.

The LDAP Data Source adapter has been tested with the DBLP XML dataset modified for import into an OpenLDAP server by a Ruby script written by ourselves.

The script converted the XML file into a LDIF file, which could be imported into OpenLDAP, a fine open-source LDAP server implementation.

6.2.3 Mappings

The test mapping for the relational adapter has been taken directly from the Faceted-DBLP website[4]. Two remaining ones (for XML and LDAP) were manually crafted by ourselves, derived from the relational mapping.

The adaptation process gave us a detailed comparison between the expressivity of the mapping languages created for our prototype converters and the expressivity of the mapping language for the well-established RDBMS-to-RDF mapping tool –

[3] `http://www.informatik.uni-trier.de/~ley/db/index.html`, retrieved on 11.12.2009

[4] `http://dblp.l3s.de/fdblp-mapping.n3`, retrieved on 14.12.2009

D2RQ. This exposed the limitations of our prototypes. We consider them to be very interesting because they may be treated as pointers for further work.

6.2.4 Target Ontology

The experiment does not use any fixed target ontology. It follows the example of the FacetedDBLP mapping and uses a wide array of classes and properties from many ontologies. The most important ones come from SWRC – Semantic Web for Research Communities. It is a bibliography-oriented ontology developed by researchers from Forschungszentrum Informatik in Karlsruhe, Germany [39]. SWRC is combined with relevant predicates from Dublin Core[5] and FOAF[6].

6.2.5 Target Database

In our experiments, the result of the conversion was loaded into NativeStore – a disk-based RDF storage component from the Sesame Framework [13, 2] version 2.3.2.

6.2.6 Results

	RDBMS	XML	LDAP
Source text file size [kB]	597 494 704	676 753 072	719 060 701
Database vendor	MySQL	Sedna	OpenLDAP
Internal database file size [kB]	1 066 767	3 074 374	2 415 300
Conversion time [h:m:s]	3:21:30	1:53:53	0:42:36
Resulting statement count	46 828 276	23 751 240	17 604 581
Resulting NativeStore size [kB]	3 084 448	2 738 536	1 848 176
Average Load speed [st/s]	4 040	3 760	7 387
Peak momentary speed [st/s]	36 114	5 575	11 401
JVM Heap Size [MB]	1 024	1 024	1 024

Table 1

6.3 Integrity Constraints

The aim of the experiments for the X2R integrity constraints is to test their applicability and performance in two important areas of functionality – validation and query optimization. The next section describes the Lehigh University BenchMark (LUBM) – a benchmarking tool, well established in the Semantic Web community. It serves as the foundation for our tests, which are described in the following sections, the first one devoted to constraint validation (Section 6.3.2), the second one to query optimization (Section 6.3.3). All tests were performed on the "target" machine (described in Section 6.2.1).

[5] `http://dublincore.org`, retrieved on 16.12.2009

[6] `http://www.foaf-project.org`, retrieved on 16.12.2009

6.3.1 Lehigh University BenchMark (LUBM)

The main goal of LUBM is to provide a way to test the performance of RDF databases in a standard and systematic way. It has been widely adopted as a de facto standard benchmarking tool for RDF stores. Many RDF database vendors publish the results of LUBM tests. An overview can be found on the World-Wide-Web Consortium website[7]. Vendors compete in load speed, query answering speed, as well as scalability, i.e. the ability to store really large datasets. The benchmark itself is described in detail in [20].

The benchmark consists of a set of 14 queries, which are to be evaluated over a dataset generated by the supplied generator. The dataset conforms to a realistic OWL ontology which describes universities, students, faculty and courses. The generator can repeatably generate data for an arbitrary number of fictitious universities. This process is repeatable which guarantees that the results are kept comparable.

The generated data conforms to a set of basic rules, outlined in plain English prose. They are given on the website [8]. We used this description to create definitions of constraints. For instance if a description says that "every Student is memberOf a Department", we can add a totality constraint on the Student class and the memberOf property. Moreover we can also add a ClassRange constraint, which would say that whenever a resource has a Student type, and there exists a triple with that resource as subject and memberOf as predicate, then the object of that triple must also have a type of Department.

The data generated by the LUBM generator has been processed to materialize all triples inferred by an OWL inferencer (SwiftOWLIM[9]), so that the queries can take them into account. All 32 constraints we created have been validated to confirm that they hold.

Queries. All 14 LUBM queries are simple. They consist of a single Basic Graph Pattern without any FILTER, UNION, OPTIONAL, ORDER BY or any other SPARQL constructs. Correct answers do require the presence of inferred triples though, as explained above. What we wanted to stress though is that the simplicity of those queries does not showcase the full capabilities of the query optimizer. More specifically only the algorithm described in [34, Section 5.4.3], coupled with the algebraic projection pushing equivalences [34, Section 4.2.4] is actually used in the tests. More advanced, SPARQL-specific optimization rules are not covered by the LUBM tests. Proper tests of their actual performance remains a point for future work.

[7] `http://esw.w3.org/topic/LargeTripleStores` retrieved on 24.01.2010

[8] `http://swat.cse.lehigh.edu/projects/lubm/profile.htm` retrieved on 24.01.2010

[9] `http://www.ontotext.com/owlim/`

6.3.2 Constraint Validation Tests

The validation entails evaluating SPARQL queries over the dataset. Each constraint yields a single SPARQL query, the cost of which is dependent on the dataset size. Thus, the total cost of validation rises with the number of constraints and the dataset size.

Two constraint validation experiments were conducted, the first one involved a full set of 32 constraints. The second experiment was done with a smaller subset of 9, limited only to those constraints necessary to perform all the possible optimization on LUBM queries. The limited set of constraints were:

- Range(`ub:undergraduateDegreeFrom`, `ub:University`) – query 2.
- ClassRange(`ub:GraduateStudent`, `ub:memberOf`, `ub:Department`) – query 2.
- Domain(`ub:Publication`, `ub:publicationAuthor`) – query 3.
- Domain(`ub:Person`, `ub:memberOf`) – query 5.
- Domain(`ub:Student`, `ub:takesCourse`) – queries 7 and 9 and 10.
- Range(`ub:takesCourse`, `ub:Course`) – query 7 and 9.
- Domain(`ub:Faculty`, `ub:teacherOf`) – query 9.
- Range(`ub:worksFor`, `ub:Department`) – query 12.
- Range(`ub:hasAlumnus`, `ub:Person`) – query 13.

Since the dataset contains all triples inferred by SwiftOWLIM, all constraints hold, as evidenced by the lack of errors reported by the validation.

Figure 1 gives the validation times for both constraint sets on various sizes of the LUBM dataset. It is clear that the validation takes considerable time. For the full set of constraints on the LUBM-100 dataset with 22,2 million triples, it takes almost exactly 30 hours. Stripping the constraint set only to the most important ones gives a validation time that is much shorter: 9 hours and 33 minutes. It is still considerable though. The fact that the validation needs to be repeated after each change in the dataset limits the viability of this approach to read-only scenarios.

We envision that the problem of long validation times can be approached in at least two ways. The first way entails parallelization of the process. It is possible to evaluate the validation queries in parallel. This would be especially beneficial on in-memory RDF stores where the disk I/O is not an issue.

The second idea for improvement is to assume that the underlying store supports inference, then analyze the ontologies present in the dataset and elicit constraints directly from those ontologies. Then we could assume that all of those constraints are valid without actually having to validate them.

6.3.3 Query Optimization Tests

When the dataset is validated, the optimizer can optimize each query. The optimization involves the Chase and Backchase (C&B) algorithm whose complexity rises

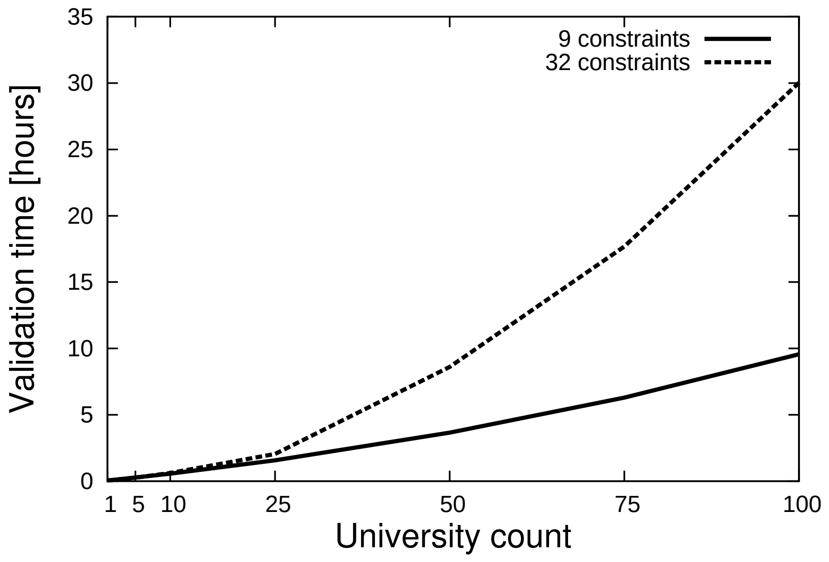


Fig. 1. Validation times

very quickly with the number of constraints and the query size (number of triple patterns in the largest basic graph pattern).

In order to perform the optimization tests, a simple testbed application has been developed. The aim of the query optimization testbed was to measure two parameters: the query evaluation time and total query time. The total query time is the time that elapses between the call to the evaluate() method in the client application and the end of iteration over the result set.

The evaluation time does not take the optimization into account. It is the time needed for the evaluation of the query by the underlying storage layer. We measure it in order to check if the optimized query is actually faster than the unoptimized one.

It is possible that an optimized query can have a better evaluation time than an unoptimized one, but a worse total time. This does not invalidate the entire semantic optimization approach though. It simply means that the performance of the optimization component plays a role here, and a better implementation would yield better results. As already mentioned in Section 5 the chase and backchase engine we use is prototypical and leaves much room for improvement. Each query was evaluated five times and the median value has been chosen for the results.

6.3.4 Results

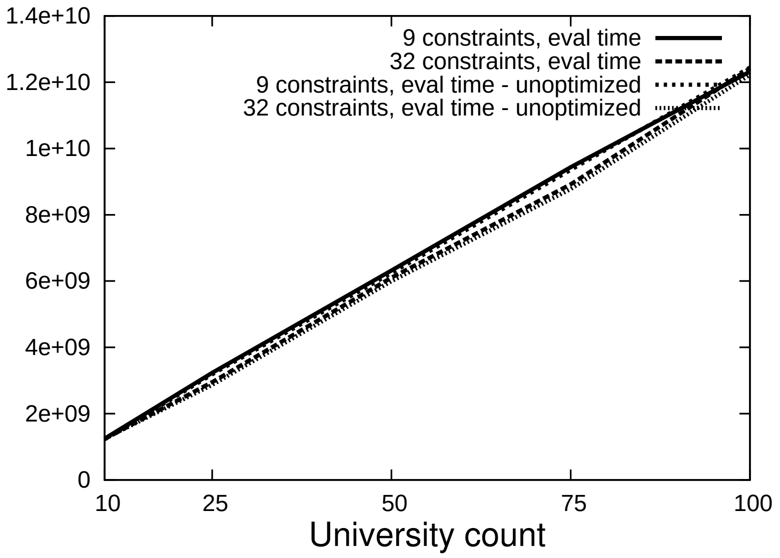
There were 14 queries. 6 of them went unchanged through the optimization process. These were queries 1, 4, 6, 8, 11 and 14. They are important, because we used them to validate our assumptions about the optimization process. We expected that unchanged queries would yield unchanged evaluation times (A2.1) and that the size of the constraint set has no effect on it (A2.2). This has been confirmed. We also expected the total times to rise with the size of the constraint set. This was also the case for all six queries. An example query 6 is shown in Figure 2.

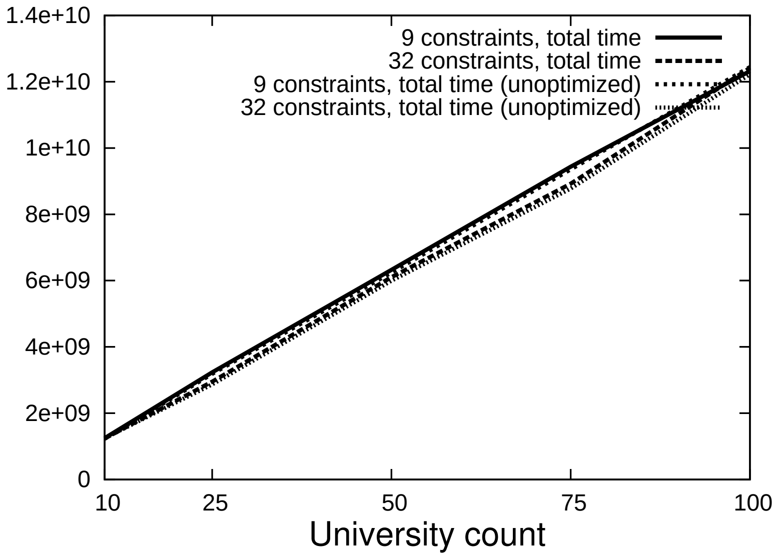
Hypothesis H2.1 is very important for the viability of the semantic optimization approach. There is little sense in devising and implementing a complex optimization algorithm if the "optimized" queries are not actually faster. From all 8 queries, which were affected by the optimization, the evaluation time was clearly improved in six: 3, 5, 7, 9, 10 and 13. Query 13 is the only query where we achieved an improvement in total query time. The optimization has changed the characteristic of the query time curves. Before the optimization it rose linearly with the dataset size, afterwards it became much more flat, almost constant. The result is visible in Figure 2.

The most important counterexample though is query 2 (also shown in Figure 2). Removing two triple patterns from 6-pattern query changed the evaluation process drastically. For LUBM-100 the evaluation time rose from 33 to 507 seconds, i.e. by a factor of 15. It seems that the patterns that were removed were critical in the evaluation process, and allowed the storage engine to reduce the search space with the help of the internal hard-disk indexes that were available. This result is a proof that without knowledge about the implementation of the underlying storage it is impossible to construct a semantic optimization engine that would guarantee that the evaluation time will not increase.

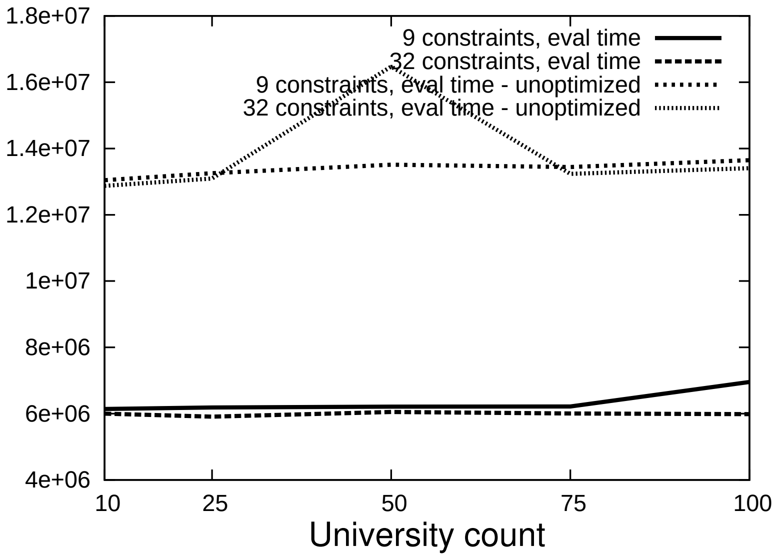
We envision that an improved version of the engine could use information about the indexes and predicate frequency to decide whether removing a particular triple pattern would actually be beneficial for the evaluation time. Moreover the same engine might use the chase algorithm not only to look for patterns to remove. The result of the Chase algorithm is an expanded query, it could be used as a repository of patterns that could be added to the query, so that the case with query 2 could actually be reversed. Exploring these ideas remains a point for future work.

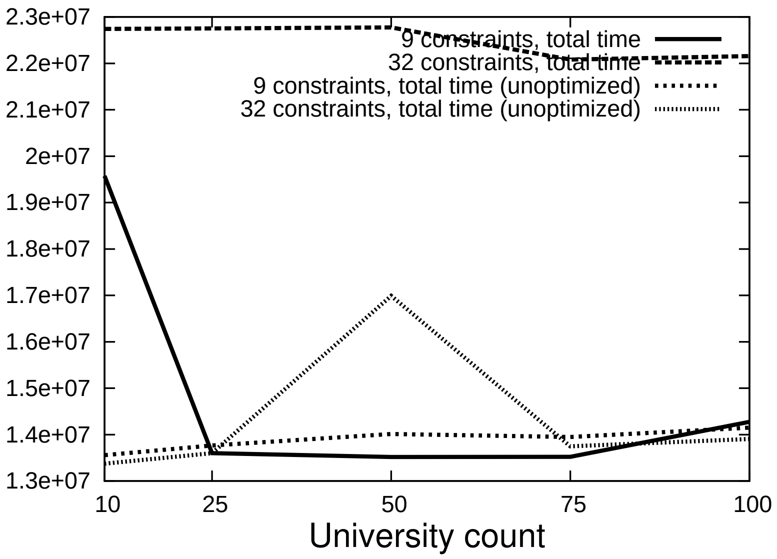
With the current prototype, only query 13 confirms the H2.2 hypothesis. In that case we see that the time rises almost linearly with that dataset size. After optimization, the curve is also linear, but with a smaller coefficient. The measurements of evaluation time in other queries indicate that this result might be improved by reimplementing the chase and backchase engine. The one we currently use is based on MARS. In our scenario we do not need the expressivity and the general-purpose functionality MARS provides. A simple assumption that all constraints and all queries are defined in terms of a single three-argument relation could allow for a much more efficient, special-purpose representation of queries, constraints which would allow for a faster and simpler implementation of the chase and backchase algorithms. Such a reimplementation could reduce the difference between the total and evaluation times in all queries. It remains a point for future work though.



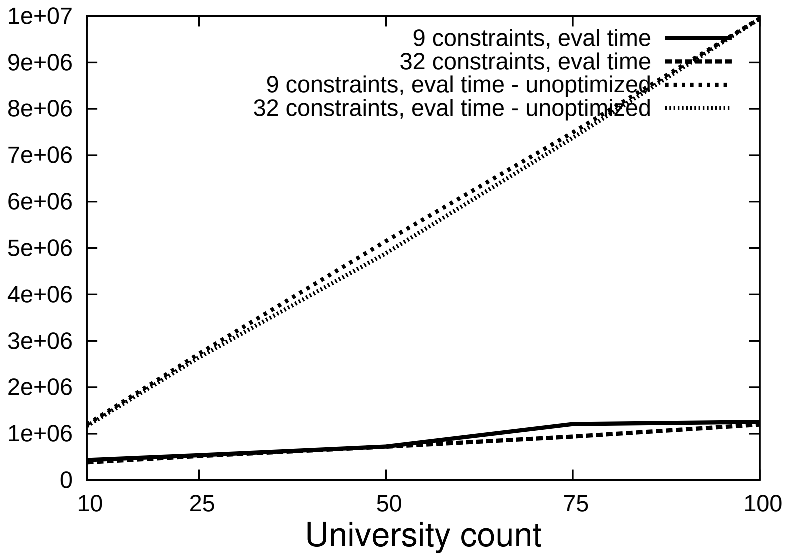


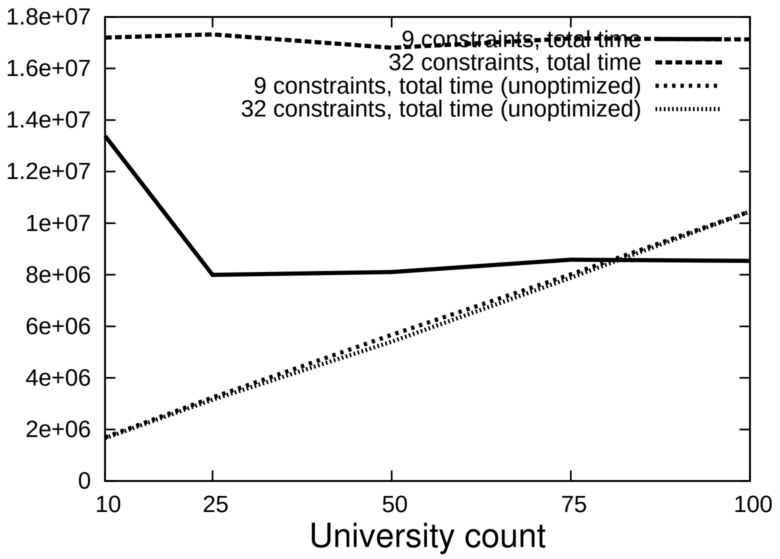
Query 6, unchanged.



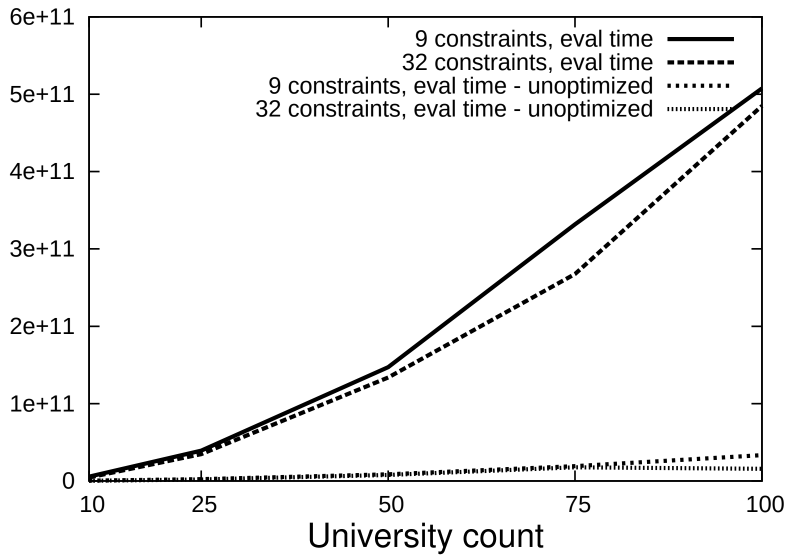


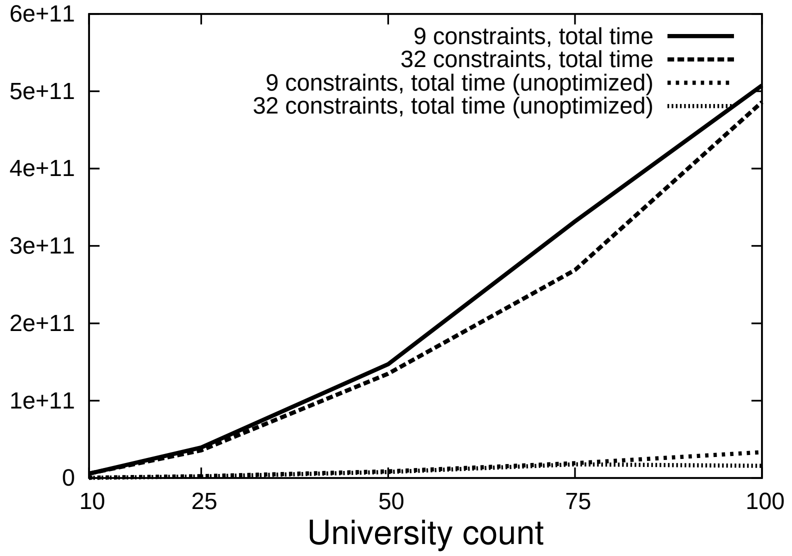
Query 5, better evaluation time but worse total time for 32 constraints.





Query 13, success.





Query 2, failure.

Graphs on the left show the evaluation times, those on the right show total times. The unit on the y-axis is a nanonsecond.

Fig. 2. Optimization experiment result highlights

7 CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The conversion experiments show that X2R is indeed capable of converting large amounts of data by using a custom mapping. They are summarized in Table 1. In all cases the sizes of the internal database files and the size of the resulting `NativeStore` were much larger than the allocated Java heap. This confirms the H1.3 hypothesis from Section 6.1 – the dataset was indeed larger than the amount of available RAM. In Section 6.2.2 we have provided numerous arguments that the chosen dataset is non-trivial and "realistic" which supports the H1.1 hypothesis. The conversion succeeded on a relational, XML and an LDAP database, which confirms the H1.2 hypothesis. The system was able to work with a large dataset for extended period of time which proves that the approach we took has the potential to be applicable in an industrial setting. On this basis we conclude that the goals for the RDF conversion task have been fulfilled.

The results of the experiments with integrity constraints show that some of the assumptions, while interesting in theory, proved to be problematic in practice. Our approach has shown three fundamental issues: validation time, optimization efficiency, and the actual evaluation time of the "optimized" queries.

Firstly, the idea that integrity constraints for RDF can be checked with SPARQL queries, advertised as promising in [24]. Figure 1 shows that in a deployment with realistic amount of data with many constraints the validation time can become very long. In our experiment it took more than 30 hours, which makes it very problematic to apply this approach as it is in any practical scenario.

Secondly, the time required by our prototype optimizer has negated almost all benefits of improved evaluation time. From the 14 queries 8 have been minimized by the optimization process. The only real improvement in the total query time is visible on the graph for LUBM query 13 (Figure 2) by extrapolating the curve to about 150 universities. Six further ones achieved better evaluation time though, which means that an improved optimization component would certainly yield better results, the benefit for query 13 would be visible for smaller dataset size and maybe in at least some of the 6 queries the benefit in evaluation time could exceed the cost of optimization.

Query 2 proved to be really problematic though. It turned out that our hypotheses were too simple. The evaluation time of a query does not always rise linearly with the dataset size. It depends on a number of factors like the indices available to the query evaluation engine or the caching employed by the storage component. The result for LUBM query 2 (Figure 2) shows that in certain cases making a query shorter does not make it faster. Our idea to implement the optimization component as independent of the underlying storage implementation was too optimistic. The optimizer would need additional knowledge to decide whether or not removing a part of the SPARQL query might make it faster, even if the result is theoretically equivalent.

7.2 Future Work

There are many avenues for improvement in all parts of the X2R functionality.

Probably the most important area from the user's point of view is the expressivity of the mapping languages. Section 6.2.3 has given numerous ideas for new features, both in the XML and the LDAP adapters.

Section 6.2.6 notes the problem with the size of the temporary file produced by the Sedna XML database, which could be alleviated by making the adapter generate many smaller queries instead of one large query. Also, various ways to parallelize the conversion process could be explored.

As described in Section 6.3.2, the validation time with 32 constraints proved excessive with large datasets. It seriously limits the applicability of this approach in real world scenarios. The most obvious idea to make it faster is with parallelization, since the validation process does not change anything in the repository, more concurrent readers would not require any synchronization. This would be especially beneficial in in-memory scenarios, on multi-processor machines, where the concurrent access to fixed storage does not become a bottleneck. It would be interesting to see how a parallelized validator performs with a large data set loaded into an in-memory database (like SwiftOWLIM) on a machine with enough available Random Access Memory.

Furthermore if we can assume that the underlying RDF store supports inference, many constraints can be elicited directly from the ontologies present in the dataset. Such constraints would not have to be validated at all. This avenue for improvement is all the more promising when one takes into account that most of the 32 LUBM constraints actually hold only because the dataset has been prepared to contain the entire OWL inference closure.

Section 6.3.4 lays out two important directions towards the improvement of the optimization engine. Firstly the performance of the Chase and Backchase engine should be improved, possibly by reimplementing it as a special-purpose tool, designed specifically for optimizing triple patterns of SPARQL queries. The scope of MARS greatly exceeded our needs. A tool designed for SPARQL could be much simpler and more efficient. Secondly the Chase and Backchase engine should leverage the information about indexes and predicate frequency from the underlying storage, possibly combining them with join-reordering algorithms, to find an optimal rewriting of Basic Graph Patterns from the original query. LUBM query 2 showed that a shorter query can be slower.

Therefore the rewriting rules given in [34] can only be understood as instructions how to make steps, but do not give any direction about where to go. More research is needed. The result for query 2 can be reversed into "in which cases does adding new triple patterns which retain equivalence make a query faster". Exploring this question, in combination with other improvements is in our opinion vital to the practical applicability of semantic optimization of SPARQL.

The last paragraph of Section 6.3.1 raises the point that the LUBM tests did not cover the advanced optimizations supported by the optimizer, i.e. multi-level projection pushing and OPTIONAL operator optimization. Proper testing for them would require a benchmark with more complex queries than those in LUBM. Michael Schmidt remarked in [35] that one of the queries from his benchmark proposal – SP^{2}Bench – could be an example for semantic optimization. This is only a very limited answer. The design of a benchmark designed specifically for complex constraint-based optimization of SPARQL queries still remains an open challenge.

The prototype does not try to tackle the problem of Chase termination. In general case it is possible to construct a set of constraints and a query for which the Chase algorithm will never end. There exist a number of known sufficient conditions that guarantee Chase termination under certain assumptions [29]. An industrial-grade optimizer would have to perform such a check before starting the Chase.

Acknowledgements

This research is partially supported by European Regional Development Fund programs PLGrid POIG.02.03.00-00-007/08-00 and ITSOA POIG.01.03.01-00-008/08-01.

REFERENCES

[1] Abiteboul, S.—Hull, R.—Vianu, V.: Foundations of Databases. Addison-Wesley 1995.

[2] Aduna: Sesame 2.3 user guide. Documentation for Sesame, available at `http://www.openrdf.org/doc/sesame2/2.3.2/users/index.html`.

[3] Auer, S.—Dietzold, S.—Lehmann, J.—Hellmann, S.—Aumueller, D.: Triplify: Light-Weight Linked Data Publication from Relational Databases. In WWW (2009), J. Quemada, G. Léon, Y. S. Marek, and W. Nejdl, (Eds.), ACM, pp. 621–630.

[4] Battle, S.: Gloze: XML to RDF and Back Again. In Proceedings of the First Jena User Conference 2006.

[5] Beckett, D.—Berners-Lee, T.: Turtle-Terse RDF Triple Language. Team Submission, published on the World Wide Web Consortium website at `http://www.w3.org/TeamSubmission/turtle/`, 14 January 2008.

[6] Beeri, C.—Vardi, M. Y.: A Proof Procedure for Data Dependencies. J. ACM, Vol. 31, 1984, No. 4, pp. 718–741.

[7] Berners-Lee, T.: Relational Databases on the Semantic Web. Design Note, published on the World Wide Web Consortium website at `http://www.w3.org/DesignIssues/RDB-RDF.html`, October 1998.

[8] Berners-Lee, T.: Why RDF Model Is Different from the XML Model. Design Note, published on the World Wide Web Consortium website at `http://www.w3.org/DesignIssues/RDB-XML.html`, September 1998.

[9] Berners-Lee, T.—Hendler, J.—Lassila, O.: The Semantic Web. Scientific American, May 2001.

[10] Bizer, C.—Cyganiak, R.—Garbers, J.—Maresch, O.—Becker, C.: The D2RQ Platform v0.7 – Treating Non-RDF Relational Databases as Virtual RDF Graphs, User Manual and Language Specification. `http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/20090810/`, 10 August 2009.

[11] Bizer, C.—Seaborne, A.: D2RQ, Treating Non-RDF Databases as Virtual RDF Graphs. Poster presented at the Third International Semantic Web Conference, Hiroshima, Japan, available at `http://www4.wiwiss.fu-berlin.de/bizer/pub/Bizer-D2RQ-ISWC2004-Poster.pdf`, November 7–11 2004.

[12] Brickley, D.—Guha, R. V.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 10 February 2004.

[13] Broekstra, J.—Kampman, A.—Van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In International Semantic Web Conference 2002, I. Horrocks, J. A. Hendler, (Eds.), Vol. 2342 of Lecture Notes in Computer Science, Springer, pp. 54–68.

[14] Deutsch, A.—Tannen, V.: MARS: A System for Publishing XML from Mixed and Redundant Storage. In VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9–12, 2003, Berlin, Germany (2003), J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, Eds., Morgan Kaufmann, pp. 201–212.

[15] Dietzold, S.: Generating RDF Models from LDAP Directories. In Proceedings of the Fifth Workshop on Scripting for the Semantic Web 2005.

[16] Dietzold, S.—Auer, S.: Accessing RDF Knowledge Bases Via LDAP Clients. In Proceedings of International Conference Semantics Systems 2007, I-SEMANTICS '07, Graz, Austria; September 5–7, 2007, T. Pellegrini and S. Schaffert, Eds., Journal of Universal Computer Science, pp. 290–296.

[17] Fehlman, M.: XML to RDF Transformation. Master's thesis, University of Zurich 2006.

[18] Garca, R.—Gil, R.: Facilitating Business Interoperability from the Semantic Web. In Business Information Systems, 10th International Conference, BIS 2007, Poznan (Poland), April 25–27, 2007, Proceedings (2007), W. Abramowicz, Ed., Vol. 4439 of Lecture Notes in Computer Science, Springer, pp. 220–232.

[19] Gruber, T. R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing? Int. J. Hum.-Comput. Stud., Vol. 43, 1995, No. 5-6, pp. 907–928.

[20] Guo, Y.—Pan, Z.—Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics, Vol. 3, 2005, No. 2-3, pp. 158–182.

[21] Klyne, G.—Caroll, J. J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 10 January 2004.

[22] Knublauch, H.: SPIN, SPARQL Inferencing Notation. Project website, available at `http://www.spinrdf.org`, October 20, 2009.

[23] Lange, C.: Krextor an Extensible XML to RDF Extraction Framework. In Proceedings of the Fifth Workshop on Scripting and Development for the Semantic Web, co-located with 6th European Semantic Web Conference (ESWC), Heraklion, Greece (31 May–4 June 2009), S. Auer, C. Bizer, and G. A. Grimnes, Eds., Vol. 449 of CEUR Workshop Proceedings, ISSN 1613-0073.

[24] Lausen, G.—Meier, M.—Schmidt, M.: SPARQLing Constraints for RDF. In EDBT 2008, A. Kemper, P. Valduriez, N. Mouaddib, J. Teubner, M. Bouzeghoub, V. Markl, L. Amsaleg, and I. Manolescu, Eds., Vol. 261 of ACM International Conference Proceeding Series, ACM, pp. 499–509.

[25] Leser, U.—Naumann, F.: Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen. dpunkt Verlag 2006.

[26] Ley, M.: Datenqualität: Eine organisatorische und technische Herausforderung – Erfahrungen von der DBLP-Bibliographie. In Grundlagen von Datenbanken 2007, H. Höpfner and F. Klan, Eds., Vol. 02/2007 of Technical Report, School of Information Technology, International University in Germany, p. 1.

[27] Ley, M.: DBLP – Some Lessons Learned. PVLDB, Vol. 2, 2009, No. 2, pp. 1493–1500.

[28] Maier, D.—Mendelzon, A. O.—Sagiv, Y.: Testing Implications of Data Dependencies (Abstract). In SIGMOD Conference 1979, P. A. Bernstein, Ed., ACM, p. 152.

[29] Meier, M.—Schmidt, M.—Lausen, G.: On Chase Termination Beyond Stratification. PVLDB, Vol. 2, 2009, No. 1, pp. 970–981.

[30] Prud'hommeaux, E.—Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation, 15 January 2008.

[31] Reif, G.: WEESA – Web Engineering for Semantic Web Applications. Ph. D. thesis, Technische Universitaet Wien, 2005.

[32] Rodrigues, T.—Rosa, P.—Cardoso, J.: Moving from Syntactic to Semantic Organizations Using JXML2OWL. Computers in Industry, Vol. 59, 2008, No. 8, pp. 808–819.

[33] Sahoo, S. S. et al.: A Survey of Current Approaches for Mapping of Relational Databases to RDF. World Wide Web Consortium RDB2RDF Incubator Group, available at `http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf`, 2009.

[34] Schmidt, M.: Foundations of SPARQL Query Optimization. Ph. D. thesis, Albert-Ludwigs-Universitt Freiburg im Breisgau, 2009.

[35] Schmidt, M.—Hornung, T.—Lausen, G.—Pinkel, C.: SP2Bench: A SPARQL Performance Benchmark. In ICDE (March 29–April 2 2009), IEEE, pp. 222–233.

[36] Schmidt, M.—Meier, M.—Lausen, G.: Foundations of SPARQL Query Optimization. In Proceedings of the 13th International Conference on Database Theory (March 22–25 2010), L. Seguofin, Ed.

[37] Seaborne, A.—Steer, D.—Williams, S.: SQL-RDF. Paper accepted at the W3CWorkshop on RDF Access to Relational Databases, Cambridge, MA, USA available at `http://www.w3.org/2007/03/RdfRDB/papers/seaborne.html`, October 25–26, 2007.

[38] Sirin, E.—Smith, M.—Wallace, E.: Opening, Closing Worlds – On Integrity Constraints. In OWLED 2008, C. Dolbear, A. Ruttenberg, U. Sattler, Eds., Vol. 432 of CEUR Workshop Proceedings, CEUR-WS.org.

[39] Sure, Y.—Bloehdorn, S.—Haase, P.—Hartmann, J.—Oberle, D.: The SWRC Ontology – Semantic Web for Research Communities. In EPIA 2005, C. Bento,

A. Cardoso, G. Dias, Eds., Vol. 3808 of Lecture Notes in Computer Science, Springer, pp. 218–231.

[40] Svihla, M.—Jelinek, I.: METAmorphoses – SQL to RDF Mapping for Semantic Web. Poster presented at the 18th International Conference on Systems for Automation of Engineering and Research (SAER 2004), Sofia, Bulgaria, available at `http://www.svihla.net/research/publications/posters/msvihla_saer2004poster_800x600.pdf`, November 7–11 2004.

[41] W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview. W3C Recommendation, 27 October 2009.

[42] Wiesner, A.—Morbach, J.—Marquardt, W.: Semantic Data Integration for Process Engineering Design Data. In ICEIS 1, 2008, J. Cordeiro and J. Filipe, Eds., pp. 190–195.

Antoni Myłka graduated in 2010 from the AGH University of Science and Technology in Kraków, Poland. During his studies he gathered experience with semantic technologies in research projects at ACK Cyfronet in Krakw and at DFKI in Kaiserslautern, Germany. Since his graduation he has been working as an independent software developer based in Krakw, cooperating with DFKI and Aduna.

Alina Myłka graduated in 2010 from the AGH University of Science and Technology in Kraków, Poland. Her formation as a software professional included internships at Fraunhofer IESE and IBM in Germany, followed by a period of strong focus on RDF-based data integration, which culminated in her Master's Thesis. Currently she works at Antenna Software on projects related to mobile web applications.

Bartosz Kryza is researcher and developer at the CYFRONET Academic Computer Center in Kraków. He has participated in several EU-IST projects as task or WP leader including FP5 CrossGrid, FP5 Pellucid, FP5 MAGIC (during research internship in France), FP6 K-Wf Grid, FP6 GREDIA and FP7 PRACE. His main areas of interest are at the convergence of Grid systems and semantic technologies, SOA architectures and virtual organisations, distributed data management and P2P technologies. He is the author or co-author of about 30 research papers published in international journals or conference proceedings.

Jacek Kitowski is Full Professor of computer science (`www.icsr.agh.edu.pl/~kito`), graduated in 1973 at the Electrical Department of the AGH University of Science and Technology in Kraków (Poland). Head of the Computer Systems Group at the Department of Computer Science of the AGH University of Science and Technology. Researcher at the Academic Computer Centre CYFRONET-AGH, with responsibility on development of high performance systems and Grid/Cloud ecosystems. He is the author or co-author of about 200 scientific papers. Topics of interest: large-scale computations, multiprocessor architectures, Cloud and Grid services, Grid and Cloud storage systems, knowledge engineering. Participant of many national and international projects, funded by European Commission and EDA. Member of program committees of international conferences and evaluator of national and international IT projects. Expert in EU Program Committee "e-Infrastructures" (Unit F3) and Director of PL-Grid Consortium running Polish NGI.