D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs

Christian Bizer
Freie Universität Berlin,
Berlin, Germany
chris@bizer.de

Andy Seaborne
Hewlett-Packard Labs,
Bristol, UK
andy.seaborne@hp.com

Abstract

As Semantic Web technologies are getting mature, there is a growing need for RDF applications to access the content of huge, live, non-RDF, legacy databases without having to replicate the whole database into RDF. In this poster, we present D2RQ, a declarative language to describe mappings between application-specific relational database schemata and RDF-S/OWL ontologies. D2RQ allows RDF applications to treat non-RDF relational databases as virtual RDF graphs, which can be queried using RDQL.

1 Introduction

It will be crucial for many real-world Semantic Web applications to be able to access the content of non-RDF relational databases used by most legacy systems. D2RQ is a declarative language to describe mappings between application-specific relational database schemata and RDF-S/OWL ontologies. Using D2RQ, Semantic Web applications can:

- query a non-RDF database using RDQL,
- publish the content of a non-RDF database on the Semantic Web using the RDF Net API,
- do RDFS and OWL inferencing over the content of a non-RDF database using the Jena ontology API,
- access information in a non-RDF database using the Jena model API.

D2RQ is implemented as a Jena graph, the basic information representation object within the Jena framework [Carroll *et al*., 2004]. A D2RQ graph wraps one or more local relational databases into a virtual, read-only RDF graph. D2RQ rewrites RDQL queries and Jena API calls into application-datamodel-specific SQL queries. The result sets of these SQL queries are transformed into RDF triples which are passed up to the higher layers of the Jena framework.
Figure 1 shows the architecture of a D2RQ usage scenario, where a relational database is maintained by a non-RDF legacy application. Using D2RQ, the content of the database can simultaneously be accessed by an RDF application or can be published on the Web using the RDF Net API [Moore and Seaborne, 2003].

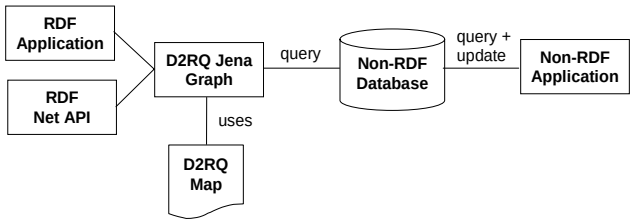

Figure 1: Application Architecture

2 The Mapping Language

The D2RQ mapping language builds on experience gained with D2R [Bizer, 2003]. The central object within D2RQ is the ClassMap. A ClassMap represents a class or a group of similar classes from the ontology. It specifies whether instances are identified by using URI column values from the database, by using an URI pattern together with the primary key values or by using blank nodes. Each ClassMap has a set of property bridges, which specify how instance properties are created and how given URIs or literals are reversed into database values. There are two types of property bridges: DatatypePropertyBridges for literals and ObjectPropertyBridges for URIs and for referring to instances created by other class maps. Property values can be created directly from database values or by using patterns and translation tables.
D2RQ supports conditional mappings on class map and property bridge level. It supports the mapping of properties being used by several classes and the handling of highly normalized table structures where instance data is spread over several tables.

3 Example

Figure 2 shows the structure of a D2RQ map relating the classes ex:Person and ex:Paper to a relational data model. Papers and Persons are linked by the ex:author property. Because authors usually have more than one publication and publications can be written by several authors, information would typically be stored in three tables on the database side: One for the persons, one for their papers and a third one for the n:m relationship between persons and papers.

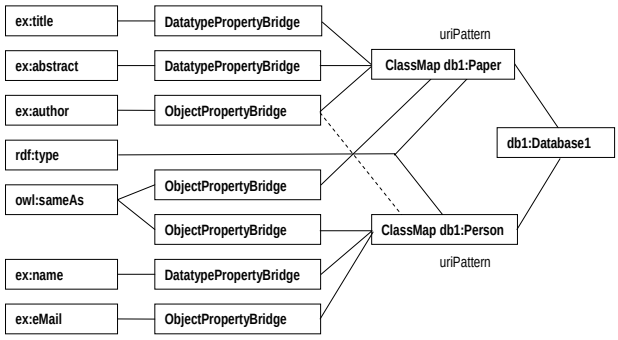

Figure 2: Structure of the Example D2RQ Map

In order to find all authors of a given paper, e.g.

```
SELECT ?x
WHERE (<http://www.papers.org/3465>,
       ex:author, ?x)
```

the following parts of the map are used:

```
ex:author rdf:type rdf:Property ;
   d2rq:propertyBridge db1:author .
db1:author rdf:type d2rq:ObjectPropertyBridge;
   d2rq:belongsToClassMap db1:Paper ;
   d2rq:refersToClassMap db1:Person ;
   d2rq:join "Papers.PaperID =
              PersonPaper.PaperID" ;
   d2rq:join "PersonPaper.PersonID =
              Persons.PersonID" .

ex:Paper rdf:type rdfs:Class ;
   d2rq:classMap db1:Paper .
db1:Paper rdf:type d2rq:ClassMap ;
   d2rq:uriPattern
   "http://www.papers.org/@@Papers.PaperID@@";
   d2rq:dataStorage db1:Database1 .

ex:Person rdf:type rdfs:Class ;
   d2rq:classMap db1:Person .
db1:Person rdf:type d2rq:ClassMap ;
   d2rq:uriPattern
   "mailto:@@Person.Email@@";
   d2rq:dataStorage db1:Database1 .
```

The ObjectPropertyBridge db1:author contains the information that the property ex:author belongs to ClassMap db1:Paper and that it refers to instances of the ClassMap db1:Person. The bridge also specifies how the corresponding database tables should be joined. Using this information and the uriPattern from ClassMap db1:Paper, the given URI is reversed into the database value 3465 and the following SQL statement is built:

```
SELECT Persons.Email
FROM PersonPaper, Persons
WHERE PersonPaper.PersonID = Persons.PersonID
AND PersonPaper.PaperID = 3465;
```

The statement is executed against db1:Database1 and the SQL result set is transformed into an RDQL result set using the uriPattern from ClassMap db1:Person.

4 Performance

We performed a series of benchmarks comparing the performance of D2RQ to the performance of the Jena2 database backend. As benchmarking dataset we used the descriptions of 200,000 papers from the DBLP Computer Science Bibliography [DBLP, 2004]. The data was stored using both an application-specific relational data model and the Jena database backend (1.6M triples). Table 3 shows the benchmarking results for different find(spo) patterns.

	Jena2 DB	D2RQ
1. find (s ? ?)	1.83 ms	0.01 ms
2. find (? p o)	1.94 ms	0.97 ms
3. find (? p ?)	42431 ms	72 ms
4. find (? ? o)	1.72 ms	3.23 ms

Table 3: Benchmarking Results

The benchmarks show that D2RQ is competitive to the Jena database backend for common patterns (pattern 1. and 2.). Both architectures have different worst cases (pattern 3. and 4.). The exact queries used for the tests and more detailed results are found in [Cyganiak, 2004].

5 Conclusion

D2RQ offers a flexible, easy-to-use access mechanism to non-RDF databases. It allows the integration of legacy databases into the data access architecture currently standardized by the W3C Data Access Working Group [W3C, 2004].
D2RQ is available under GNU GPL. The complete language specification and further examples are found at http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rq/

References

[Bizer, 2003] Christian Bizer. D2R MAP – A Database to RDF Mapping Language. Poster at the *12th World Wide Web Conference*, Budapest, May 2003.

[Moore and Seaborne, 2003] Graham Moore, Andy Seaborne. RDF Net API. *W3C Member Submission*, October 2003.

[Carroll *et al.*, 2004] Jeremy J. Carroll *et al.*: Jena: Implementing the Semantic Web Recommendations. In *Proceedings of the 13th World Wide Web Conference*, New York City, May 2004.

[Cyganiak, 2004] Richard Cyganiak: Benchmarking D2RQ V0.2. http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rq/benchmarks, August 2004.

[DBLP, 2004] DBLP: Computer Science Bibliography Website. http://dblp.uni-trier.de/

[W3C, 2004] W3C: Data Access Working Group Website. http://www.w3.org/2001/sw/DataAccess/