Transient and persistent RDF views over relational databases in the context of digital repositories

Nikolaos Konstantinou[1], Dimitrios-Emmanuel Spanos[1], Nikolas Mitrou[2]

[1] Hellenic Academic Libraries Link
Iroon Polytechniou 9, Zografou, 15780, Athens, Greece
[2] School of Electrical and Computer Engineering, National Technical University of Athens
Iroon Polytechniou 9, Zografou, 15780, Athens, Greece
nkons@cn.ntua.gr, dspanos@cn.ntua.gr, mitrou@cs.ntua.gr

Abstract. As far as digital repositories are concerned, numerous benefits emerge from the disposal of their contents as Linked Open Data (LOD). This leads more and more repositories towards this direction. However, several factors need to be taken into account in doing so, among which is whether the transition needs to be materialized in real-time or in asynchronous time intervals. In this paper we provide the problem framework in the context of digital repositories, we discuss the benefits and drawbacks of both approaches and draw our conclusions after evaluating a set of performance measurements. Overall, we argue that in contexts with infrequent data updates, as is the case with digital repositories, persistent RDF views are more efficient than real-time SPARQL-to-SQL rewriting systems in terms of query response times, especially when expensive SQL queries are involved.

Keywords: Linked Open Data, RDF Views, Bibliographic information, Digital Repositories, R2RML, Mapping.

1 Introduction

The Linked Open Data (LOD) movement is constantly gaining worldwide acceptance, emerging as one of the most prominent initiatives of the Web 3.0 era. As it can be observed, in many aspects of public information, a shift toward openness is taking place. The value of exposing data as LOD is being recognized in the cultural heritage domain (europeana.eu, clarosnet.org), governance (data.gov.uk), even in the news world (guardian.co.uk/data).

The technological building blocks that contribute to this shift have reached to a maturity level, able to sustain production environments available to public access. This is made available using technologies such as HTTP, XML, RDF and SPARQL, all internationally accepted W3C standards. Having these as technological background, the LOD vision is largely being materialized, slowly but steadily, by bringing existing data into the Semantic Web.

As to what drives the changes towards this direction, we can observe numerous benefits, both for the publishers, as well as for the target audience, or consumers:

- Ease of *synthesis* with external data sources, in the form of integration (beyond OAI-PMH), fusion, and mashups. The end user (or developer) can perform searches and retrieve results spanning various repositories, from a single SPARQL endpoint. Also, it is made possible to download parts or the whole data in order to combine it with other data and process it according to his/her needs.
- Semantic *enrichment*. Term and definition ambiguity is eliminated, allowing data to be uniquely understood and consumed both by humans as well as by software agents.
- *Inference*. It is possible to infer implicit facts based on explicitly stated ones. These new facts can then be added to the graph, thus augmenting the initial knowledge base.
- *Reusability*. Third party applications can be built on top of the datasets, as the data can be reused in third-party systems. This can be materialized, either by including the information in their datasets, or by real-time querying the published resources.
- *Intelligence* in the queries. Taking into advantage ontology hierarchy and concept interrelations, results can be obtained that exceed the keyword capabilities. In the same manner, Google uses the Freebase ontology[1] in order to return intelligent results which are more related to the users' queries.
- Digital repository *content can be linked*, and made part of the broader context of the Web. Instead of being an isolated dataset targeting a special group of people, library data can better fulfill their purpose by being part of the information users can discover and reuse on the Web.
- Richer *expressiveness* in describing and querying available information. Both the terminology that can be used to describe the dataset, as well as the queries that can be posed against it, with the use of semantic web technologies can be more complex and more expressive, allowing for a richer set of capabilities.

In order to offer solutions towards creating LOD, the methodological approaches that enable it can largely be regarded as falling into one of the following two approaches: in the first approach, *"transient"* RDF views are offered on top of the data, in the sense that the RDF graph is not materialized; instead, queries on the RDF graph are answered with data originating from the actual dataset, in a manner similar to the concept of SQL views. The second approach involves *"persistent"* RDF views, meaning that the data is exported (or dumped, as it is often called) asynchronously in RDF graphs. In the *"persistent"* approach, the idea is similar to the materialized view: data is exported in an RDF graph leaving the source unaltered.

It is interesting to mention, since adopting RDF as a dataset format to work upon entails so many benefits, a valid question would be, why not redesign new systems to operate fully using RDF graphs as their data backend? First of all, technologies such as relational databases have matured in the latest years far more than semantic technologies, offering a richer capability set. Furthermore, current established practices utilize technologies that have been successfully tested through time and have proven effective in preserving digital information and assuring its unaltered endurance in time. Therefore, such mature and reliable technologies cannot be

[1] Freebase API: https://developers.google.com/freebase/

abolished without at least allowing for a period of time to run both technologies alongside. This would allow for any problems to be accentuated and be solved before jumping to new technologies. In the case of LOD generation, potential risks are more associated with lack of expertise by the personnel that needs to be trained in order to adopt and operate the new technologies, and less associated with the maturity of the related technologies themselves.

Next, we discuss about available approaches regarding how to operate an institutional repository, whose operation typically involves a number of persons, an established methodology, and an infrastructure that is optimized towards serving its goals. With this in mind, instead of fully migrating to newer technologies, we rather suggest operating them side-by-side, as an additional content distributing channel, over the same source dataset, comprising digital repository contents.

All the above lead to the conclusion that in order to expose digital repository contents as LOD, several policy-related choices have to be made, since several alternative approaches exist in the literature, without any one-size-fits-all approach [1]. One of the most important factors to be considered is discussed in this paper: Should RDF provisions take place in real-time or should database contents be dumped into RDF at time intervals? Or, as explained before, should the RDF view over the contents be *transient* or *persistent*?

Both approaches constitute viable approaches, each with its specific characteristics, benefits and drawbacks. However, each case requires specific handling, in the sense that there are no one-solution-fits-all approaches. In this paper we analyze the pros and cons of each method as far as the institutional digital repository domain is concerned, taking into account the particularities it presents. Performance measurements are conducted concerning the exporting and querying times in variable initial datasets and settings, and respective measurements are presented and discussed.

The paper is structured as follows. Section 2 overviews related approaches that can be found in the bibliography. Section 3 presents the environment in which the measurements took place, the results, and an in-depth analysis and discussion. Section 4 concludes the paper with our most important conclusions and directions future work could take.

2 Related Work

The problem of generating RDF content from existing data sources has been investigated extensively and has gradually become a common task for data providers who wish to make their data available as RDF and reap the associated benefits discussed in Section 1. The data sources that a provider may have at her disposal will normally range from unstructured, free-text documents to semi-structured spreadsheets and structured databases.

The latter ones represent one of the most popular sources of data, with widespread adoption and a mature theoretical and practical background. Likewise, the problem of mapping relational database contents to an RDF graph has attracted a fair amount of attention and several solutions for carrying out this task are available. Such solutions and tools – often coined by the term RDB2RDF tools – present considerable variance and can be classified to distinct categories, according to a number of criteria [2]. One such criterion is the access paradigm of the generated RDF graph, according to which RDB2RDF methods can be classified to *massive dump* and *query-driven* ones. The former ones, also known as batch transformation or Extract-Transform-Load (ETL) approaches, generate a new RDF graph from a relational database instance (e.g. [3, 4]) and store it in physical form in some external storage medium. This external medium is often a database especially customized for the management and retrieval of RDF data, which is referred to as a *triple store*. The RDF graph generated by such approaches is said to be *materialized*. Triple stores do not provide any means to transform, or maintain any kind of mappings between the relational database contents and the resulting triples, leaving the synchronization methodology up to the user. On the contrary, query-driven approaches provide access to an RDF graph that is implied and does not exist in physical form. In this case, the RDF graph is *virtual* and is only considered when some appropriate request is made, usually in the shape of a semantic query.

This distinction of tools and approaches can also be viewed under the prism of the *data synchronization* criterion, according to which methods are distinguished depending on whether the generated RDF graph always reflects the current database contents or not. Transient views, as they have been defined in Section 1, have no need of a synchronization scheme, since the accordance among the RDF graph and the underlying database is always guaranteed. Another advantage of transient views stems from the fact that they do not need any additional storage for the RDF graph produced, given that the latter is *implied* and not materialized at all.

These two advantages highlight the superiority of transient views over persistent ones. It comes as no surprise that a lot of research effort focused, over the previous years, in efficient algorithms that translate SPARQL queries over the RDF graph in semantically equivalent SQL ones that are executed over the underlying relational database instance [5, 6]. Although, on first thought, the online query translation approach might seem inefficient, some evaluation experiments, such as the ones in [7], in fact show that some SPARQL-to-SQL rewriting engines (e.g. D2RQ [8] or Virtuoso RDF Views [9, 10]) outperform triple stores in the query answering task, achieving lower responses. This is due to the maturity and optimizations strategies of relational database systems that already outperform triple stores by factors up to one hundred [7]. Therefore, as long as the SPARQL-to-SQL translation does not introduce a large delay, the transient view access paradigm will still outperform triple stores. Still however, this is not an undisputed claim, as other works have shown that such rewriting engines perform more poorly than triple stores [11].

We investigate the performance of both persistent and transient views in a digital repository context and argue that in contexts with infrequent data updates, a static approach might be more suitable than a dynamic rewriting RDB2RDF system.

3 Evaluation

This section analyzes the performance evaluation experiments that were conducted. The experimental setup is described, as well as the obtained results and conclusions that can be drawn.

3.1 Experiments Setup

In order to measure the performance of the proposed approach, three separate DSpace installations were created. Using a random generator, these installations were populated with 1k, 10k, and 100k items, respectively. The metadata that was assigned originates from the Dublin Core (DC) vocabulary, as used in "vanilla" DSpace installations. Each randomly generated item was set to contain between 5 and 30 metadata fields, with random text values ranging from 2 to 50 text characters. Moreover, a number of users were inserted to each of these repositories, populating them with 1k, 10k, and 100k users respectively. As a result, we had three repositories, one with 1k items and 1k users, 10k items and 10k users, 100k items and 100k users.

Technically, DSpace 3.1 was used, backed by a PostgreSQL 9.2 RDBMS, on a Windows 7, 64-bit machine, running on a 2.10GHz Intel Core Duo, with 4 GB RAM. In the same infrastructure, both the D2RQ experimental R2RML version 4 (available online at download.d2rq.org) and the OpenLink Virtuoso server, open-source version 6.16, x64 were installed and configured.

In the transient view case, queries were performed using D2RQ and an R2RML [12] mapping file over the DSpace database. In the persistent view case, queries were performed after exporting the DSpace database as an RDF graph, using D2RQ and R2RML Parser (a tool that was introduced in [3]) with the same R2RML mapping file, and subsequently loading the RDF dump in the Virtuoso instance.

While the Virtuoso Universal Server supports R2RML mappings, the feature of viewing an external database as an RDF graph is available only in its commercial release, which was not available at the time of the tests. Therefore, it was not possible to test Virtuoso's R2RML views over the PostgreSQL DSpace schema. Instead, in order to measure Virtuoso's transient view performance, we had to dump the PostgreSQL database contents and load them into Virtuoso.

It is interesting to note that, in order to create and populate the experimental repositories with dummy data, bulk SQL insertions needed to be performed in the database. This is an operation that requires caution, since, unless care is taken, the required time could be unacceptable. Technically, this involved removing database table indexes and re-creating them at the end of the insertions.

Regarding Virtuoso, we noted that in order to execute complex queries on Virtuoso, using R2RML-based transient views, the program memory used (`MaxMemPoolSize` variable) had to be increased from 400M (default value) to 800M. We also noted that database caching, for some measurements, influences greatly the results while in other cases it seems to not have any impact at all. For instance, the SPARQL query Q2c (see Appendix II) on graph 1c using D2RQ, took 0.89 seconds, while subsequent calls took 0.33, 0.35, and 0.36 seconds, respectively. However, the time that was required to dump database contents into graph 2s using D2RQ, seems to be slightly, if at all, affected by caching, as it took 96.79s, 95.17s, 96.47s, and 93.89s, at consecutive executions. In either case, in this paper the measurements contain the average of several measurements, without counting the first one.

Table 1 below gathers the results regarding the time that was needed to export database contents as RDF graphs in (a) simple and (b) more complex mappings that will next be analyzed.

Table 1. Time taken to export database contents as RDF in cases of (a) simple mappings on the DSpace `eperson` table, and (b) more complex mappings containing many `JOIN` statements among many tables.

Users	triples	D2RQ	R2RML Parser
1k	3,004	14.52	3.30
10k	30,004	95.58	6.79
100k	300,004	906.26	25.06

(a)

Items	triples	D2RQ	R2RML Parser
1k	16,482	3.15	0.914
10k	159,840	28.96	7.732
100k	1,592,790	290.92	80.442

(b)

The first conclusion that can be read off these experimental measurements is the fact that dumping the contents of the DSpace database to an RDF graph takes much longer using D2RQ than using R2RML Parser. Therefore, when real-time access to the data is required, D2RQ is preferred, but in cases when dumps at time intervals suffice, the R2RML Parser tool is preferred. Of course, dumping the data into RDF, requires some time afterwards in order to load the graphs into Virtuoso, but as it is shown next, it is a small sacrifice considering the speed that it gives to queries.

3.2 Results regarding simple mappings

In order to measure behavior in simple settings, the mapping definition that was used targeted only the users that are stored in the DSpace installation (tables `eperson`, `epersongroup`, and `epersongroup2eperson`, the last one holding information about the many-to-many relationships among persons and groups). An excerpt of the mapping file is presented in Appendix I.

Table 1(a) shows the time it took to export the results into an RDF graph. After exporting the RDF graphs, three test cases were considered:

a. Transient views, using D2RQ over PostgreSQL, and an R2RML mapping
b. Persistent RDF views, using Virtuoso, over an RDF dump of the database according to the R2RML mapping
c. Transient views, using Virtuoso over its relational database backend, and an R2RML mapping.

In case *b*, the RDF graphs were loaded in the Virtuoso instance. The required time was *0.53*, *2.16*, and *19.12* seconds, respectively. In order to measure SPARQL performance, the queries presented in Appendix II were devised.

Table 2 below sums up the measurement results. As it can be observed from the query response times, in cases *a* and *c*, the most demanding query was Q2s, taking more than 1h to compute over graph 3s (containing 100k users). This behavior is due to the numerous (6) triple patterns in the graph pattern. Query Q1s also appeared to be demanding, taking 398.74 seconds to compute over graph 3s. However, none of these delays was observed in case *b* (persistent RDF view), in which the most demanding query was Q1s, taking 2.31 seconds to compute over graph 3s.

Table 2. Query response times, in seconds, in simple mapping settings.

	Graph 1s			**Graph 2s**			**Graph 3s**		
Q1s	6.18	0.1	0.56	44.75	0.31	0.88	398.74	2.31	3.8
Q2s	11.48	0.07	2310	11.76	0.08	3522	11.91	0.12	4358
Q3s	3.18	0.04	0.22	11.44	0.04	0.68	57.08	0.04	1.28
	a	*b*	*c*	*a*	*b*	*c*	*a*	*b*	*c*

3.3 Results regarding complex mappings

The second set of measurements was performed as follows: After populating the DSpace repositories with 1k, 10k, and 100k items, respectively, a mapping file was created, aiming at offering a view over the metadata values in the repository. This mapping file tends to become very complex since each mapping declaration can comprise results from 5 joined tables, a fact that is due to the highly normalized DSpace schema. Appendix I shows an excerpt of the mapping file, specifically the part that targets at the `dc.contributor.advisor` values.

Table 1(b) holds the time in seconds that was required to export the database contents as an RDF graph, using D2RQ, and R2RML Parser, over the same mapping file and relational database backend.

Subsequently, the resulting graphs were inserted in a Virtuoso instance. This process took *1.87*, *11.04*, and *201.03* seconds, respectively. Next, the three SPARQL queries that are presented in Appendix II were devised, in order to measure performance. Table 3 below concentrates the measurement results.

Table 3. Query response times, in seconds, in complex mappings.

	Graph 1c		**Graph 2c**		**Graph 3c**	
Q1c	125.34	0.27	1100.58	1.77	13921.64	11.18
Q2c	0.34	0.048	0.35	0.05	1.04	0.05
Q3c	144.01	0.13	1338.84	2.19	>6h	10.19
	D2RQ	*Virtuoso*	*D2RQ*	*Virtuoso*	*D2RQ*	*Virtuoso*

It must be noted that these queries in this experiment were evaluated in real-time against the D2RQ installation (transient views), and against the RDF graph dumps that were inserted in the Virtuoso instance (persistent). Although Virtuoso supports R2RML, it was not possible in this case to evaluate the queries against its R2RML implementation since it does not yet support the R2RML `rr:sqlQuery` construct that allows for arbitrary SQL queries to be posed against the database.

Overall, in the results table we can observe that Q3c was the most resource-hungry. Taking more than 20 minutes to compute over graph 2c, it was left overnight to compute over graph 3c, and was stopped since this amount of time was considered unacceptable, considering that the same query over the same graph in the persistent RDF view approach took 10.19 seconds to compute. Query Q2c was the fastest to compute at all times since it was not supposed to return any results. Query Q1c was more interesting since its graph pattern containing 6 triple patterns took approximately 3.87 hours for D2RQ to compute on graph 3c (containing 100k items), and 11.18 seconds for Virtuoso.

3.4 Discussion

Among the most important evaluation results are the ones visualized in Figure 1 below. From Figure 1(a), we can deduce that for queries Q1s and Q3s, query execution times increase as the size of the underlying graph increases, while query Q2s execution time remains more or less the same since it does not return any results. In Figure 1(b), in order to be objective in the measurements regarding Virtuoso performance, we added to the Q1c response time the time that was needed in order to dump into RDF the relational database contents, using R2RML Parser, and to subsequently load the RDF dump into Virtuoso. Also in this case, the execution time increases as the graph size increases, a fact that also holds for dumping the RDF using R2RML Parser, loading the dump into Virtuoso, and query Q1c answering over graphs 1c, 2c, and 3c in Virtuoso.

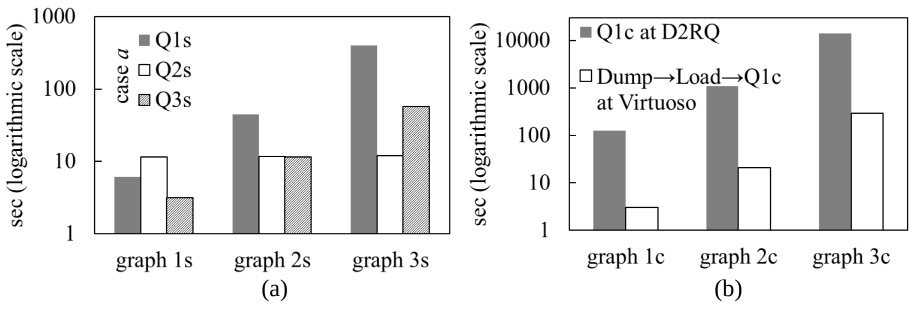

Fig. 1. In 1(a), we depict a query response time visualization in the simple mapping case *a*, while in 1(b) we visualize query Q1c execution time over D2RQ (transient RDF view) and over Virtuoso (persistent RDF view, after dumping the database contents using R2RML Parser, and loading the RDF dump into Virtuoso) in the complex mapping setting.

Overall, as it can be deduced from the experiment outcomes, in the case of digital repositories, real-time SPARQL-to-SQL conversions are not the optimal approach, despite the presence of database indexes that would presumably increase performance compared to plain RDF graphs. The round-trips to the database pose a burden that cannot be alleviated by relational database indexing techniques. RDF dumps perform much faster, especially in the cases of SPARQL queries that involve many triple patterns that are subsequently translated to numerous `JOIN` statements, which are usually expensive. Therefore, despite the advantages transient views demonstrate in the general case, in the case of digital repositories the additional computational burden they impose causes persistent views to be more preferable.

Regarding the initial time, required to export the database in an RDF graph, the R2RML parser concluded its export in much less time than D2RQ. Of course, this required the extra step of loading the RDF dump into Virtuoso, as illustrated and explained in Figure 1(b).

Overall, using Virtuoso with R2RML views enabled seems to be performing well; this solution however, comes at the expense of the following: R2RML transient views are only offered over Virtuoso's relational database backend, in the open-source version. Connection to external data sources is available only in the commercial Virtuoso edition. Moreover, no arbitrary SQL queries are supported as logical tables in the R2RML mapping file, thus diminishing mapping potential and capabilities.

4 Conclusions and Future Work

In this paper we present and evaluate an approach for exposing digital library information as LOD. After introducing the problem framework and examining several of the approaches that exist in the literature, we perform a set of measurements over two distinct approaches, and evaluate the measurement results. The first case concerns on-the-fly, transient RDF views over the relational database contents while the other case concerns querying asynchronous exports, i.e. persistent RDF dumps.

As it can be generally concluded from the measurements, querying RDF dumps instead of performing real-time round trips to the database is in general a more efficient approach. The answer was not clear beforehand, since, as one would expect. SPARQL-to-SQL translators can take into account indexes and database optimizations, but on the other hand, this translation in real-time is costly in terms of computational burden.

Simple as it may seem, on-the-fly SPARQL-to-SQL query translations is not a solution that will suit all environments and is not justified for every occasion. It would be advisable to prefer real-time query answering over transient RDF views when the data is subject to frequent changes, and less frequent queries (for example, such as in social networks). Cases such as institutional repositories and bibliographic archives in general are not typically updated to a significant amount daily, and selection queries over their contents are far more frequent than the updates.

The cost of not having real-time results may not be as critical, considering that RDF updates could take place in a manner similar to maintaining search indexes, typically used to enable full-text search in web applications. The trade-off in data freshness is largely remedied by the improvement in the query answering mechanism.

Also noteworthy is the fact that still, exporting data as RDF covers half of the requirements that have to be met before publishing repository data: the second half of the problem concerns its bibliographic dimension. Widespread ontologies have to be used where applicable in order to offer meaningful, semantically enriched descriptions of the DSpace repository data. Moreover, linking the data to third party datasets is an aspect that is not hereby discussed, as it is out of the scope of the paper. Overall, this paper's contribution is a methodology that offers an insight in the initial problems associated with the effort required to publish digital repository data as (Linked) Open Data, and the results one could expect.

Future steps that could be followed in order to expand this work include considering more mapping tools supporting R2RML (such as Ultrawrap (capsenta.com)), in order to evaluate dump times and query times. Additionally, more institutional repository solutions (such as Eprints (eprints.org)) or triple stores (such as Sesame (openrdf.org)) could be considered for inclusion in the measurements.

Acknowledgements. This work was partially funded by the Hellenic Academic E-Books project (http://www.kallipos.gr).

References

1. Villazon-Terrazas, B., Vila-Suero, D., Garijo, D., Vilches-Blazquez, L.M., Poveda-Villalon, M., Mora, J., Corcho, O., Gomez-Perez, A: Publishing Linked Data - There is no One-Size-Fits-All Formula. Proceedings of the European Data Forum (2012)
2. Spanos, D.-E., Stavrou, P., Mitrou, N.: Bringing relational databases into the semantic web: A survey. Semantic Web Journal, Vol. 3, No. 2 (2012) 169-209
3. Konstantinou, N., Spanos, D.-E., Houssos, N., Mitrou, N.: Exposing Scholarly Information as Linked Open Data: RDFizing DSpace contents. The Electronic Library (2013), in press
4. Auer, S, Dietzold, S., Lehmann, J., Hellmann, S., Aumueller, D.: Triplify – Light-Weight Linked Data Publication from Relational Databases. Proceedings of the 18th international conference on World Wide Web (WWW '09), New York, NY, USA (2009) 621-630
5. Chebotko, A., Lu, S., and Fotouhi, F.: Semantics Preserving SPARQL-to-SQL Translation. Data & Knowledge Engineering, Vol. 68, No. 10 (2009) 973–1000
6. Cyganiak, R.: A Relational Algebra for SPARQL, Technical Report HPL 2005-170 (2005)
7. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal On Semantic Web and Information Systems, Vol. 5, No. 2 (2009) 1–24
8. Bizer, C., Cyganiak, R.: D2R Server - Publishing Relational Databases on the Semantic Web. Proceedings of the 5th International Semantic Web Conference (2006)
9. Erling, O., Mikhailov, I.: RDF support in the Virtuoso DBMS. Proceedings of the 1st Conference on Social Semantic Web, Leipzig, Germany (2007) 59–68
10. Blakeley, C.: Virtuoso RDF Views Getting Started Guide. Available online at http://www.openlinksw.co.uk/virtuoso/Whitepapers/pdf/Virtuoso_SQL_to_RDF_Mapping.pdf (2007) accessed July 2nd 2013
11. Gray, A. J. G., Gray, N., and Ounis, I.: Can RDB2RDF Tools Feasibly Expose Large Science Archives for Data Integration?. Proceedings of the 6th European Semantic Web Conference (ESWC 2009), The Semantic Web: Research and Applications, LNCS Vol. 5554, Springer (2009) 491–505

12. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. W3C Recommendation. Available online at http://www.w3.org/TR/r2rml/ (2012) accessed August 29th 2013

Appendix I – Mapping file excerpts

Next, we provide the most important excerpts from the R2RML mapping files used during the experiments. In the simple mapping case, the declarations are as follows:

```
map:persons-groups
   rr:logicalTable [ rr:tableName '"epersongroup2eperson"'; ];
   rr:subjectMap [ rr:template
'http://data.example.org/repository/group/{"eperson_group_id"}';
];
   rr:predicateObjectMap [
      rr:predicate foaf:member;
      rr:objectMap [ rr:template
'http://data.example.org/repository/person/{"eperson_id"}';
                     rr:termType rr:IRI; ] ].
```

In the complex mapping case, the SQL queries get more complicated, as in the excerpt that follows:

```
map:dc-contributor-advisor
    rr:logicalTable <#dc-contributor-advisor-view>;
    rr:subjectMap [ rr:template
'http://data.example.org/repository/item/{"handle"}';
    ];
    rr:predicateObjectMap [
        rr:predicate dc:contributor;
        rr:objectMap [ rr:column '"text_value"' ]; ].

<#dc-contributor-advisor-view>
    rr:sqlQuery """
    SELECT h.handle AS handle, mv.text_value AS text_value
    FROM handle AS h, item AS i, metadatavalue AS mv,
metadataschemaregistry AS msr, metadatafieldregistry AS mfr
WHERE
    i.in_archive=TRUE AND
    h.resource_id=i.item_id AND
    h.resource_type_id=2 AND
    msr.metadata_schema_id=mfr.metadata_schema_id AND
    mfr.metadata_field_id=mv.metadata_field_id AND
    mv.text_value is not null AND
    i.item_id=mv.item_id AND
    msr.namespace='http://dublincore.org/documents/dcmi-terms/'
AND
    mfr.element='contributor' AND
    mfr.qualifier='advisor'
    """.
```

Appendix II – SPARQL queries

Appendix II concentrates the SPARQL queries that were executed against the mapping results, in order to conduct the measurements presented in Tables 1, 2 and 3.

Table 4. On the left, we provide the SPARQL queries Q1s, Q2s, and Q3s that were executed against simple mappings, and on the right the SPARQL queries Q1c, Q2c, and Q3c that were executed against more complex mappings.

Q1s	**Q1c**

`SELECT DISTINCT ?eperson ?name`
`WHERE {`
` ?eperson rdf:type`
`foaf:Person.`
`?eperson foaf:name ?name.`
`FILTER (?name != "mlo vqlbcbk"`
`)}`
`ORDER BY ?eperson`
`LIMIT 500`
	`SELECT DISTINCT ?item ?title`
	`?creator`
	`WHERE {`
	` ?item dcterms:title ?title.`
	` ?item dcterms:creator ?creator.`
	` ?item dcterms:identifier ?id .`
	` ?item dcterms:type ?type.`
	` ?item dcterms:subject ?subj.`
	` ?item dcterms:date ?date.`
	`FILTER (?date != "2008-06-`
	`20T00:00:00" )}`
	`ORDER BY ?creator`
	`LIMIT 100`

Q2s	**Q2c**

`SELECT DISTINCT ?eperson1`
`?groupname1 ?eperson2`
`?groupname2`
`WHERE {`
`?eperson1 rdf:type`
`foaf:Person.`
`?eperson2 rdf:type`
`foaf:Person.`
`?group1 foaf:member ?eperson1.`
`?group2 foaf:member ?eperson2.`
`?group1 rdf:type foaf:Group.`
`?group2 rdf:type foaf:Group.`
`OPTIONAL {`
`?group1 foaf:name ?groupname1.`
`?group2 foaf:name ?groupname2.`
`} } LIMIT 500`
	`SELECT DISTINCT ?item1 ?item2`
	`?creator1 ?type1 ?type2`
	`WHERE {`
	`?item1 dcterms:title "example".`
	`?item1 dcterms:creator ?creator1.`
	`?item1 dcterms:identifier ?id1.`
	`?item2 dcterms:title "example".`
	`?item2 dcterms:creator ?creator1.`
	`?item2 dcterms:identifier ?id2.`
	`OPTIONAL{`
	`?item1 dcterms:type ?type1.`
	`?item2 dcterms:type ?type2. } }`
	`ORDER BY ?creator1`
	`LIMIT 100`

Q3s	**Q3c**

`SELECT DISTINCT ?eperson`
`WHERE {`
`?group foaf:member ?eperson.`
`?group foaf:name`
`"Administrator".`
`?eperson foaf:name "john`
`smith"`
`} ORDER BY ?eperson`
	`SELECT DISTINCT ?item ?title`
	`?creator`
	`WHERE {`
	`?item dcterms:title ?title.`
	`?item dcterms:creator ?creator.`
	`?item dcterms:identifier ?id`
	`OPTIONAL{ ?item dcterms:type ?type }`
	`OPTIONAL{ ?item dcterms:subject`
	`?subj }`
	`OPTIONAL{ ?item dcterms:date ?date.`
	`FILTER (?date > "2008-06-`
	`20T00:00:00"^^<http://www.w3.org/200`
	`1/XMLSchema#dateTime> ) } } ORDER BY`
	`?creator`