An empirical performance evaluation of a semantic-based data retrieving process from RDBs & RDF data storages

Leandro Tabares Martín[1], Félix Oscar Fernández Peña[2], Amed Abel Leiva Mederos[3], Jyrki Nummenmaa[4]

[1] Universidad de las Ciencias Informáticas, Carretera a San Antonio de los Baños Km 2½, La Habana, Cuba.
[2] Universidad Técnica de Ambato, Ambato, Ecuador.
[3] Universidad Central "Marta Abreu" de las Villas, Villa Clara, Cuba.
[4] Universidad de Tampere, Tampere, Finlandia.

Autores para correspondencia: ltmartin@uci.cu

Fecha de recepción: 19 de junio del 2016 - Fecha de aceptación: 24 de julio del 2016

ABSTRACT

SQL and, more recently, SPARQL are standard languages of the software industry for retrieving data. Other studies showed that retrieving data by using a SPARQL query is much slower than the semantically equivalent SQL query. Nevertheless, some recent proposals optimized database servers for SPARQL queries. This paper presents the results of a comparative analysis between the capability of SQL and SPARQL with respect to the retrieval of data from a relational database and RDF-triples. A free and open source-based scenario was constructed by using PostgreSQL and Virtuoso for storing data, and *RETRI*, a data retrieving software built in JavaScript which displays data views in a specific XML format. Open data from the British National Library Bibliographic Data Set were used in the experiment; results were analyzed from a performance perspective.

Keywords: Data views, retrieving process performance, semantic web.

RESUMEN

SQL y más recientemente SPARQL, son lenguajes estándares para la recuperación de datos en la industria del software. Otros estudios han mostrado que recuperar datos utilizando consultas SPARQL es mucho más lento que su consulta semánticamente equivalente en SQL. Sin embargo, algunas propuestas recientes han optimizado servidores de bases de datos para consultas SPARQL. En este artículo se presentan los resultados de comparar SQL y SPARQL en cuanto a la recuperación de datos a partir de bases de datos relacionales y tripletas RDF respectivamente. Un escenario basado en software libre y de código abierto fue construido usando PostgreSQL y Virtuoso para almacenar datos y *RETRI*, un software para la recuperación de datos desarrollado en JavaScript, que muestra vistas de datos almacenadas en un formato XML específico. Se utilizaron datos abiertos del conjunto de datos de la Biblioteca Nacional Británica en el experimento y los resultados fueron analizados desde una perspectiva de rendimiento.

Palabras clave: Vistas de datos, rendimiento del proceso de recuperación, web semántica.

1. INTRODUCTION

Since Resource Description Framework (RDF) makes it possible to define the meaning of data in a machine readable form (Motik *et al.*, 2009), it seems that the semantic web technologies are helpful in the alignment of Relational Databases (RDB) towards the semantic dimension of data views manageability (Vavliakis *et al*., 2013). The evolution of RDF into Web Ontology Language (OWL) allows a richer semantic description based on Description Logics (Horrocks *et al*., 2003; Chen *et al*., 2015). OWL has been used in many specific scenarios for the construction of flexible data semantic models (Čerāns & Būmans 2011; Munir *et al*., 2012; Chen *et al*., 2015; Botoeva *et al*., 2016; Calvanese *et al*., 2016).

Interest in mapping relational data to RDF is increasing for the purpose of publishing linked data (Sudairy & Vasista, 2011; Bolchini *et al*., 2013; Chen *et al*., 2015; Lausch *et al*., 2015). In this direction, SPARQL is the World Wide Web Consortium (W3C) recommended query language for information retrieving from RDF documents. In the semantic web vision, SPARQL is considered the theoretical equivalent to SQL in RDB (Munir *et al*., 2012; Vavliakis *et al*., 2013).

Information retrieval problems have been discussed during several decades (van Rijsbergen, 1977; Bolchini *et al*., 2013; Lin *et al*., 2014; Chen *et al*., 2015; Gupta *et al*., 2015; Glavaš & Šnajder, 2014; Manning *et al*., 2009) and they are one of the open research areas nowadays (Gupta *et al*., 2015; Gupta & Bendersky, 2015; Janowicz *et al*., 2011). Different approaches are discussed on this area but a semantic-based approach is accepted as a need for information systems (Janowicz *et al*., 2011; Malhotra & Nair, 2015; Zhai, 2015). Meanwhile, there is a huge volume of structured data residing in traditional RDBs (Vavliakis *et al*., 2013; Bolchini *et al*., 2013; Zheng *et al*., 2013). Consequently, there is a research question we intend to answer: are RDF Management Systems performance able to compete with RDB Management Systems performance in current state of the art of data management technologies? In this paper, we present the results of comparing SQL and SPARQL for retrieving data from a relational database and RDF-triples, respectively, in order to generate customized data views. Relational data were stored in a PostgreSQL server and equivalent RDF-triples were stored in Virtuoso Open Source server. These servers were chosen taking into account their recognition in the community of free software.

The paper is organized as follows. First, the experiment scenario is described (see section 2). In section 3, we explain the theoretical foundation of the data retrieving process against RDB and RDF. In section 4, details of the proposed experiment and its results are discussed. Finally, conclusions and future work are analyzed in section 5.

2. DESCRIPTION OF THE EXPERIMENT SCENARIO

2.1. British National Library bibliographic data set

The British National Library (BNL) allows public access to the bibliographic data set structured according to the Linked Data Principles. The data set has been registered at "https://datahub.io/ dataset/bluk-bnb-basic" and contains the information created by librarians during the technical

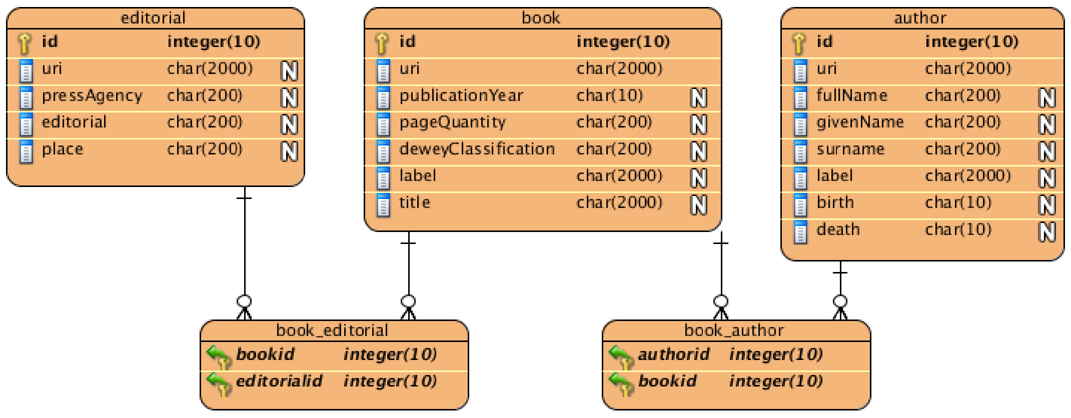


Figure 1. Design of database for storing the BNL Bibliographic Data Set Structure into a relational database.

processing of their bibliographic resources. The data has been structured according the International Federation of Library Associations (IFLA) standard ontologies and vocabularies such as Dublin Core. A full description of the data set structure is depicted on “http://www.bl.uk/bibliographic/pdfs/bldatamodelbook.pdf". For our experiment purposes a relational database was created in order to store the same information stored in the BNL data set. The database structure is depicted in Figure 1.

2.2. ViewOnto

ViewOnto is a web ontology for the formal description of heterogeneous data sources. Its goal is twofold: by instantiating *ViewOnto* 1) the semantics of concepts and the relationships among the conceptual representation of data is made explicit and 2) the semantic description of data is properly linked to the data source through the declaration of correctly defined syntactic statements for retrieving actual data (Fernández-Peña *et al*., 2016). Figure 2 graphically depicts the elements of *ViewOnto* used for describing the bibliographic data set of the BNL.

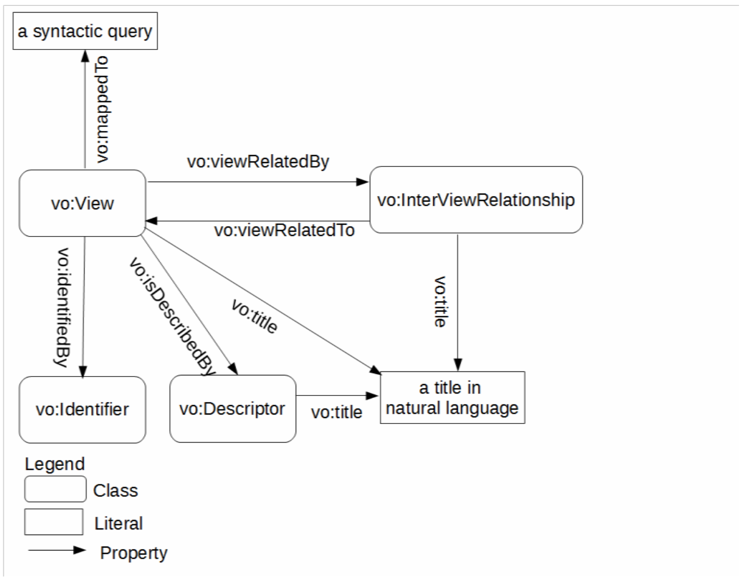


Figure 2. *ViewOnto* T-Box overview.

The BNL bibliographic data set contains information of the institutional catalog like authors, books, editorials and the relationships between them. The semantics of these concepts are described by instantiating *ViewOnto* on the specific scenario of the BNL bibliographic data set. *Author*, *Book* and *Editorial* are defined as instances of the class *View*. Each set of concept's data fields implies the definition of a corresponding object property *isDescribedBy*. The domain of the object property *isDescribedBy* is a semantically described instance of the class *Descriptor*. Meanwhile, the data property *title* is used for declaring a natural language sentence that describes *views* and *descriptors*. Finally, the data property *mappedTo* is used for establishing the liaison between the semantic layer of data (the conceptual definition of data) and the syntactic layer of data (the SQL/SPARQL query sentence for retrieving actual data from the datasource). The description of the BNB bibliographic data set created has fourteen individuals of *Descriptor* and three individuals of the class *View*. Figure 3 depicts a general overview of the instantiation of *ViewOnto* in the BNL scenario.

The *InterViewRelationship* allows to describe the existing relationships among data views. Each *InterViewRelationship* uses an object property named *viewRelatedTo* in order to represent the *View* instance it is related to. Figure 3 represents a partial instantiation of the data views and its properties for the use case of the British National Library Bibliographic Data Set. Every data view has a *title* and a *mappedTo* data type properties. The *mappedTo* property describes which is the SPARQL expression that allows to retrieve data of the related concept. Figure 4 shows the editing of the *mappedTo* property in Protégé, a versatile tool for editing web ontologies.

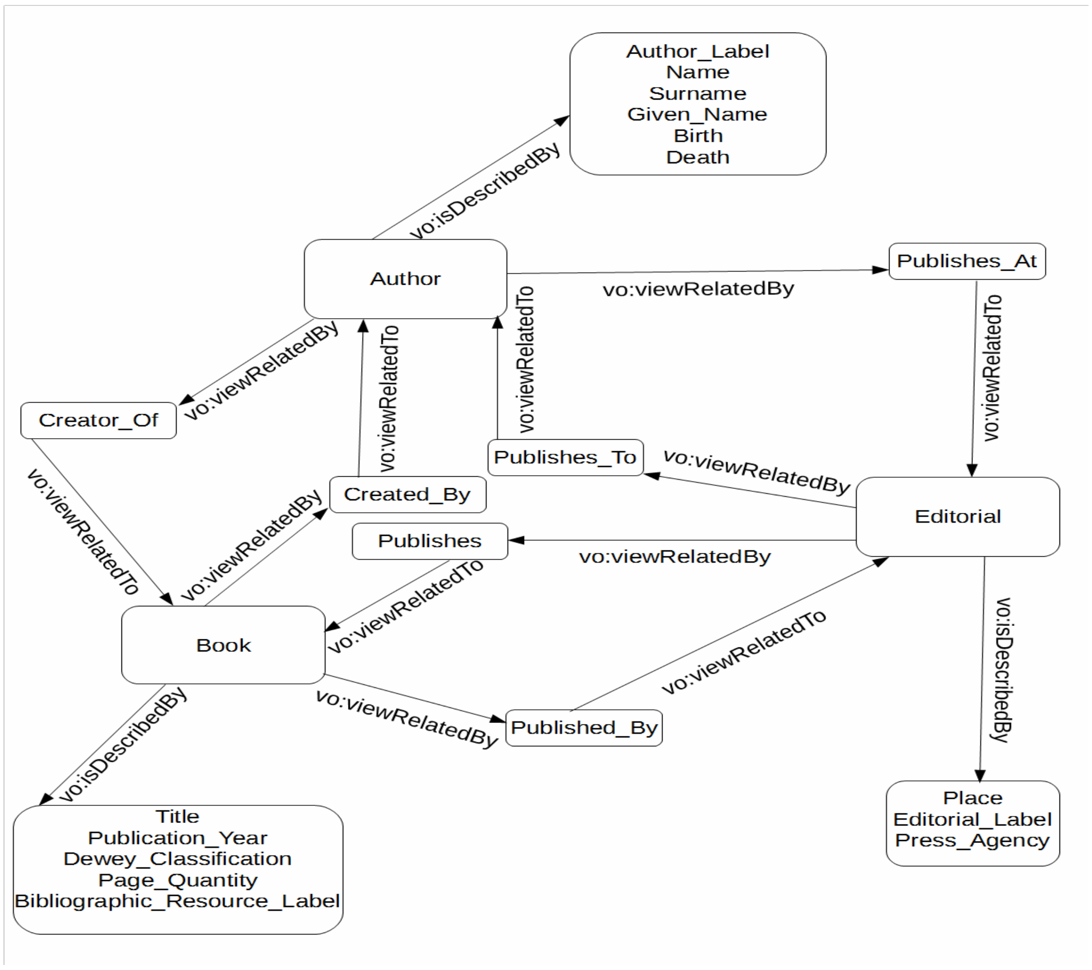
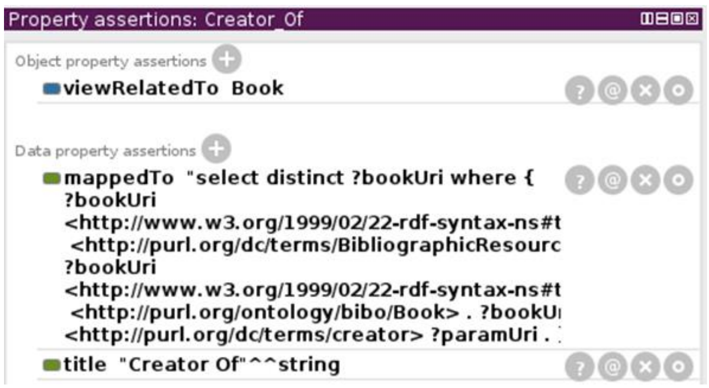


Figure 3. *ViewOnto* A-Box. Instantiating data views and the relationships among them.


Property assertions: Creator_Of

Object property assertions

viewRelatedTo Book

Data property assertions

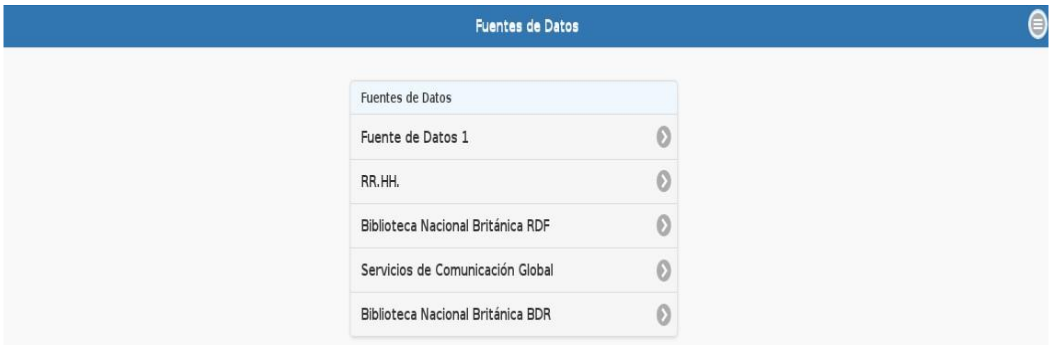
mappedTo "select distinct ?bookUri where {
?bookUri
<http://www.w3.org/1999/02/22-rdf-syntax-ns#t
<http://purl.org/dc/terms/BibliographicResourc
?bookUri
<http://www.w3.org/1999/02/22-rdf-syntax-ns#t
<http://purl.org/ontology/bibo/Book> . ?bookUı
<http://purl.org/dc/terms/creator> ?paramUri .

title "Creator Of"^^string

Figure 4. Individual of the *InterViewRelationship* class of *ViewOnto* named *Creator_of*.

2.3. RETRI

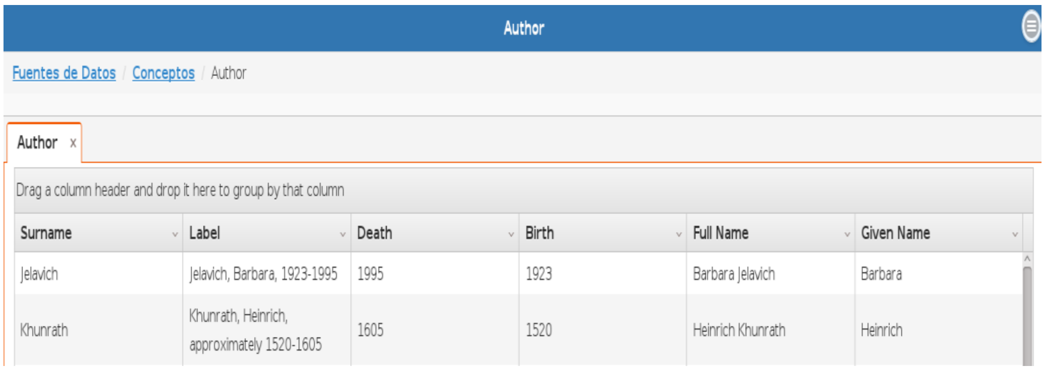
An application was developed for retrieving semantically described data by using *ViewOnto*. *RETRI* is a platform independent application that enables the user to browse data stored in heterogeneous data sources. The application was inspired by the foundations of context aware browsing formalized by Namiot (Namiot, 2012). A facet-based interface was defined for enabling user interaction with data. Once the user has log in, the user gets access to a list of available data sources (Fig. 5). The user chooses a data source and the list of predefined data views is deployed on the interface. By choosing any data view, a tabular view of data is deployed as depicted in Figure 6.



Fuentes de Datos

Fuentes de Datos
Fuente de Datos 1
RR.HH.
Biblioteca Nacional Británica RDF
Servicios de Comunicación Global
Biblioteca Nacional Británica BDR

Figure 5. *RETRI*'s data source selection user interface.



Author

Fuentes de Datos / Conceptos / Author

Author x

Drag a column header and drop it here to group by that column

Surname	Label	Death	Birth	Full Name	Given Name
Jelavich	Jelavich, Barbara, 1923-1995	1995	1923	Barbara Jelavich	Barbara
Khunrath	Khunrath, Heinrich, approximately 1520-1605	1605	1520	Heinrich Khunrath	Heinrich

Figure 6. Tabular view of data of *Authors* deployed in *RETRI*.

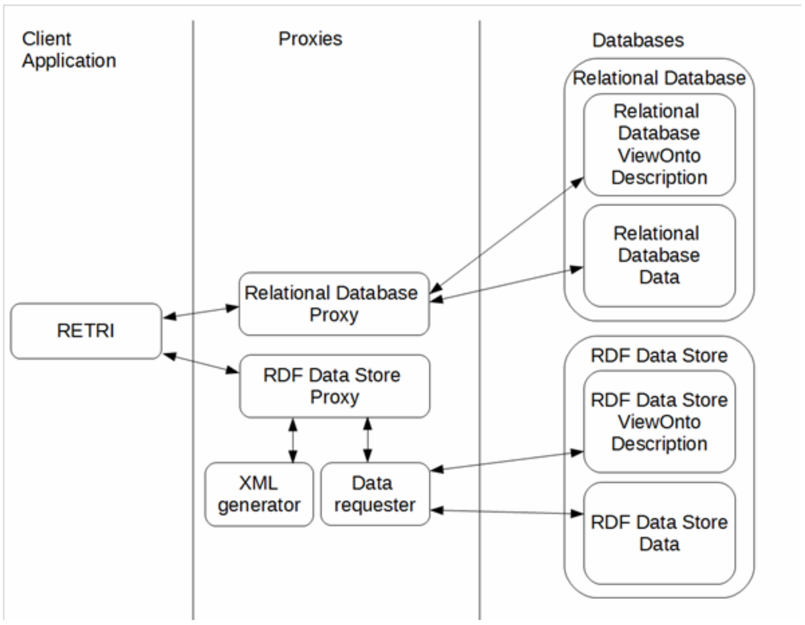


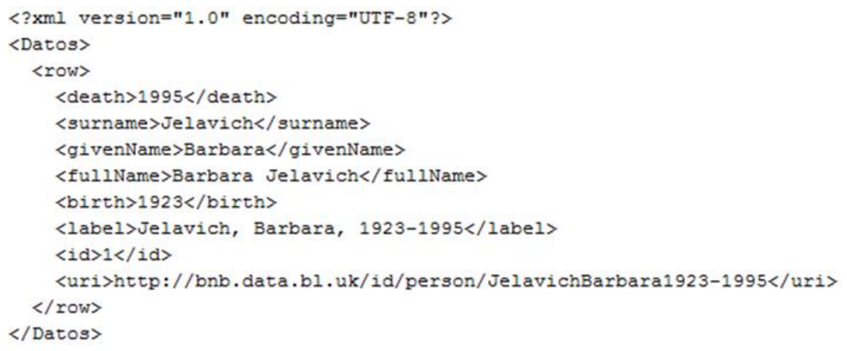
Figure 7. Three-layered architecture of *RETRI*.

3. DATA RETRIEVING MECHANISM

The data retrieving mechanism is based on a three-layers software architecture (as depicted in Fig. 7). The top layer of the architecture is composed by a 100% HTML/JavaScript-encoded user interface. Users are allowed to browse, reorder, filter and hide/show data. These functionalities were depicted as fundamental in a context-aware browsing tool (Namiot, 2012). This data is semantically described by instantiating *ViewOnto*. This application requests data to the middle layer (a proxy layer, actually) of the software architecture. The proxy layer exposes a set of web services translating the semantic information contained in the instances of *ViewOnto* into the data required by *RETRI*'s user interfaces. XML is used as the data language for data exchange. The proxy layer makes transparent the differences between data sources to *RETRI*. The bottom layer comprises two levels of data storage: the semantic description of data sources in instances of *ViewOnto* and the actual data sources in the corresponding RDB and RDF-based data sources.

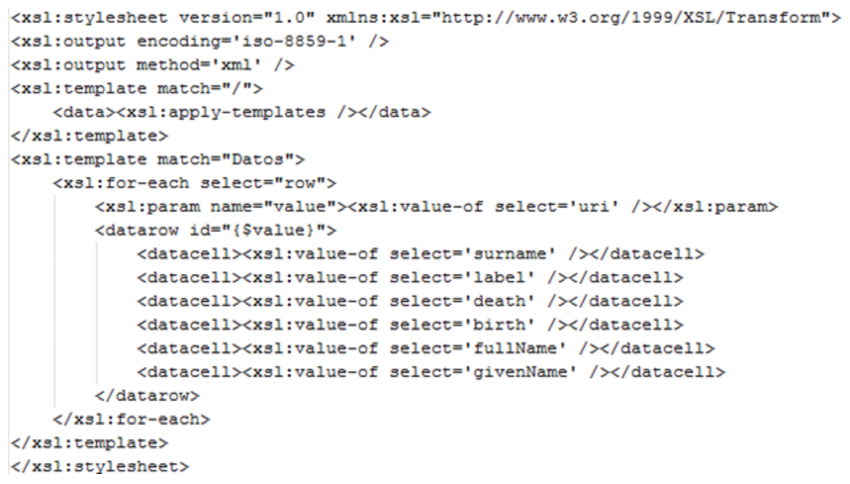
3.1. Retrieving data from relational databases

The process starts when a user chooses a RDB data source that has been previously described by instantiating *ViewOnto*. The process of describing data sources stored in relational databases includes the mapping of semantic data views to actual data by conforming a valid SQL query expression in the instances of the *mappedTo* property. A web service was designed as a proxy between the semantic and syntactic layers of language. When actual data is required for showing a semantic view to the end user, the web service 1) sends the SQL query instantiated in the *mappedTo* property to the database server, 2) transforms the response into an application-independent XML file as depicted in Figures 8 and 3) fits the response into a *RETRI*-understandable XML file by using an XSLT style sheet as depicted in Figure 9. The web service exposes four REST-based operations, as depicted in the activity diagrams of Figures 10 and 11.



```
<?xml version="1.0" encoding="UTF-8"?>
<Datos>
  <row>
    <death>1995</death>
    <surname>Jelavich</surname>
    <givenName>Barbara</givenName>
    <fullName>Barbara Jelavich</fullName>
    <birth>1923</birth>
    <label>Jelavich, Barbara, 1923-1995</label>
    <id>1</id>
    <uri>http://bnb.data.bl.uk/id/person/JelavichBarbara1923-1995</uri>
  </row>
</Datos>
```

Figure 8. Application-independent XML generated by the proxy.



```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding='iso-8859-1' />
<xsl:output method='xml' />
<xsl:template match="/">
    <data><xsl:apply-templates /></data>
</xsl:template>
<xsl:template match="Datos">
    <xsl:for-each select="row">
        <xsl:param name="value"><xsl:value-of select='uri' /></xsl:param>
        <datarow id="{$value}">
            <datacell><xsl:value-of select='surname' /></datacell>
            <datacell><xsl:value-of select='label' /></datacell>
            <datacell><xsl:value-of select='death' /></datacell>
            <datacell><xsl:value-of select='birth' /></datacell>
            <datacell><xsl:value-of select='fullName' /></datacell>
            <datacell><xsl:value-of select='givenName' /></datacell>
        </datarow>
    </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Figure 9. Style sheet used to transform the Author application-independent XML document into a XML understandable by RETRI.

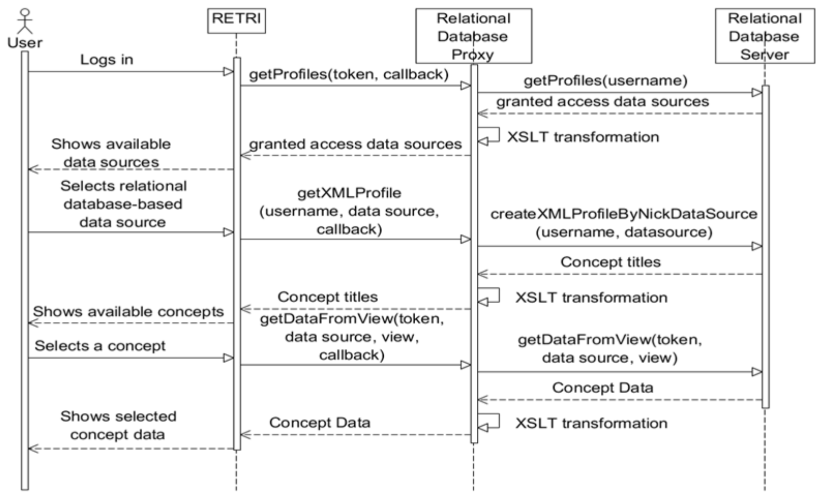

User
RETRI
Relational Database Proxy
Relational Database Server
Logs in
getProfiles(token, callback)
getProfiles(username)
granted access data sources
XSLT transformation
Shows available data sources
granted access data sources
Selects relational database-based data source
getXMLProfile (username, data source, callback)
createXMLProfileByNickDataSource (username, datasource)
Concept titles
Concept titles
XSLT transformation
Shows available concepts
getDataFromView(token, data source, view, callback)
Selects a concept
getDataFromView(token, data source, view)
Concept Data
Shows selected concept data
Concept Data
XSLT transformation

Figure 10. Semantic data retrieval process from a relational database.

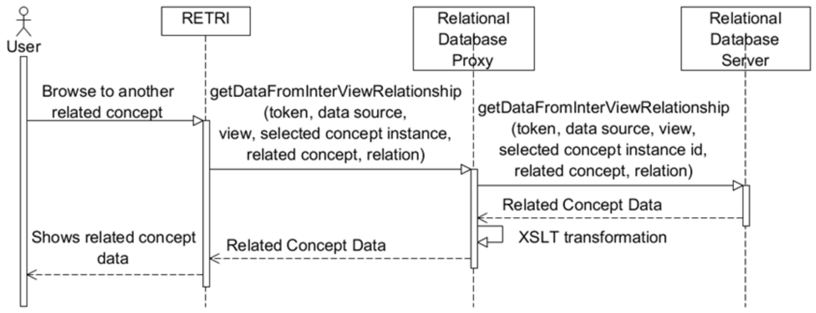

User
RETRI
Relational Database Proxy
Relational Database Server
Browse to another related concept
getDataFromInterViewRelationship (token, data source, view, selected concept instance, related concept, relation)
getDataFromInterViewRelationship (token, data source, view, selected concept instance id, related concept, relation)
Related Concept Data
XSLT transformation
Shows related concept data
Related Concept Data

Figure 11. Relational related concept data retrieval process.

3.2. *Retrieving data from rdf-based data stores*

Once a user chooses an RDF-based data source described in an instance of *ViewOnto*, *RETRI* generates a query to the proxy service for retrieving data from RDF-based data store (RDF proxy from now on).

A web service was designed as a proxy between the semantic and syntactic layers of language. When actual data is required for showing a semantic view to the end user, the web service 1) sends the SPARQL query instantiated in the *mappedTo* property to an SPARQL endpoint and 2) transforms the response into a *RETRI*-understandable XML file. The web service exposes four REST-based operations. All of the web service operations are interrelated in the process depicted in the activity diagrams of Figures 12 and 13.

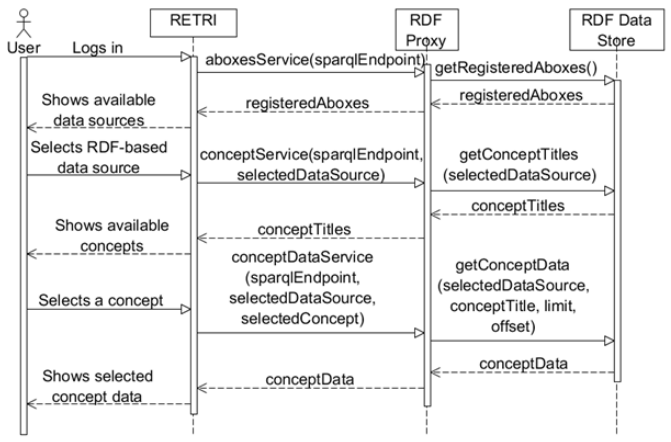

User
RETRI
RDF Proxy
RDF Data Store
Logs in
aboxesService(sparqlEndpoint)
getRegisteredAboxes()
registeredAboxes
Shows available data sources
registeredAboxes
Selects RDF-based data source
conceptService(sparqlEndpoint, selectedDataSource)
getConceptTitles (selectedDataSource)
conceptTitles
Shows available concepts
conceptTitles
conceptDataService (sparqlEndpoint, selectedDataSource, selectedConcept)
Selects a concept
getConceptData (selectedDataSource, conceptTitle, limit, offset)
conceptData
Shows selected concept data
conceptData

Figure 12. Semantic data retrieval process from RDF data store.

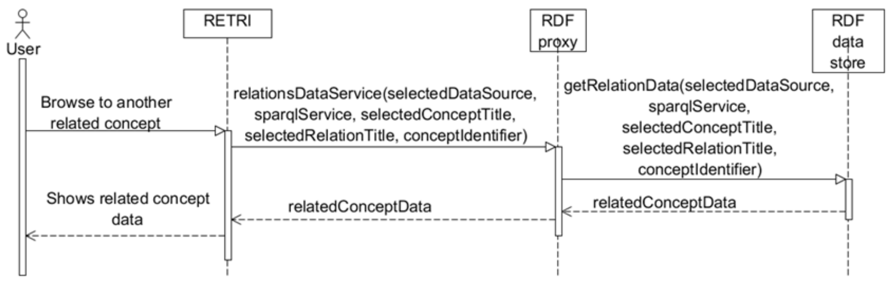

User
RETRI
RDF proxy
RDF data store
Browse to another related concept
relationsDataService(selectedDataSource, sparqlService, selectedConceptTitle, selectedRelationTitle, conceptIdentifier)
getRelationData(selectedDataSource, sparqlService, selectedConceptTitle, selectedRelationTitle, conceptIdentifier)
Shows related concept data
relatedConceptData
relatedConceptData

Figure 13. RDF Related concept data retrieval process.

4. EXPERIMENTS

4.1. Experiment design

An experiment was conducted in order to demonstrate if RDF Management Systems performance is able to compete with RDB Management Systems performance in current state of the art of data management technologies, taking into account the viability of the semantic description of heterogeneous data sources by using *ViewOnto*. The experiment was designed as follows.

- **Goal**: Analyzing, from a performance perspective, the process retrieving data stored on RDBs and RDF-based data sets.
- **Participants**: Fifty rounds retrieving data were carried out on each iteration of the experiment.
- **Research question**: Is the data retrieval process from RDB more efficient than from RDF data stores?
- **Experiment materials**: A computer with an Intel-core I5 2410 processor, 8 gigabytes of RAM and a hard disk of 5400 RPM was used during the experiment. The operating system was OpenSuse 42.1, the relational database used was PostgreSQL version 9.4.1 and the RDF data store used was Virtuoso Open Source version 7.2.1. Apache 2.4 and Tomcat 8.0.28 were used as web servers.
- **Tasks**: During each round of the experiment, *RETRI* was used for retrieving the data view corresponding to the concepts *Book*, *Editorial* and *Author* from RDB and RDF graphs. Diversifying the nature of retrieved data is suggested by Lin (Lin *et al*., 2014) for this kind of experiments. Four different RDB data sets and the semantically equivalent data sets in RDF (eight data sets in total) were used in each round. The main difference between those data sets is the amount of records they contain. The amount of records in each case is equivalent to 500, 1K, 1.5K and 2K. Retrieving more than 2K records at once is not considered to be a good practice; a sort of pagination mechanism should be used if more than this amount of records is going to be retrieved.
- **Hypothesis**: The semantics-based data retrieving from an RDB is faster than data retrieving from RDF graphs.
- **Variable**: Response time.

4.2. Experiment results

In each iteration of the experiment, fifty queries were carried out using the Graphic User Interface of *RETRI*. Response time for each query processing, XML adjustment and browser loading processes were tabulated. Resulting data allowed us to determine the influence of each phase in the efficiency of the whole process. Next, table 1 illustrates the mean value of response time for the processes of querying data, adjusting data into the proper XML format -understandable by *RETRI*- and the browser response time -total time from when *RETRI* sends the data request until the moment in which the response is fully deployed on the canvas of the user interface.

Standard deviation was calculated for validating experiment results. Thirty, of a total of one thousand and two hundred observations, were excluded as the distance from those observations and the mean value was longer than expected.

Table 1. Measurement of the time in the processes studied during the experiment.

Iteration number	1		2		3		4	
	Time	σ	Time	σ	Time	σ	Time	σ
Query time for Author concept over a relational database								
Query (ms)	11.44	2.12	21.74	3.57	32.18	5.6	40.5	4.33
XML	13738.04	217.57	54164.16	1323.1	119932.61	3390.67	224669.35	4160.17
Response time (s)	15.51	0.27	56.24	1.33	122.12	3.38	227.44	4.46
Query time for Author concept over a RDF data storage								
Query (ms)	180.3	26.48	212.12	32.70	261.12	40.14	324.26	56.39
XML	5.28	1.08	8.82	1.83	10.92	1.42	15.2	2.08
Response time (s)	2.03	0.18	2.22	0.20	2.54	0.23	2.66	0.17
Query time for Book concept over a relational database								
Query (ms)	15.26	2.73	28.46	4.47	41.18	5.46	54.15	6.85
XML	14184.54	277.30	56956.44	1609.19	126135.55	4142.45	227707.91	10202.14
Response time (s)	16.30	0.36	59.28	1.86	128.62	4.18	230.45	10.65
Query time for Book concept over a RDF data storage								
Query (ms)	275.94	41.33	330.78	59.37	488.32	85.63	445.54	37.25
XML	4.19	0.87	6.62	1.28	9.36	1.41	12.02	1.71
Response time (s)	2.42	0.32	3.40	0.45	4.14	0.64	2.71	0.19
Query time for Editorial concept over a relational database								
Query (ms)	2.69	0.71	3.90	0.74	6	1.20	7.83	1.59
XML	3312.45	115.57	14436.54	819.37	32251.85	1289.01	56427.43	1145.60
Response time (s)	5.20	0.17	17.36	1.11	35.28	1.59	58.59	1.24
Query time for Editorial concept over a RDF data storage								
Query (ms)	111.33	16.01	147	23.77	237.92	37.73	290.52	49.14
XML	4.51	0.93	5.85	0.85	9.08	1.55	11.16	1.72
Response time (s)	1.92	0.14	3.10	0.45	2.24	0.19	2.63	0.25

XSLT has been used in other proposals like *Expertus* (Jayasinghe *et al*., 2013). Then, we decided to corroborate the throughput of XSLT in the transformation of XML documents. Two different methods were used for adjusting data into the XML schema that *RETRI* is able to read: using XSLT transformations and a Java API for working with XML files. With the former method, the generic XML containing the data from the RDB is adjusted into the XML format that *RETRI* understands. With the latter method, the record set retrieved from Virtuoso was read and its data migrated to the proper XML file.

Querying structured data from a PostgreSQL server showed to be less time consuming than retrieving data from a Virtuoso server. The response time variable responds to a linear function in PostgreSQL and to an exponential function in Virtuoso, as shown in Figure 14. However, when analyzing the time taken by the process of adjusting data into the proper XML format, the outcome was not what we expected. The process of adjusting data from one XML format into another one by using XSLT is considerably more time consuming than generating an XML file with a Java API, as depicted in Figure 15.

Figure 16 depicts that the total response time when bringing data into the application canvas is more time consuming when working with the XSLT transformer for generating the proper XML. Adjusting data into the appropriate XML format -understandable by *RETRI*- was actually the most expensive task in the scenario of experimentation. Against what we expected, the total response time of retrieving data from the Virtuoso server had better outcomes than retrieving data from the PostgreSQL server.

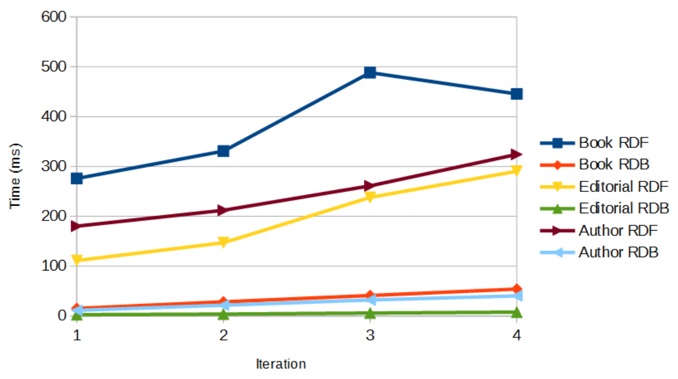


Figure 14. Average time measured when querying data.

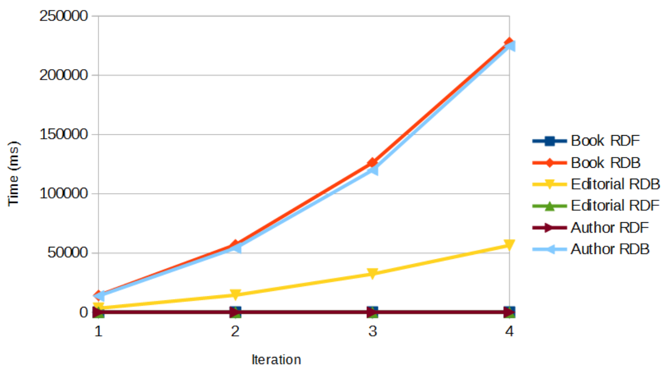


Figure 15. Average time measured during the adjustment of data into the XML format.

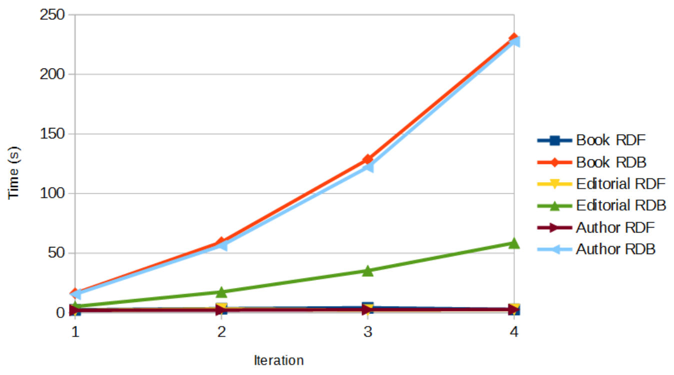


Figure 16. Average total response time.

The XSLT-based XML adjustment process took the 95.46, 95 and 96.31 percent of the whole time for retrieving data from the relational databases of Author, Book and Editorial, respectively. However, retrieving data from the equivalent RDF data storages took 0.41, 0.26 and 0.31 percent of the whole time spent for retrieving data. Making the same analysis from a different point of view, we may say that when the volume of data increases, the duration of the XSLT-based XML adjustment process showed an exponential growth while the duration of the equivalent Java-based XML adjustment process showed a linear growth, as depicted in Figure 15. Using a Java API showed up the best performance and its efficiency did not decay when adjusting big XML data.

In brief, the experiment showed up that PostgreSQL keeps on better efficiency for retrieving data in comparison to Virtuoso. However, the most expensive process is not retrieving data but adjusting data into the proper XML format of the evaluated scenario and it was not XSLT but the Java API what showed the best outcomes in adjusting data.

5. CONCLUSIONS AND FUTURE WORK

The experiment that we conducted corroborates that until today, a PostgreSQL server keeps been more efficient than a Virtuoso Server while working, in similar scenarios, with the same data, stored in a RDB and RDF files, respectively. This result was actually expected from a theoretical point of view. Nevertheless, it is important to note that the semantic enrichment of the data management in Virtuoso, with the support of SPARQL queries, and its ability to manage data residing in RDF documents, increases the flexibility of semantic-based data retrieving processes.

On the other hand, the experiment showed up that retrieving data is by far not as expensive as the process of adjusting the resulting data into a specific XML format. Time spent on the adjusting of XML documents by using XSLT grows on exponential bases with the increase of the amount of data records while it grows on linear bases when a Java-API is used. Since XML is the standard defacto for the interoperability of applications, adjusting data into an XML format is actually a critical process. Storing RDF documents on a Virtuoso server and using a Java-API for generating XML in the output allowed us to exploit the semantic enrichment of Virtuoso without affecting the efficiency of the whole data retrieving process.

As a final result, we may say that the best results with the semantic-based data retrieving process deployed by using *RETRI* were attained when using Virtuoso as the database server and a Java-API, in the proxy layer, for generating the XML format. Nevertheless, it is important to note that, even when obtained results mean a step forward in the area of open source software for managing semantically enriched structured data sources, these results are not conclusive due to the diverse nature of the variables involved in a data retrieval process. In the future work, we will study the efficiency of Virtuoso for retrieving data from a *ViewOnto* instance in order to estimate the entropy of data in a semantic-based data retrieving process. Also the performance of XSLT-based data transformations will be deeper studied as well as Java-API-based transformations in order to identify their strength use cases.

REFERENCES

Al-Sudairy, M.T., T. Vasista, 2011. Semantic data integration approaches for e-governance. *International Journal of Web & Semantic Technology*, 2(1).

Bolchini, C., E. Quintarelli, L. Tanca, 2013. CARVE: Context-aware automatic view definition over relational databases. *Information Systems*, 38(1), 45-67.

Botoeva, E., D. Calvanes, B. Cogrel, M. Rezk, G. Xiao, 2016. *OBDA Beyond Relational DBs: A Study for MongoDB*. In: Lenzerini, M., R. Peñaloza (Eds.). International Workshop on Description Logics. Cape Town, South Africa: CEUR Workshop Proceedings. Available at: http://ceur-ws.org/Vol-1577/paper_40.pdf.

Calvanese, D., P. Liuzzo, A. Mosca, J. Remesal, M. Rezk, G. Rull, 2016. Ontology-based data integration in EPNet: Production and distribution of food during the Roman Empire. *Engineering Applications of Artificial Intelligence*, pp.1-18. Available at: http://linkinghub.elsevier.com/ retrieve/pii/S0952197616000099.

Čerāns, K., G. Bümans, 2011. RDB2OWL: a RDB-to-RDF / OWL Mapping Specification Language. *Information Systems*, 139-152.

Chen, C., G. Zhao, Y. Yu, H. Deng, 2015. Multiple views system to support awareness for cooperative design. *Computer-Aided Design*, 63, 39-51.

Fernández-Peña, F., P. Urrutia-Urrutia, R.A. Cañete Bajuelo, J. Nummenmaa, 2016. A conceptual data model for the automatic generation of data views. *Applied Mathematics & Information Sciences*, 10(4), 1331-1342.

Glavaš, G., J. Šnajder, 2014. Expert systems with applications event graphs for information retrieval and multi-document summarization. *Expert Systems with Applications*, 41(15), 6904-6916.

Gupta, M., M. Bendersky, 2015. *Information retrieval with verbose queries*. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, pp. 1121-1124.

Gupta, Y., A. Saini, A.K. Saxena, 2015. Expert systems with applications: a new fuzzy logic based ranking function for efficient information Retrieval system. *Expert Systems with Applications*, 42(3), 1223-1234.

Horrocks, I., P. Patel-Schneider, F. van Harmelen, 2003. From SHIQ and RDF to OWL: The Making of a Web Ontology Language - Google Search. *Journal of Web Semantics*, 1(1), 7-26.

Janowicz, K., M. Raubal, W. Kuhn, 2011. The semantics of similarity in geographic information retrieval. *Journal of Spatial Information Science*, 2011(2), 29-57.

Jayasinghe, D., J. Kimball, T. Zhu, S. Choudhary, C. Pu, 2013. *An infrastructure for automating large-scale performance studies and data processing*. Available at http://www.istc-cc.cmu.edu/ publications/ papers/2013/bigdata.pdf, pp. 7.

Lausch, A., Schmidt, A. & Tischendorf, L., 2015. Data mining and linked open data - New perspectives for data analysis in environmental research. *Ecological Modelling*, 295, 5-17.

Lin, Y., C. Cole, K. Dalkir, 2014. The relationship between perceived value and information source use during KM strategic decision-making: A study of 17 Chinese business managers. *Information Processing & Management*, 50(1), 156-174.

Malhotra, M., T.G. Nair, 2015. Evolution of knowledge representation and retrieval techniques. *Intelligent Systems and Applications*, 7(7), 18-28.

Manning, C.D., P. Raghavan, H. Shütze, 2009. *Introduction to information retrieval*, Cambridge University Press.

Motik, B., I. Horrocks, U. Sattler, 2009. Bridging the gap between OWL and relational databases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2), 74-89.

Munir, K., M. Odeh, R. McClatchey, 2012. Ontology-driven relational query formulation using the semantic and assertional capabilities of OWL-DL. *Knowledge-Based Systems*, 35, 144-159.

Namiot, D., 2012. *Context-Aware browsing: A practical approach*. In: 6th International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST). IEEE, pp. 18-23.

van Rijsbergen, C.J., 1977. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of documentation*, 33(2), 106-119.

Vavliakis, K.N., T.K. Grollios, P.A. Mitkas, 2013. RDOTE - Publishing relational databases into the semantic web. *The Journal of Systems & Software*, 86(1), 89-99.

Zhai, C., 2015. *Towards a game-theoretic framework for information retrieval*. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 543.

Zheng, Q., Z. Wu, X. Cheng, L. Jiang, J. Liu, 2013. Learning to crawl deep web. *Information Systems*, 38(6), 801-819.