DESIGN AND IMPLEMENTATION OF THE GRAFFITI RDB2RDF DATA ACCESS SOLUTION

[1]Vishniakov V.A.; [2]Borodaenko D.S.; [3]Borodaenko J.V.
[1]Management department
Minsk Management Institute,
[2]EPAM,
[3]Economic department
Belorussian State University of Informatics and Radioelectronics
Minsk, The Republic of Belarus
e-mail: vish2002@list.ru, angdraug@gmail.com, jborodaenko@mail.ru

***Abstract* —The paper explores in detail the architecture and implementation considerations of the Graffiti relational RDF storage system. Requirements that are specific to systems interfacing with legacy relational databases are analysed along with those common to all RDF storage systems.**

Keywords: Graffiti; RDF storage system; Web applications

I. INTRODUCTION

The primary purpose of the RDF technology to present data available on the Web in a manner that facilitates automated processing [1]. This purpose defines the first requirement for an RDF store: compatibility with operating systems and programming languages that are widely used for deployment and development of Web applications.

Another universal requirement is absence of limitations on distribution and code reuse, particularly dependencies on proprietary commercially-licensed software. The most direct approach to satisfying this requirement is to use free and open source software (FOSS) exclusively.

II. OS AND DB

Times Roman According to the 2011 survey by W3Techs [2], Linux is used by more active web sites than any other OS, and dominates other open source systems by at least an order of magnitude. Ruby has not gained similar level of prominence among other programming languages on the Web, but it has other benefits over competing platforms, such as clear and easy to read syntax and a very high level of abstraction, which is an important factor in reducing the development effort. Ruby's credibility is also strengthened by its use by high profile sites such as Twitter.

The most popular open source relational databases are PostgreSQL, MySQL, and SQLite. Since one of the biggest use cases for relational to RDF query translation is to provide semantic access to legacy data, where the choice of DBMS has already been made, support for all three choices is required. While PostgreSQL allows to implement stored procedures in a variety of programming languages, MySQL and SQLite are limited to PL/SQL.

III. GRAFFITI RDF

The above requirements allowed to define the following development platform for Graffiti RDF store: Linux operating system; Ruby programming language; PostgreSQL, MySQL, and SQLite relational database management systems; PL/SQL stored procedures.

Graffiti RDF store is implemented in the form of a Ruby library providing an API for performing RDF queries in Squish query language. Following code fragment demonstrates the use of Graffiti in a Ruby application:

```
db = Sequel.connect(:adapter => 'pg', :database =>
dbname)
config = File.open('rdf.yaml') {|f| YAML.load(f.read)
}
store = Graffiti::Store.new(db, config)
data = store.fetch(%{
SELECT ?date, ?title
WHERE (dc::date ?r ?date FILTER ?date >= :start)
(dc::title ?r ?title)
ORDER BY ?date DESC}, 10, 0, :start => Time.now
- 24*3600)
```

A full deployment of an RDF application based on Graffiti typically includes one or more application servers hosting the application itself, Graffiti library, and Sequel library used for interfacing with relational databases; single cache server running an instance of a distributed synchronous cache system SynCache; one or more database servers hosting the relational database and Graffiti-specific stored procedures.

Graffiti library is implemented in the following classes, presented on diagram 1: Store provides the RDF store API; RdfConfig implements configuration for mapping relational schema to RDF model; SquishQuery and its subclasses SquishSelect and SquishAssert are responsible for parsing and executing queries and assertions written in Squish query language; SqlMapper and helper classes SqlExpression and SqlNodeBinding implement translation of Squish queries into SQL in line with the configuration provided by RdfConfig.

<<graffiti-classes.png>>

Diagram 1. Class diagram of the Graffiti library

Query translation performed by SqlMapper class relies heavily on logical inference implemented on the database level in form of stored procedures. Graffiti RDF store supports entailment rules for the following properties from the RDFS and OWL vocabularies: rdfs:subClassOf, rdfs:subPropertyOf, owl:TransitiveProperty.

Stored procedures implementing entailment rules for rdfs:subClassOf are triggered on every insert and delete for the subclass table. When a tuple is inserted with primary key not specified, a template tuple is inserted into the superclass table, and primary key value is copied from superclass table to subclass table. Delete operations are cascaded to all linked subclass and superclass tables.

Inference for rdfs:subPropertyOf is done at the query translation stage and relies on a stored procedure that returns value of an attribute when special sub property mapping attribute is set, and NULL when it's unset. This allows a translated query to select only tupelos that have sub property attribute values matching the sub property specified in the RDF query.

Inference for owl:TransitiveProperty relies on a separate transitive closure table for every attribute mapped to a transitive property. A set of stored procedures are triggered on insert, update, and delete operation to keep the transitive closure up to date with the base table.

IV. Software architecture

The software architecture outlined above provides Graffiti with the following unique advantages. Using Ruby API allows the application and Graffiti to exchange data within a single process, avoiding the latencies and overhead involved in communicating over a network protocol. It also
allows to use parameterised queries which improves the hit rate on the cache of translated queries. In the same time, query translation within the application shifts part of the load away from the database, and in distributed deployments application usually scales much better than the database [4, 5].

Implementation of logical inference on the database level minimizes the number of interactions between Graffiti and RDBMS that are required to complete a single RDF query or assertion, which has positive impact both on performance and ACID compliance of the whole system.

The above in combination with the high level of abstraction provided by the Ruby programming language has allowed to implement the whole RDF store in only 1000 lines of Ruby code and 200 lines of PL/SQL. The small size of the code base makes the ongoing maintenance and development require less effort than the existing systems counting hundreds of thousands of lines of code in lower-level Java and C++ programming languages. More details about this solution can find in monographic [4].

V. Realisation

The open publication messages system based on the RDF semantic data model and used RDF storage Graffiti as main data accesses tool has been realised. The use of model RDF has simplified the data change with other applications. It has allowed to realize full decentralise process of site content structure.

The investigation of RDF Graffiti functional characteristics and productivity are shown that this system surpasses on their possibilities the majority of such existing systems reflecting relation data to RDF.

The Graffiti system advantage before the best analogy system Virtuoso RDF Views includes the more high flexibility (supporting reunification of RDF ratification and compact module architecture) and scalability (supporting for the use of logical inference procedures keeping and the possibility of distribution calculation on RDF requests processing).

The storage system RDF-data Graffiti and system open publication Samizdat ware used in electronic publication process of newspaper “Computer vesti”. It allows to high of material moderation efficiently publishing by site users. Among this to high the efficiently of electronic newspaper version for short the working time of informatics support in number click calculation.

The semantic search and analyzing of corporative information module has been realized and introduced in the decision making system of one Byelorussian IT company on the base of RDF storage Graffiti. This module has been provided the company chief management operational accesses to various application data without the staff working time expenditure on information sanding from uncoordinated programs.

The investigation results have been used in the Minsk Management Institute management department study process. It has allowed the study process on discipline “Artificial management systems” make more quality owing to the forming by students the study material system understanding and the receiving practical habit in area semantic structure RDF simulation [6].

[1] Ermolayev, V. Towards a Framework for Agent-Enabled Semantic Web Service Composition / V. Ermolayev, N. Keberle, S. Plaksin // International Journal of Web Services Research. – 2004. – Vol. 1, No. 3. – P. 63-87.

[2] 2. Usage of operating systems for websites [Electronic resource] / W3Techs, August 2011. - Mode of access: http://w3techs.com/technologies/overview/operating_system/all. Date of access: 14.05.2012.

[3] 3. Graffiti RDF Store [Electronic resource] / D. Borodaenko – Semantic Future, December 2011. - Mode of access:http://semanticfuture.net/index.php/Graffiti. Date of access: 14.05.2012.

[4] 4. Vishniakov V.A., Borodaenko D.S., Borodaenko J.V.. Models and Tools of Integration of Applications, Marketing, Outsourcing, Knowledge Processing in Computer Nets. Minsk. MIM, 2011. – 350pp.

[5] 5. Borodaenko D.S Functional particularities and productivity of the storage RDF data system in relational DBMS // BSUIR Reports.-.2010. - № 4(50). - P. 95-99

[6] Borodaenko D.S., Vishniakov V.A. The perspectives of RDF technologies use in the net of universities // Informatization of Education. – 2010. - № 4. – P. 44-53.