Dominik Tomaszuk
University of Bialystok

MANAGING A GRAPH STORE IN THE REST ARCHITECTURAL STYLE

Abstract. This paper describes a simple method of communication between graph store and client over a web. We propose a mechanism based on the Hypertext Transfer Protocol standard. It can be used to express, select, and update operations across various Resource Description Framework (RDF) data sources. We present a fast method that suffices clients to know only Uniform Resource Locator based on Representational State Transfer (REST).

Keywords: Semantic Web, Resource Description Framework (RDF), Representational State Transfer (REST), Graph store, Query languages

1. Introduction

A graph store is a purpose-built database for the storage and retrieval of Resource Description Framework (RDF) data. Much like in the case of other databases, one can find and modify data in graph store via web services.

Following [1], let I be the set of all Internationalized Resource Identifier (IRI) references, B an infinite set of blank nodes, L the set RDF plain literals, and D the set of all RDF typed literals. I, B, L and D are pairwise disjoint. Let $O = I \cup B \cup L \cup D$ and $S = I \cup B$. An RDF triple T is a triple in $S \times I \times O$. If $T = (s, p, o)$ is RDF triple, s is called the subject, p the predicate and o the object. An RDF graph G is a set of RDF triples T. It is collection, which is represented by a labeled, directed multigraph. An RDF graph store GS is a set $\{G_0, (u_1, G_1), (u_2, G_2), \ldots, (u_n, G_n)\}$, where each G_i is a RDF graph and u_i is an IRI reference. Each IRI is distinct. G_0 is called a default graph and each pair (u_i, G_i) is called a named graph [2]. An RDF store should have one default graph and zero or more named graphs.

RDF graph store providers do not have any explicit way to express any intention concerning access to data. In the paper we attempt to define the proper methods to find and modify the RDF data in a graph store with web

ISBN 978–83–7431–362–9 ISSN 0860-150X

service means. In this paper a new architectural style access to graph stores based on Representational State Transfer (REST) [3] is presented.

The paper is constructed as follows. Section 2 is devoted to related work. In Section 3, we propose a flexible solution for managing RDF graph store data. The paper ends with conclusions.

2. Related Work

Classical web services tools were focused on Remote Procedure Call (RPC), and as a result this style is widely supported. Unfortunately, it is often implemented by mapping services directly to language-specific functions calls.

The most popular protocol based on RPC style is XML-RPC [4]. It uses Extensible Markup Language (XML) in request body and response returned values. An XML-RPC message uses POST Hypertext Transfer Protocol (HTTP) method. Another RPC approach is JSON-RPC [5]. Its requests and responses are encoded in JavaScript Object Notation (JSON). A remote method can be also invoked by sending a request to a remote service using sockets or HTTP protocol.

Simple Object Access Protocol (SOAP) [6] is the successor of XML-RPC. SOAP is a protocol for exchanging structured information in the implementation of various web services. It relies on XML for its message format, and usually relies on HTTP, for message negotiation and transmission. It is also based on RPC style. SOAP has become the underlying layer of a more complex set of web services, based on Web Services Description Language (WSDL). Unfortunately, SOAP is very complex.

In the context of Semantic Web, there is also a new proposal for query processing and returning the query results service [7]. It describes a means of conveying SPARQL queries and updates from clients to query processors. SPARQL Protocol is described as an HTTP binding of abstract interface. It uses WSDL to describe a means for conveying SPARQL queries. Unfortunately, this protocol is dedicated only for graph stores which use SPARQL query language. Furthermore, this SOAP-based protocol is complex and can be significantly slower, because it usese verbose XML syntax. Additionally, this protocol disregards many of HTTP's existing capabilities such as: authentication, caching and content type negotiation. In contrast, here we do not use RPC style. We concentrate on defining mechanisms strictly dedicated to web graph stores, preserving the feature of using IRIs, Multipurpose Internet Mail Extensions (MIME) types, HTTP response codes, and thus allowing existing layered proxy and gateway components to perform supplementary options on the web such as HTTP caching and security enforcement. What is important is that our approach differs from the idea presented in [7] in that it does not depend on XML syntax or the complex architecture of [6].

3. RESTful graph store

In this Section we discuss the idea of using the RESTful access to graph store data. We define three aspects of RESTful graph store: (1) the set of query operations supported by the graph store, (2) the set of managing operations supported by the graph store and (3) the MIME of the data supported by the graph store and HTTP response status codes. Additionally, we introduce the ability to map the proposed IRI to SPARQL queries. We also outline the equivalent operations to more advanced queries.

3.1. Graph queries

In this Subsection we present query operations dedicated to graphs in the graph store. Let V be the set of all variables, then RDF triple pattern TP is a pattern in $(S \cup V) \times (U \cup V) \times (O \cup V)$ and V is infinite and disjoint from I, B, L and D (see Section 1). A variable is prefixed by "-" and the "-" is not part of the variable name. Triple patterns are started after the graph and are separated by "/". Elements of the triple pattern are separated by "|".

A Graph Queries Operation is an action that accepts some arguments A and transforms a graph store GS to another graph store GS': $OpGraphQueries_{GS}(A) = GS'$. Arguments should be in the RDF triples form, triple patterns form or empty set. The result is either GS' in case of correct execution or GS in case of error. These operations (Table 1) allow clients to manipulate RDF triples:

- $OpSelect(tp)$ with $\{tp : tp \in TP\}$. This combines the operations of projecting from the graph store.
- $OpInsert(t)$ with $\{t : t \in T\}$. This adds triples into the graph store.
- $OpUpdate(tp_1, tp_2)$ with $\{tp_1 : tp_1 \in TP\}$ and $\{tp_2 : tp_2 \in TP\}$. This can update triples from the graph store.
- $OpDelete(t)$ with $\{t : t \in T\}$. This removes triples from the graph store.
- $OpAsk(tp)$ with $\{tp : tp \in TP\}$. This tests whether or not a query has a solution.
- $OpDescribe()$. This returns a result containing data about graph store resources.

Table 1
RESTful graph store HTTP methods for graph update and query

HTTP Methods	Operations	Mapping to SPARQL
GET	$OpSelect(tp)$	SELECT
POST	$OpInsert(t)$	INSERT DATA
PUT	$OpUpdate(tp_1, tp_2)$	DELETE/INSERT
DELETE	$OpDelete(t)$	DELETE DATA
HEAD	$OpAsk(tp)$	ASK
OPTIONS	$OpDescribe()$	DESCRIBE

These operations should be executes with IRI defined in Augmented Backus-Naur Form (ABNF) [8]:
IRI = scheme "://" ihost "/" graph ["/" query]
scheme = "http" / "https" ; supported protocols
graph = "default" / ∗(iunreserved / pct-encoded / sub-delims) ; default or named graph
query = ∗ (iunreserved / pct-encoded / sub-delims) ; triples and triple patterns

Rules *ihost*, *iunreserved*, *pct-encoded* and *sub-delims* are defined in [9].

The IRI example of selecting triples from graph store: *http://example.org/default/-x|foaf:name|-name*. All graphs and triple patterns should be encoded in [9].

3.2. Graph management

A Graph Management Operation is an action that accepts some arguments A and transforms a graph store GS to another graph store GS': $OpGraphmanagementGS(A) = GS'$. Arguments should be in the IRIs form or empty set. The operation performs the described transformation of the graph store either completely or leaves the graph store unchanged. These operations (Table 2) allow clients to manipulate graphs:

- $OpList()$. Selects all triples from graph in graph store.
- $OpCreate(u)$ with $u \in I$. Creates a graph in the graph store.
- $OpLoad(u_1, u_2)$ with $u_1 \in I$ and $u_2 \in I$. Reads an RDF document and inserts its triples into the graph in the graph store.
- $OpDrop(u)$ with $u \in I$. Removes the specified graph from the graph store.
- $OpInfo()$. Returns information about graph store. It may display SPARQL Service Description [10]

Table 2
RESTful graph store HTTP methods for graph management

HTTP Methods	Operations	Mapping to SPARQL
GET	$OpList()$	SELECT * WHERE ?s ?p ?o
POST	$OpCreate(u)$	CREATE
PUT	$OpLoad(u_1, u_2)$	LOAD
DELETE	$OpDrop(u)$	DROP
OPTIONS	$OpInfo()$	*none*

These operations should be executed with IRI defined in ABNF:
IRI = scheme “://” ihost [“/” references]
scheme = “http” / “https” ; supported protocols
references = reference [“/” reference]
*reference = * (iunreserved / pct-encoded / sub-delims) ; IRI reference*
Rules *iunreserved*, *pct-encoded* and *sub-delims* are defined in [9].
The IRI example of listing triples from graph store: *http://example.org/default*. All graphs should be encoded in [9].

3.3. Media types and status codes
In this Subsection we discuss the body of request and response HTTP messages [11]. Supported MIME types can be represented by syntaxes, such as: Turtle [12], RDF/XML [13], RDF/JSON [14], or any other valid type. A request depends on an *Accept* header and a response depends on a *Content-Type* header. The response codes are presented in Table 3.

Table 3
Relationship between HTTP status codes and methods

Status code	HTTP methods	Response Contains
200 (OK)	GET, POST, PUT, DELETE	Serialized data?
204 (No Content)	POST, PUT, DELETE	Empty
304 (Not Modified)	GET	Empty?
400 (Bad Request)	GET, POST, PUT, DELETE	Empty or serialized error message
404 (Not Found)	GET, PUT, DELETE	Empty
409 (Conflict)	POST, PUT, DELETE	Empty or Serialized error message

3.4. Mapping to SPARQL

In this Subsection we show how to map SPARQL 1.1 [15, 16] clauses that are not showed in Table 1 and Table 2 to proposed operations. These mappings are presented in Table 4.

Table 4

Equivalent operations

SPARQL clause	Equivalent operations
DELETE	$OpUpdate(\varnothing, tp_2)$
INSERT	$OpUpdate(tp_1, \varnothing)$
DELETE WHERE	$OpUpdate(\varnothing, \varnothing)$
CLEAR	$OpUpdate(\varnothing, tp_2)$
COPY	$OpDrop(u);\ OpUpdate(tp_1, \varnothing))$
MOVE	$OpDrop(u);\ OpUpdate(tp_1, \varnothing);\ OpDrop(u)$
ADD	$OpUpdate(tp_1, \varnothing)$

4. Conclusions

The problem of how to adjust query to a graph store has produced many proposals. Most of them are hard to use without dedicated tools, hence making the problem seem difficult. We assume that the graph stores, to be more functional, should provide a simple mechanism to execute the queries. The main motivation for this paper is the lack of such requirements.

We have produced a simple, thought-out and closed graph store proposal. We believe that our idea is an interesting approach, because it is graph store independent. Our proposal can work either with mobile and other devices or a web browser and other software as a graph store client and server.

Acknowledgments

The author would like to thank WebID Incubator Group from the World Wide Web Consortium. Professor Henryk Rybinski's comments and support were invaluable.

R E F E R E N C E S

[1] Klyne G., Carroll J. J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium (2004)

[2] Carroll J. J., Bizer C., Hayes P., Stickler P.: Named graphs, provenance and trust. 14th International Conference on World Wide Web. ACM (2005)

[3] Fielding R. T., Taylor R. N.: Principled Design of the Modern Web Architecture, Transactions on Internet Technology (TOIT), Volume 2 Issue 2 (2002)

[4] Cerami E.: Web Services Essentials. O'Reilly & Associates (2002)

[5] Aziz A., Kollhof J.: JSON-RPC 2.0 Specification, JSON-RPC Working Group (2010)

[6] Ryman A.: Simple object access protocol (SOAP) and Web services. 23rd International Conference on Software Engineering (2001)

[7] Clark K. G., Feigenbaum L., E. Torres E.: SPARQL Protocol for RDF. World Wide Web Consortium (2008)

[8] Crocker D., Overell P.: Augmented BNF for Syntax Specifications: ABNF. Internet Engineering Task Force (2008)

]9] M. Duerst M., Suignard M.: Internationalized Resource Identifiers (IRIs). Internet Engineering Task Force (2005)

[10] Williams G. T.: SPARQL 1.1 Service Description, World Wide Web Consortium (2009)

[11] Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T.: Hypertext Transfer Protocol – HTTP/1.1. Internet Engineering Task Force (1999)

[12] Eric Prud'hommeaux E., Gavin Carothers G.: Terse RDF Triple Language. World Wide Web Consortium (2011)

[13] Beckett D.: A retrospective on the development of the RDF/XML Revised Syntax. 2nd International Semantic Web Conference (2003)

[14] Tomaszuk D.: Named graphs in RDF/JSON serialization. Zeszyty Naukowe Politechniki Gdańskiej (2011)

[15] Harris S., Seaborne A.: SPARQL 1.1 Query Language. World Wide Web Consortium (2011)

[16] Gearon P., Passant A., Polleres P.: SPARQL 1.1 Update. World Wide Web Consortium (2011)

Dominik Tomaszuk
Institute of Computer Science
University of Bialystok, Poland
dtomaszuk@ii.uwb.edu.pl