Managing a graph store in the Linked Data and REST style

Dominik Tomaszuk[1]
[1]University of Bialystok, Institute of Computer Science
dtomaszuk@ii.uwb.edu.pl

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) graph stores. We present a fast method that suffice clients to use only Linked Data and Representational State Transfer style.

Keywords. Semantic Web, Linked Data, Representational State Transfer (REST), Graph store

1. Introduction and motivation

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 $\{G_0, (u_1, G_1), (u_2, G_2), ..., (u_n, G_n)\}$, where each G_i is a RDF graph and u_i is an IRI reference. Each u_i is distinct. G_0 is called 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.

The rules of Linked Data [3] can provide a very simple guide for selecting and modifying data on the graph store. In RESTful web services [4] requests and responses are built around the transfer of representations of resources. We attempt to define the proper methods to find and modify the RDF data in graph store with web service means. In this paper a new architectural style access to graph stores based on Linked Data and Representational State Transfer (REST) is presented. This proposal can be used without dedicated applications, it can be executed in web browsers.

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

2. 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 manage 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 ability to map the proposed IRI to SPARQL queries. We also outline the equivalent operations to more advanced queries.

2.1. 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 $(S \cup V) \times (I \cup V) \times (O \cup V)$ 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 should begin after graph name. Multiple triple patterns are separated by "/". Elements of triple pattern are separated by "|". All prefixes of abbreviated IRIs should be associated with well-known IRI [5]. If prefixes are not in well-known IRI, it is literal. Typed literal is suffixed by "+" and IRI to data type. Blank node is prefixed 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 $\{tp : tp \in TP\}$. It combines the operations of projecting from the graph store.
- *OpInsert(t)* with $\{t : t \in T\}$. It adds triples into the graph store.
- $OpUpdate(tp_1, tp_2)$ with $\{tp_1 : tp_1 \in TP\}$ and $\{tp_2 : tp_2 \in TP\}$. It can update triples from the graph store. Argument tp_1 represents data which is inserted and argument tp_2 represents data which is deleted. The updated triple patterns should be symmetrical in part of IRI[1].
- *OpDelete(t)* with $\{t : t \in T\}$. It removes triples from the graph store.
- *OpAsk(tp)* with $\{tp : tp \in TP\}$. It tests whether or not a query has a solution.
- *OpDescribe()*. It returns a result containing data about graph store resources.

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

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

This operations should be executed with IRI defined in Augmented Backus–Naur Form (ABNF) [6]:
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 [6].

For example *http://example.org/default/-x|foaf:name|-name* with GET method selecting triples from default graph that have FOAF [7] name[2] in predicate. All triple patterns should be encoded in [8].

2.2. Graph management

A Graph Managment Operation is an action that accepts some arguments *A* and transforms a graph store *GS* to another graph store *GS'*: $OpGraphmanagment_{GS}(B) = GS'$. 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()*. It selects all triples from graph into graph store.

1 E.g. *http://example.org/g/a|b|c/x|y|z*, where triple pattern *a|b|c* is inserted and *x|y|z* is deleted.
2 Assuming that the foaf:name is associated /.well-known/ with FOAF vocabluary.

- *OpCreate(u)* with $u \in I$. It creates a graph in the graph store.
- *OpLoad(u_1, u_2)* with $u_1 \in I$ and $u_2 \in I$. It reads an RDF document and inserts its triples into the graph in the graph store.
- *OpDrop(u)* with $u \in I$. It removes the specified graph from the graph store.
- *OpInfo()*. It returns information about graph store. It may display SPARQL Service Description [9]

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*

Table 2: RESTful graph store HTTP methods for graph management

This 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 [6].

For example *http://example.org/default* with GET method listing triples in default graph from graph store.

2.3. Media types and status codes

In this Subsection we discuss the body of request and response HTTP messages [10]. Supported MIME types can be represented by syntaxes, such as: Turtle [11], RDF/XML [12], RDF/JSON [13], or any other valid type. A request depends on *Accept* header and a response depends on *Content-Type* header. The response codes are presented in Table 3.

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

Table 3: Relationship between HTTP status codes and methods

2.4. Mapping to SPARQL

In this Subsection we show how to map SPARQL 1.1 [14, 15] clauses that are not shown in Table 1 and Table 2 to proposed operations. This mappings are presented in Table 4.

SPARQL clause	Equivalent operations
DELETE	*OpUpdate({} tp_2)*
INSERT	*OpUpdate(tp_1, {})*
DELETE WHERE	*OpUpdate({}, {})*
CLEAR	*OpUpdate({} tp_2)*
COPY	*OpDrop(u)*; *OpUpdate(tp_1, {})*)
MOVE	*OpDrop(u); OpUpdate(tp_1, {})*; *OpDrop(u)*
ADD	*OpUpdate(tp_1, {})*

Table 4: Equivalent operations

3. Related Work

The 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 [16]. It uses Extensible Markup Language (XML) in request body and response returned values. An XML-RPC message use POST Hypertext Transfer Protocol (HTTP) method. Another RPC approach is JSON-RPC [17]. 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) [18] 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 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 new proposal for query processing and returning the query results service [19]. 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 are using SPARQL query language. Furthermore, this SOAP-based protocol is complex and it can be significantly slower because it uses 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 allows existing layered proxy and gateway components to perform supplementary options on the web such as HTTP caching and security enforcement. What is important is the fact that, our approach differs from the idea presented in [19] in that it does not depend on XML syntax or complex architecture of [18]. Yet another approach is provided in [20, 21], which respect REST style. Unfortunately, [20, 21] are dependent on the SPARQL. Furthermore, [20] does not include all operations on graph store. In [21] there is also arbitrariness in the HTTP methods used in graph store operations.

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 seems difficult. We assume that the graph stores, to be more functional, should provide a simple mechanism to execute the queries.

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 as a graph store high scalability enterprise server.

Acknowledgments

Professor Henryk Rybinski’s comments and support were invaluable.

References

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. Bizer C., Heath T., Berners-Lee T.: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems (2009)
4. Fielding R. T., Taylor R. N.: Principled Design of the Modern Web Architecture, Transactions on Internet Technology (TOIT), Volume 2 Issue 2 (2002)
5. Nottingham M., Hammer-Lahav E.: Defining Well-Known Uniform Resource Identifiers (URIs). Internet Engineering Task Force (2010)
6. Crocker D., Overell P.: Augmented BNF for Syntax Specifications: ABNF. Internet Engineering Task Force (2008)
7. Brickley D., Miller L.: FOAF Vocabulary Specification 0.98. FOAF Project (2010)
8. M. Duerst M., Suignard M.: Internationalized Resource Identifiers (IRIs). Internet Engineering Task Force (2005)
9. Williams G. T.: SPARQL 1.1 Service Description, World Wide Web Consortium (2009)
10. 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)
11. Prud'hommeaux E., Carothers G.: Terse RDF Triple Language. World Wide Web Consortium (2011)
12. Beckett D.: A retrospective on the development of the RDF/XML Revised Syntax. 2nd International Semantic Web Conference (2003)
13. Tomaszuk D.: Named graphs in RDF/JSON serialization. Zeszyty Naukowe Politechniki Gdańskiej (2011)
14. Harris S., Seaborne A.: SPARQL 1.1 Query Language. World Wide Web Consortium (2011)
15. Gearon P., Passant A., Polleres P.: SPARQL 1.1 Update. World Wide Web Consortium (2011)
16. Cerami E.: Web Services Essentials. O'Reilly & Associates (2002)
17. Aziz A., Kollhof J.: JSON-RPC 2.0 Specification, JSON-RPC Working Group (2010)
18. Ryman A.: Simple object access protocol (SOAP) and Web services. 23rd International Conference on Software Engineering (2001)
19. Clark K. G., Feigenbaum L., Torres E.: SPARQL Protocol for RDF. World Wide Web Consortium (2008)
20. Wilde E., Hausenblas M.: RESTful SPARQL? You name it!: aligning SPARQL with REST and resource orientation. WEWST '09 Proceedings of the 4th Workshop on Emerging Web Services Technology (2009)
21. Ogbuji C.: SPARQL 1.1 Graph Store HTTP Protocol. World Wide Web Consortium (2008)