SPARQL/Update: A language for updating RDF graphs

Andy Seaborne, Geetha Manjunath
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2007-102
July 3, 2007*

RDF, semantic web, SPARQL

This document describes SPARQL/Update (nicknamed "SPARUL"), an update language for RDF graphs. It uses a syntax derived form SPARQL. Update operations are performed on a collection of graphs in a Graph Store. Operations are provided to change existing RDF graphs as well as create and remove graphs with the Graph Store.

This document does not discuss the SPARQL/Update protocol.

* Internal Accession Date Only

Approved for External Publication

SPARQL/Update
A language for updating RDF graphs

Authors:
Andy Seaborne, Hewlett-Packard
Geetha Manjunath, Hewlett-Packard
Version:
Version 2 : 2007-03-28

Abstract

This document describes SPARQL/Update (nicknamed "SPARUL"), an update language for RDF graphs. It uses a syntax derived form SPARQL. Update operations are performed on a collection of graphs in a Graph Store. Operations are provided to change existing RDF graphs as well as create and remove graphs with the Graph Store.

This document does not discuss protocol issues.

Status of This Document

Published for discussion.

Table of Contents

1 Introduction

SPARQL/Update is a language to express updates to an RDF store. It is intended to be a standard mechanism by which updates to a remote RDF Store can be described, communicated and stored. SPARQL/Update is a companion language to SPARQL and is envisaged to be used in conjunction with SPARQL query language and protocol. The reuse of the SPARQL syntax, in both style and detail, reduces the learning curve for developers and reduces implementation costs.

SPARQL/Update provides the following facilities:

- Insert new triples to an RDF graph.

- Delete triples from an RDF graph.
- Perform a group of update operations as a single action.
- Add a new RDF Graph to a Graph Store.
- Remove an RDF graph from a Graph Store.

This document also describes the use HTTP for the transfer of update operations.

See also:

- the wiki page SparqlUpdateLanguage for a related proposal
- Delta: an ontology for the distribution of differences between RDF graphs

1.1 Scope and Limitations

SPARQL/Update is not:

- a replacement for RSS or Atom, which advertise changes.
- a mechanism to exchange changes to RDF graphs between applications.
- a general web-wide approach to update and distribution of RDF-based information.

1.2 Document Conventions

Language forms are show as:

```
INSERT { template } [ WHERE { pattern } ]
```

`[]` indicates an optional part of the syntax. `[]` are still used for blank nodes in SPARQL requests.

`Italics` indicate an item is some syntax element derived from SPARQL.

Examples are shown as:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT { <http://example/egbook3> dc:title  "This is an example title" }
```

2 Examples

This section gives some example snippets in the proposed SPARQL/Update language that would be used to update a remote RDF store.

(a) Adding some triples to a graph. The snippet describes two RDF triples to be inserted into the default graph of the RDF store.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT
{ <http://example/book3> dc:title    "A new book" ;
                         dc:creator  "A.N.Other" .
}
```

(b) This SPARQL/Update request contains a triple to be deleted and a triple to be added (used here to correct a book title). The requested modification is effected in the named graph identified by the URI http://example/bookStore.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

MODIFY GRAPH <http://example/bookStore>
DELETE
 { <http://example/book3>  dc:title  "Fundamentals of Compiler Desing" }
INSERT
 { <http://example/book3>  dc:title  "Fundamentals of Compiler Design" }
```

(c) The example below has a request to delete all records of old books (with date before year 2000)

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

DELETE
 { ?book ?p ?v }
WHERE
  { ?book dc:date ?date .
    FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
  }
```

(d) This snippet copies records from one named graph to another named graph based on a pattern

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

INSERT INTO GRAPH <http://example/bookStore2>
 { ?book ?p ?v }
WHERE
  { GRAPH  <http://example/bookStore>
       { ?book dc:date ?date .
         FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
         ?book ?p ?v
  } }
```

(e) An example to move records from one named graph to another named graph based on a pattern

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

INSERT INTO GRAPH <http://example/bookStore2>
 { ?book ?p ?v }
WHERE
  { GRAPH  <http://example/bookStore>
     { ?book dc:date ?date .
       FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
     }
  }

DELETE FROM GRAPH <http://example/bookStore>
 { ?book ?p ?v }
WHERE
  { GRAPH  <http://example/bookStore>
      { ?book dc:date ?date .
        FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
```

```
    }
}
```

3 The Graph Store

A Graph Store is a repository of RDF graphs managed by a single service. Like an RDF Dataset operated on by SPARQL, a Graph Store has zero or one unnamed graphs and zero or more named graphs. Additionally, graphs can be added or deleted.

In the simple case, where there is one default graph and no named graphs, SPARQL/Update is a language for update of a single graph.

3.1 Graph Store and SPARQL Query Services

If an update service is managing some graph store, then there is no presumption that this exactly corresponds to any RDF dataset offered by some query service. The dataset of the query service may be formed from graphs managed by the update service but the dataset offered by the query can be a subset of the graphs in the update service dataset and the names of those graphs may be different. The composition of the RDF dataset may also change.

3.2 SPARQL/Update Services

SPARQL/Update requests are a sequence of operations. Each request should be treated atomically by a SPARQL/Update service.

If two different update services are managing their respective graph stores share a common graph, there is no presumption that the updates done through one service would be propagated to the other. The behaviour of these services with respect to each other is implementation dependent. It is recommended that such deployment scenarios are completely avoided. An implementation must target SPARQL queries on updated graphs if the SPARQL and SPARUL end points are the same.

4 Update Language

SPARQL/Update supports two categories of update operations on a Graph Store:

- Graph Update - addition and removal of triples from one of the graphs in the graph store.
- Graph Management - creating and deletion of graphs within the graph store.

4.1 Graph Update

Graph update operations change existing graphs in the Graph Store but do not create or delete them.

The fundamental operation of a graph update is MODIFY. There are two restricted forms for INSERT and DELETE of triples; both are special cases of the MODIFY operation. A modify operation consists of a group of triples to be deleted and a group of triples to be added. The specification of the triples can be based on a query pattern.

The `LOAD` operation reads the contents of one graph into another.

The `CLEAR` operation removes all the triples in a graph.

4.1.1 MODIFY

```
# UPDATE outline syntax : general form:
MODIFY [ GRAPH <uri> ]
DELETE { template }
INSERT { template }
[ WHERE { pattern } ]
```

The *template* and *pattern* forms are as defined in SPARQL for construct templates and graph patterns.

The deletion of the triples happens before the insertion. The pattern in WHERE clause is evaluated only once, before the delete part of the operation is performed. The overall processing model is that the pattern is executed, the results used to instantiate the DELETE template, the deletes performed, the results used again to instantiate the INSERT template, and the inserts performed.

The *pattern* is evaluated as in a SPARQL query SELECT * WHERE { *pattern* } and all the named variables are made available to the INSERT template and the DELETE template for defining the triples to be inserted or deleted.

The GRAPH <uri> clause names the graph in the graph store to be updated; if omitted, the MODIFY applies to the unnamed graph.

If the operation is on a graph that does not exist, an error is generated. Graphs are not created by the graph update operations. The graph management operations have to be used to create new graphs.

4.1.2 DELETE

The DELETE operation is similar to the MODIFY operation with an empty INSERT template. The graph URI, if present, must be a valid named graph in the Graph Store.

```
DELETE [ FROM GRAPH <uri> ]* { template } [ WHERE { pattern } ]
```

```
# Equivalent to
MODIFY [ GRAPH <uri> ]*
DELETE { template }
INSERT { }
[ WHERE { pattern } ]
```

4.1.3 INSERT

The INSERT operation is similar to the MODIFY operation with an empty DELETE template. The graph URI, if present, must be a valid named graph in the Graph Store. The CREATE graph management operator should be used to create a new named graph in the Graph Store.

```
INSERT [ INTO GRAPH <uri> ]* { template } [ WHERE { pattern } ]
```

```
#Equivalent to
MODIFY [ GRAPH <uri> ]*
DELETE { }
INSERT { template }
[ WHERE { pattern } ]
```

4.1.4 LOAD

The LOAD operation copies all the triples of a remote graph into the specified graph, or if not specified, the default graph of the Graph Store.

```
LOAD <remoteURI> [ INTO GRAPH <uri> ]
```

4.1.5 CLEAR

The CLEAR operations removes all the triples of the specified graph or if not specified, the default graph of the Graph Store. This operation does not remove the graph from the Graph Store.

```
# Remove all triples.
CLEAR [ GRAPH <uri> ]
```

It has the same effect as:

```
# Remove all triples.
DELETE [ GRAPH <uri> ] { ?s ?p ?o } WHERE { ?s ?p ?o }
```

but is a clearer expression of empting a graph.

4.2 Graph Management

Graph management operations create and destroy named graphs in the Graph Store.

The default graph in a Graph Store always exists.

4.2.1 Graph Creation

```
CREATE [ SILENT ] GRAPH <uri>
```

This operation creates a new named graph with a name as specified by the URI. After the successful completion of this operation, the new named graph is available for any further graph update operations with MODIFY, INSERT and DELETE operators.

The SPARQL/Update service generates an error if the graph referred by the URI already exists unless the keyword SILENT is present when no error is generated and execution of the sequence of SPARQL/Update operations continues.

4.2.2 Graph Removal

```
DROP [ SILENT ] GRAPH <uri>
```

This operation removes the specified named graph from the Graph Store associated with the SPARQL/Update service endpoint. After successful completion of this operation, the named graph is no more available for further graph update operations.

The SPARQL/Update service, by default, is expected to generate an error if the specified named graph does not exist. If SILENT is present, this error is ignored and execution of a sequence of SPARQL/Update operations continues.

5 SPARQL/Update Grammar

Rules from rule *Prologue* are taken from the SPARQL grammar.

[1]	*SPARUL*	::=	Prologue ((Update | Manage))*
[2]	*Update*	::=	((Modify | Insert | Delete) WhereClause?) | ((Load | Clear))
[3]	*Modify*	::=	'MODIFY' GraphIRI* 'DELETE' ConstructTemplate 'INSERT' ConstructTemplate
[4]	*Delete*	::=	'DELETE' ('FROM'? IRIref)* ConstructTemplate
[5]	*Insert*	::=	'INSERT' ('INTO'? IRIref)* ConstructTemplate
[6]	*GraphIRI*	::=	'GRAPH' IRIref
[7]	*Load*	::=	'LOAD' IRIref+ ('INTO' IRIref)?
[8]	*Clear*	::=	'CLEAR' GraphIRI?
[9]	*Manage*	::=	Create | Drop
[10]	*Create*	::=	'CREATE' 'SILENT'? GraphIRI
[11]	*Drop*	::=	'DROP' 'SILENT'? GraphIRI
[12]	*Prologue*	::=	BaseDecl? PrefixDecl*
[13]	*BaseDecl*	::=	'BASE' Q_IRI_REF
[14]	*PrefixDecl*	::=	'PREFIX' QNAME_NS Q_IRI_REF
[15]	*WhereClause*	::=	'WHERE'? GroupGraphPattern
[16]	*GroupGraphPattern*	::=	'{' TriplesBlock? ((GraphPatternNotTriples | Filter) '.'? TriplesBlock?)* '}'
[17]	*TriplesBlock*	::=	TriplesSameSubject ('.' TriplesBlock?)?
[18]	*GraphPatternNotTriples*	::=	OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern
[19]	*OptionalGraphPattern*	::=	'OPTIONAL' GroupGraphPattern
[20]	*GraphGraphPattern*	::=	'GRAPH' VarOrIRIref GroupGraphPattern
[21]	*GroupOrUnionGraphPattern*	::=	GroupGraphPattern ('UNION' GroupGraphPattern)*

[22]	*Filter*	::=	'FILTER' Constraint
[23]	*Constraint*	::=	BrackettedExpression | BuiltInCall | FunctionCall
[24]	*FunctionCall*	::=	IRIref ArgList
[25]	*ArgList*	::=	(NIL | '(' Expression (',' Expression)* ')')
[26]	*ConstructTemplate*	::=	'{' ConstructTriples? '}'
[27]	*ConstructTriples*	::=	TriplesSameSubject ('.' ConstructTriples?)?
[28]	*TriplesSameSubject*	::=	VarOrTerm PropertyListNotEmpty | TriplesNode PropertyList
[29]	*PropertyListNotEmpty*	::=	Verb ObjectList (';' (Verb ObjectList)?)*
[30]	*PropertyList*	::=	PropertyListNotEmpty?
[31]	*ObjectList*	::=	Object (',' Object)*
[32]	*Object*	::=	GraphNode
[33]	*Verb*	::=	VarOrIRIref | 'a'
[34]	*TriplesNode*	::=	Collection | BlankNodePropertyList
[35]	*BlankNodePropertyList*	::=	'[' PropertyListNotEmpty ']'
[36]	*Collection*	::=	'(' GraphNode+ ')'
[37]	*GraphNode*	::=	VarOrTerm | TriplesNode
[38]	*VarOrTerm*	::=	Var | GraphTerm
[39]	*VarOrIRIref*	::=	Var | IRIref
[40]	*Var*	::=	VAR1 | VAR2
[41]	*GraphTerm*	::=	IRIref | RDFLiteral | NumericLiteral | BooleanLiteral | BlankNode | NIL
[42]	*Expression*	::=	ConditionalOrExpression
[43]	*ConditionalOrExpression*	::=	ConditionalAndExpression ('||' ConditionalAndExpression)*
[44]	*ConditionalAndExpression*	::=	ValueLogical ('&&' ValueLogical)*
[45]	*ValueLogical*	::=	RelationalExpression

[46]	*RelationalExpression*	::=	NumericExpression ('=' NumericExpression | '!=' NumericExpression | '<' NumericExpression | '>' NumericExpression | '<=' NumericExpression | '>=' NumericExpression)?
[47]	*NumericExpression*	::=	AdditiveExpression
[48]	*AdditiveExpression*	::=	MultiplicativeExpression ('+' MultiplicativeExpression | '-' MultiplicativeExpression | NumericLiteralPositive | NumericLiteralNegative)*
[49]	*MultiplicativeExpression*	::=	UnaryExpression ('*' UnaryExpression | '/' UnaryExpression)*
[50]	*UnaryExpression*	::=	'!' PrimaryExpression | '+' PrimaryExpression | '-' PrimaryExpression | PrimaryExpression
[51]	*PrimaryExpression*	::=	BrackettedExpression | BuiltInCall | IRIrefOrFunction | RDFLiteral | NumericLiteral | BooleanLiteral | Var
[52]	*BrackettedExpression*	::=	'(' Expression ')'
[53]	*BuiltInCall*	::=	'STR' '(' Expression ')' | 'LANG' '(' Expression ')' | 'LANGMATCHES' '(' Expression ',' Expression ')' | 'DATATYPE' '(' Expression ')' | 'BOUND' '(' Var ')' | 'sameTerm' '(' Expression ',' Expression ')' | 'isIRI' '(' Expression ')' | 'isURI' '(' Expression ')' | 'isBLANK' '(' Expression ')' | 'isLITERAL' '(' Expression ')' | RegexExpression
[54]	*RegexExpression*	::=	'REGEX' '(' Expression ',' Expression (',' Expression)? ')'
[55]	*IRIrefOrFunction*	::=	IRIref ArgList?
[56]	*RDFLiteral*	::=	String (LANGTAG | ('^^' IRIref))?
[57]	*NumericLiteral*	::=	NumericLiteralUnsigned | NumericLiteralPositive | NumericLiteralNegative
[58]	*NumericLiteralUnsigned*	::=	INTEGER | DECIMAL | DOUBLE

```
[59]  NumericLiteralPositive  ::= INTEGER_POSITIVE | DECIMAL_POSITIVE | DOUBLE_POSITIVE
[60]  NumericLiteralNegative  ::= INTEGER_NEGATIVE | DECIMAL_NEGATIVE | DOUBLE_NEGATIVE
[61]  BooleanLiteral          ::= 'true' | 'false'
[62]  String                  ::= STRING_LITERAL1 | STRING_LITERAL2 | STRING_LITERAL_LONG1 | STRING_LITERAL_LONG2
[63]  IRIref                  ::= Q_IRI_REF | QName
[64]  QName                   ::= QNAME_LN | QNAME_NS
[65]  BlankNode               ::= BLANK_NODE_LABEL | ANON
[66]  Q_IRI_REF               ::= '<' ([^<>'{}|^`]-[#x00-#x20])* '>'
[67]  QNAME_NS                ::= NCNAME_PREFIX? ':'
[68]  QNAME_LN                ::= QNAME_NS NCNAME
[69]  BLANK_NODE_LABEL        ::= '_:' NCNAME
[70]  VAR1                    ::= '?' VARNAME
[71]  VAR2                    ::= '$' VARNAME
[72]  LANGTAG                 ::= '@' [a-zA-Z]+ ('-' [a-zA-Z0-9]+)*
[73]  INTEGER                 ::= [0-9]+
[74]  DECIMAL                 ::= [0-9]+ '.' [0-9]* | '.' [0-9]+
[75]  DOUBLE                  ::= [0-9]+ '.' [0-9]* EXPONENT | '.' ([0-9])+ EXPONENT | ([0-9])+ EXPONENT
[76]  INTEGER_POSITIVE        ::= '+' INTEGER
[77]  DECIMAL_POSITIVE        ::= '+' DECIMAL
[78]  DOUBLE_POSITIVE         ::= '+' DOUBLE
[79]  INTEGER_NEGATIVE        ::= '-' INTEGER
[80]  DECIMAL_NEGATIVE        ::= '-' DECIMAL
[81]  DOUBLE_NEGATIVE         ::= '-' DOUBLE
[82]  EXPONENT                ::= [eE] [+-]? [0-9]+
```

```
[83]   STRING_LITERAL1          ::= "'" ( ([^#x27#x5C#xA#xD]) | ECHAR )* "'"
[84]   STRING_LITERAL2          ::= '"' ( ([^#x22#x5C#xA#xD]) | ECHAR )* '"'
[85]   STRING_LITERAL_LONG1     ::= "'''" ( ( "'" | "''" )? ( [^'\] | ECHAR ) )
                                    * "'''"
[86]   STRING_LITERAL_LONG2     ::= '"""' ( ( '"' | '""' )? ( [^"\] | ECHAR ) )
                                    * '"""'
[87]   ECHAR                    ::= '\' [tbnrf\"']
[88]   NIL                      ::= '(' WS* ')'
[89]   WS                       ::= #x20 | #x9 | #xD | #xA
[90]   ANON                     ::= '[' WS* ']'
[91]   NCCHAR1P                 ::= [A-Z] | [a-z] | [#x00C0-#x00D6] | [#x00D8-
                                    #x00F6] | [#x00F8-#x02FF] | [#x0370-#x037D]
                                    | [#x037F-#x1FFF] | [#x200C-#x200D] |
                                    [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-
                                    #xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD]
                                    | [#x10000-#xEFFFF]
[92]   NCCHAR1                  ::= NCCHAR1P | '_'
[93]   VARNAME                  ::= ( NCCHAR1 | [0-9] ) ( NCCHAR1 | [0-9] |
                                    #x00B7 | [#x0300-#x036F] | [#x203F-#x2040] )
                                    *
[94]   NCCHAR                   ::= NCCHAR1 | '-' | [0-9] | #x00B7 | [#x0300-
                                    #x036F] | [#x203F-#x2040]
[95]   NCNAME_PREFIX            ::= NCCHAR1P ((NCCHAR|'.')* NCCHAR)?
[96]   NCNAME                   ::= NCCHAR1 ((NCCHAR|'.')* NCCHAR)?
```

References

[SPARQL-Q]
SPARQL Query Language for RDF, Eric Prud'hommeaux, Andy Seaborne (editors),W3C Working Draft (work in progress).

[SPARQL-R]
SPARQL Query Results XML Format , D. Beckett, Editor, W3C Working Draft (work in progress), 27 May 2005, http://www.w3.org/TR/2005/WD-rdf-sparql-XMLres-20050527/ . Latest version available at http://www.w3.org/TR/rdf-sparql-XMLres/ .

[SPARQL-P]
SPARQL Protocol for RDF, K. Clark, Editor, W3C Working Draft (work in progress).

[RFC3986]
RFC 3986 *Uniform Resource Identifier (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter January 2005

[RFC3987]
RFC 3987, "Internationalized Resource Identifiers (IRIs)", M. Dürst , M. Suignard

[UNICODE]

The Unicode Standard, Version 4. ISBN 0-321-18578-1, as updated from time to time by the publication of new versions. The latest version of Unicode and additional information on versions of the standard and of the Unicode Character Database is available at http://www.unicode.org/unicode/standard/versions/.