5-24-06 Notes
Glenn Takanishi

Context Path Traversal in RDF Graphs

1 Introduction

One of the main problems of finding patterns in textual information is isolating the context[1] in which each sentence is expressed. Isolating context is a constantly occurring problem when reading, querying and making inferences in textual data. A semantic network[2] is constructed so that each word in the network gets its meaning from its association with other words in its neigborhood proximity.

In the semantic network model, the relative positions of RDF triples[3] with respect to each other in the network defines what each triple is. The context in which a triple is defined depends on the other triples in its local neigborhood. Context can further be determined by interpreting reference markers for each triple when traversing paths the graph network.

This notion that the contexts will reveal itself as one dynamically traverses the network structures is important for making inferences. This essential feature of a query language is not explicitly implemented in SPARQL using explicit schemas, or recursive or nested forward and backtracking search algorithms. Instead, since SPARQL performs pattern matching on graphs or subgraphs, we essentially have to do "path navigation" by refining the pattern matching process submitting better input graphs by "trial-and-error" so to speak. This strategy keeps the algorthmic processes simple.

There are pros and cons to this: If you know what you want to query for, okay. However, if the you're seeking to discover patterns you cannot be expected to foresee, then maybe using a heuristically guided search strategy is better. These issues are related to using the semantic network software which helps in generating "context paths." These software will have the effect of limiting the scope of the selection paths.

2 Context Nodes

A triple (s, p, o) is a directed link between two nodes in a network. A set of triples makes a graph. In the mechanics of navigating or traversing graphs, it's important to understand how much information which determines the traversal paths are missing . The meaning of each isolated triple maybe ambigous because it can have multiple meanings. So traversible graph paths depends not only on the relative positions of the triples in the graph network, but on something more called context[4] .

In the RDF data model's indexing architecture, context is defined within a pair (t, c) where t = (s, p, o) triple, and c is a node (URI reference or literal) in RDF space. The context in RDF space may also be defined as a named subgraphs which is itself

composed of a set of triples. The context is a coordinate in the graph.

2.1 Example Query

SPARQL can query multiple named graphs in one query. The following query finds people described in two named FOAF graphs.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
FROM NAMED <phil-foaf.rdf>
FROM NAMED <jim-foaf.rdf>
WHERE {
  GRAPH <phil-foaf.rdf> {
    ?x rdf:type foaf:Person .
    ?x foaf:name ?name .
  } .
  GRAPH <jim-foaf.rdf> {
    ?y rdf:type foaf:Person .
    ?y foaf:name ?name .
  } .
}
```

I could not get the exact form of the query above to work using Redland's Rasqal query tool. However, a simpler form [figure 1] worked. The RDF graphs phil-foaf.rdf and jim-foaf.rdf were downloaded from different web sites, but both contained the name and mbox-sha1sum field types.

A query can use named graphs as pattern matching templates against the search data. The following query[1] from Philip McCarthy's article show a named query used as a template on RSS data.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rss: <http://purl.org/rss/1.0/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?known_name ?link
FROM <http://planetrdf.com/index.rdf>
FROM NAMED <phil-foaf.rdf>
WHERE {
  GRAPH <phil-foaf.rdf> {
    ?me foaf:name "Phil McCarthy" .
    ?me foaf:knows ?known_person .
    ?known_person foaf:name ?known_name .
  } .
  ?item dc:creator ?known_name .
  ?item rss:title ?title .
  ?item rss:link ?link .
  ?item dc:date ?date.
}

ORDER BY DESC[?date] LIMIT 10
```

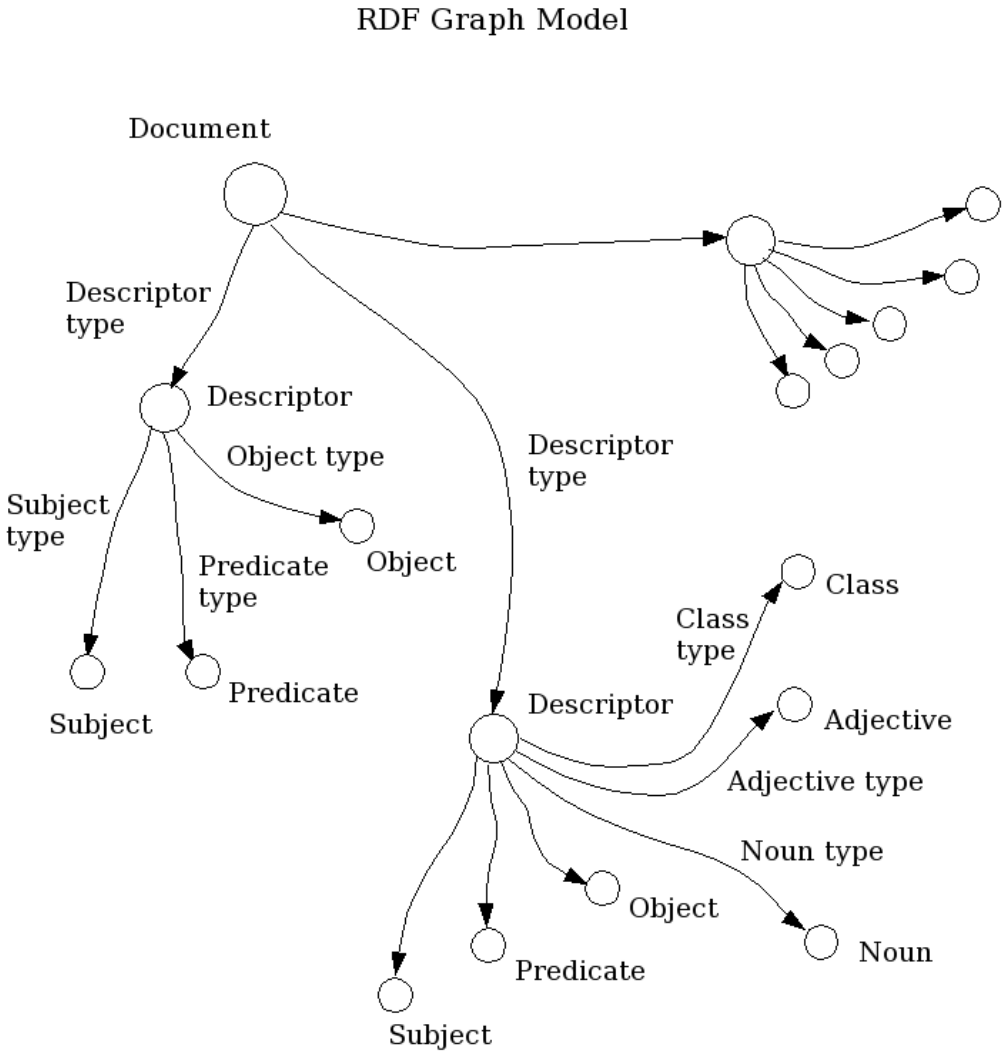

Figure 2. The RDF Graph Model

3. Path Expression

One thing to note in this XML record is the rdf type **xml:base** which is the name of the text document on our local computer.

The graph database only accepts RDF triples. The XML-RDF record above needs to be serialized, that is, separated into individual triples, before the data record can be handed to the RDF store. I used Dave Beckett's Raptor program, specifically rapper, to serialize the file. The output below shows the triples in N3 format. You can see references to the document URI file name in the first column below.

```
rapper: Parsing file test2.rdf
<http://localhost/textdata/simple.txt#x0001> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://localhost/ontology/descriptor#Type-Triple> .
<http://localhost/textdata/simple.txt#x0001>
<http://localhost/ontology/descriptor#subject> "this" .
<http://localhost/textdata/simple.txt#x0001>
<http://localhost/ontology/descriptor#predicate> "is" .
<http://localhost/textdata/simple.txt#x0001>
<http://localhost/ontology/descriptor#object> "test" .
<http://localhost/textdata/simple.txt#x0002> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://localhost/ontology/descriptor#Type-Chunk> .
<http://localhost/textdata/simple.txt#x0002>
<http://localhost/ontology/descriptor#noun> "emergency broadcast system" .
rapper: Parsing returned 6 statements
```

We can feed this output directly into the Redland RDF store, and run SPARQL queries to get answers.

Comments

The beauty of using the graph model is that almost any kind of unstructured data can be represented relatively easily. For example, you could annotate or tag picture images or music sound files with RDF atoms. In principle, one could easily mix imagery metadata information with our text metadata. It would all get linked into graphs.

References

1. RDF Semantics: W3C Recommendation 10 February 2004;
http://www.w3.org/TR/2004/REC-rdf-mt- 20040210/

2. M. Ross Quinlan, Semantic Memory; 1968, Semantic Information Processing, M. Minsky (ed), 216-260, MIT Press

3. Web and Semantic Web Query Languages: A Survey:
http://www.pms.ifi.lmu.de/publikationen/PMS-FB/PMS-FB-2005-14-slides.html

Appendix

SPARQL – RDF Query Language

After a couple of years in which different query languages for RDF data have been developed in parallel, the W3C has launched a working group on RDF Data Access with the purpose of defining a standard query language and a protocol for accessing RDF data. Meanwhile first versions of working drafts on both aspects are available from the W3C homepage. In the following, we concentrate on the proposed query language called SPARQL and briefly discuss the design choices made by the working group with respect to the above mentioned aspects of RDF query languages.

A first observation that we have to make about SPARQL is that it explicitly excludes a number of features that can be found in other RDF query languages. The most obvious restriction of the language is that it currently does not address for reasoning. The semantics of RDF schema as well as the use of rules are considered to be out of the scope of the current work on SPARQL. This decision is unfortunate as the ability to define and reason about schema information is an important aspect of semantic web data. On the other hand, it is understandable, that this initial effort for defining a standard focuses on the basic graph matching issue and technicalities around the use of the language. In the following, we will briefly introduce the way SPARQL allows the user to specify graph patterns to be matched against an RDF model and mention some additional features that have been included to address real world problems.
The general structure of a SPARQL Query is similar to the one of many previous SQL-like RDF query languages. In particular, a SPARQL query consists of the following main elements:

PREFIX: In this first part of a query abbreviations for name spaces can be defined to improve readability in the following parts.
SELECT: In the second part of the query, the return variables and structures are

specified.
WHERE: The final part specifies constraints on the return variables in terms of a graph pattern and constraints on values in the matched graph

In the following, we will have a closer look at the WHERE part of SPARQL queries and the different ways in which constraints can be formulated. Further, we will briefly discuss two other constructs that are used to return results different from a simple variable binding.

Matching Graph Structures

The SPARQL approach to describing graph patterns clearly follows the triple-centered approach. Single RDF statements are represented by triple patterns that consist of a sequence of three terms that either refer to identifiers of resources or are marked as variables by an initial question mark. Triple patterns are delimited by a dot. Statements can be grouped using brackets. The query below shows a simple example of a query for names of people and their mailboxes.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
        { ?x foaf:name ?name .
          ?x foaf:mbox ?mbox }
```

In order to reduce redundancy in the graph patterns, the language introduces a shorthand notation for triples with the same subject and for triples with the same subject and the same predicate. The following are examples of this shorthand notation where the original statement is on the left and the shorthand notation on the right-hand side.

```
?x  foaf:name  ?name .                  ?x  foaf:name  ?name ;
?x  foaf:mbox  ?mbox .                      foaf:mbox  ?mbox


?x  foaf:nick  "Alice" .                ?x foaf:nick  "Alice" , "Alice_" ..
?x  foaf:nick  "Alice_" .
```

In SPARQL, blank nodes are in general represented by an empty set of brackets. In order to reduce the size of the query expression, in cases where the blank node is the subject of a triple, the brackets can also be drawn around the predicate and the object to indicate, that the subject is not specified. This shorthand notation can also be combined with other abbreviations, e.g. in cases where several statements refer to the same blank node. The following example illustrates this case. Again the original expression is shown on the left hand and the abbreviation on the right hand side.

```
[] foaf:name  ?name .                   [ foaf:name  ?name ;
[] foaf:mbox  <alice@example.org>         foaf:mbox  <alice@example.org> ]
```

Special attention is paid to the flexible nature of RDF data. In particular, the language allows us to specify optional parts of a graph structure. Parts of the query expression can be marked with the prefix OPTIONAL indicating that the following group of statements in optional. This option block normally contains return variables. These are returned if the optional part can be found in the source data. If it is not found, the

graph is still assumed to match the data, but no values are returned for the respective variables. The following query for example returns the names of people and their mailbox if they have one.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name .
                OPTIONAL { ?x  foaf:mbox  ?mbox }}
```

In addition to optional parts in query expressions, the language also allows to define alternative representations of the same data in terms of UNION queries.

Matching Literals

As discussed above, the second aspect of matching RDF data is the matching of labels in the RDF graph. In the last section, we already discussed the use of resource identifiers in triple patterns. SPARQL also allows the use of literals and datatype values in triple patterns. The following query retrieves the subjects of all triples in which the object is the integer value 42.

```
SELECT ?v WHERE { ?v ?p 42 }
```

This way of representing datatypes just by a value is actually a shortcut for a more elaborated representation that consists of the value and information about the corresponding datatype. The complete version of the above query is the following.

```
SELECT ?x WHERE { ?x ?p "42"^^xsd:integer }
```

Besides this way of using datatype values that enforce an exact match, SPARQL also contains elements for defining constraints of a particular datatype value in the query expression. These constraints are identified by the keyword “FILTER” and can consist of complex arithmetic expressions over the corresponding datatype. In the current version the language defines operators for Boolean expressions, numerical values, strings, dates and variables. Further, the language specification contains casting operators for certain datatypes. The query below is a simple example of a query that returns the price and the title of items that have a price of less than 30.

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x ns:price ?price .
        FILTER ?price < 30 .
        ?x dc:title ?title . }
```

Together with the ability to define flexible graph patterns, this provides a quite powerful language for matching RDF on the data level without considering the schema.

Alternative Query Modes

While the use of the SELECT construct to determine bindings for a set of query variables is the standard use for SPARQL, the language also has other modes of use. In particular, the current version of the specification makes provisions for two kinds of alternative uses. The first is the use as a transformation language. In this mode, the SELECT part of the query is replaced by a CONSTRUCT part that specifies an RDF Data structure to be filled with values bound to return variables. For each matching

combination of values, the RDF structure is instantiated and stored as part of the result. In this way RDF data can be transformed from one structure to another. The example below describes a query that transforms FOAF data into the vcard format.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?x  vcard:N _:v .
          _:v vcard:givenName ?gname .
          _:v vcard:familyName ?fname }
WHERE
 {
   { ?x foaf:firstname ?gname } UNION  { ?x foaf:givenname   ?gname } .
   { ?x foaf:surname   ?fname } UNION  { ?x foaf:family_name ?fname } .
 }
```

The second alternative way of using the language is to ask yes/no questions. In this case, the query expression that would normally be in the WHERE part of the query is used in combination with the keyword ASK. Such a query returns 'true' if the query expression could be matched with the data. Otherwise the query returns 'false'. The query below asks if there is a person called Alice that has a particular email address.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" ;
               foaf:mbox  <mailto:alice@work.example> }
```

Multiple Graphs

An aspect that has been neglected in most previous proposals is the ability to query data from multiple RDF graphs. In the se previous languages, often the only way of distinguishing different sources of information was in terms of namespaces. As different models, are often using the same namespace, this is not a satisfactory solution to the problem of distinguishing sources. SPARQL addresses this problem by providing the possibility to explicitly refer to named sources using the keyword GRAPH. In this case the sub-expression that refers to a certain graph is grouped in a block. The query below shows an example where data from two different FOAF files is combined in the query:

```
 SELECT ?mbox ?age ?ppd
WHERE
{
  GRAPH data:aliceFoaf
  {
    ?alice foaf:mbox <mailto:alice@work.example> ;
           foaf:knows ?whom .
    ?whom  foaf:mbox ?mbox ;
           rdfs:seeAlso ?ppd .
    ?ppd  a foaf:PersonalProfileDocument .
  } .
  GRAPH ?ppd
  {
     ?w foaf:mbox ?mbox ;
       foaf:age ?age
  }
```

```
}
```