Title: Questions for bioRDF

Author: Fred Zemke
Date: September 5, 2006

References

[SPARQL CR] "SPARQL query language for RDF", Candidate Recommendation, http://www.w3.org/TR/2006/CR-rdf-sparql-query-20060406/

[rq24] "SPARQL query language for RDF", Editor's draft, http://www.w3.org/2001/sw/DataAccess/rq23/24 (the version I printed was dated 2006/06/12)

[RDF primer] "RDF primer", Recommendation, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[RDF semantics] "RDF semantics", Recommendation, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

1. Introduction

I am a newcomer to semantic technology; my background previously has been with relational databases. I became interested in SPARQL because it is an emerging query language for data that is not helpfully served by SQL.

As I have gotten involved in SPARQL, I have encountered assumptions and mindsets that are contrary to my experience with relational databases. The purpose of this paper is to ask users of RDF about their expected use scenarios, to either confirm or deny that my experience with relational databases is relevant to the semantic space.

My concerns revolve around three topics: blank nodes, duplicates, and counting.

1.1 Blank nodes in RDF

[RDF primer] introduces blank nodes in section 2.3 "Structured property values and blank nodes", where it says in part "Structured information ... is represented in RDF by considering the aggregate thing to be described ... as a resource, and then making statements about that new resource... This way of representing structured information in RDF can involve generating numerous "intermediate" URIrefs ... to represent aggregates... . Such concepts may never need to be referred to directly from outside a particular graph, and hence may not require "universal" identifiers." The solution is the notion of blank node, which is a node of an RDF graph that has no IRI or literal value.

I constructed an elementary example that embodied one of the key issues discovered during the early days of relational technology. The example involves purchase orders. Each purchase order has multiple line items. Each line item consists of a part number, quantity and unit price. It looks like a line item is a candidate for treatment as a structured data value as described by the [RDF primer]. Here is an example of a graph for a purchase order:

```
/* graph 1 */
:po po:lineitem _:line1 .
_:line1 po:part 'xyz' .
_:line1 po:quantity 1 .
_:line1 po:price 12.99 .

:po po:lineitem _:line2 .
_:line2 po:part 'xyz' .
_:line2 po:quantity 1 .
_:line2 po:price 12.99 .
```

This example features a purchase order indicated by the IRI :po, with two line items, indicated by the blank nodes _:line1 and _:line2.

Or does it have two line items? Notice that the part, quantity and price in each of the line items is the same. Thus the line items are duplicates of one another. This is a realistic scenario in actual purchase order applications.

[RDF semantics] section 1.5 “Blank nodes as existential variables” says “Blank nodes are treated as simply indicating the existence of a thing, without using, or saying anything about, the name of that thing.” In this light, the two blank nodes both assert “there exists a line item whose part is ‘xyz’, quantity is 1 and price is 12.99”. From this, one cannot conclude that there exists *two* line items whose part is ‘xyz’, quantity is 1 and price is 12.99.

Examples like this one are very important to relational technology. Based on mathematical set theory, the early theorists called for duplicate elimination, and some products did that (you can even find some theorists who still hold that position). However, duplicates happen regularly in real-world applications, and the users demanded that duplicates should not be eliminated. For example, to compute the total price of the purchase order, users insist that it is incorrect to suppress the duplicates and only total the distinct prices.

So relational databases surrendered this point to pragmatism; my question is whether there is a similar issue facing RDF.

Arguably, the example is not good RDF. Certainly it clashes with [RDF semantics] treatment of blank nodes as existentially quantified variables. [RDF semantics] is regarded as more definitive than [RDF primer].

We should also note that with RDF’s “open world” orientation, it is conceivable that the two line items were contributed to the graph separately. Each contributor was providing a true fact, “the purchase order :po has a line item with part ‘xyz’, quantity 1 and price 12.99”.

Even in the primer's terms, if one wanted to assert that the purchase order had a particular list of line items, no more and no less, then one should use an RDF collection, which is a linked list.

Using a linked list for the example is feasible but burdensome. The rewritten graph looks like this:

```
/* graph 2 */
:po po:lineitemlist _:listhead .
_:listhead rdf:first _:line1 .
_:listhead rdf:rest _:listcontinued .
_:listcontinued rdf:first _:line2 .
_:listcontinued rdf:rest rdf:nil .
_:line1 po:part 'xyz' .
_:line1 po:quantity 1 .
_:line1 po:price 12.99 .
_:line2 po:part 'xyz' .
_:line2 po:quantity 1 .
_:line2 po:price 12.99 .
```

This does create the requisite distinction between _:line1 and _:line2, because they are reached along different paths from the starting point, :po. However, it has introduced a big headache from a processing standpoint, because you have to walk the list to find all line items. Walking a list is a recursive task. SPARQL does not have any recursive capabilities. There is no SPARQL query that will get the line items in the revised graph. Of course, SPARQL might be enhanced with recursion, but I think it is noteworthy that SQL avoids the need for recursion until it encounters more complicated data structures than lists.

As a compromise solution, the user might keep the initial flattened design but use IRIs instead of blank nodes to represent each line item. In that case the data might be

```
/* graph 3 */
:po po:lineitem :poline1 .
:poline1 po:part 'xyz' .
:poline1 po:quantity 1 .
:poline1 po:price 12.99 .

:po po:lineitem :poline2 .
:poline2 po:part 'xyz' .
:poline2 po:quantity 1 .
:poline2 po:price 12.99 .
```

Some would say that you still have not really asserted that there are two distinct line items — how do you know that :poline1 and :poline2 are not two names for the same thing? From this standpoint, it is commonly stated that you need OWL in order to make deductions about cardinality. However, many are willing to use the "distinct names" principle, which treats two things with distinct names as distinct.

My summary is that one could argue that this is a question of educating the user about the right and wrong uses of RDF. But I come back to the experience in the relational database world, where we found that sometimes something that is correct theoretically must yield to user expectations. In the case of RDF, I am wondering whether users will really embrace the existential interpretation of blank nodes. Compounding the problem, the correct way to handle this example requires lists, which are cumbersome to process, so that even if a user is properly educated about the correct way to do things, there will be incentives to do it the wrong way.

1.2 Blank nodes in SPARQL

So far I have only talked about representing duplicate data. Now let's talk about querying it. To parallel the [RDF semantic]'s notion of a blank node as an existential variable in the data, SPARQL has the notion that a blank node identifier in the query should also be treated existentially.

Let's look at some examples. The user would like to know the total price of the purchase order. Here is the user's first attempt at a query:

```
SELECT ?quantity ?price
WHERE { :po po:lineitem _:line1 .
        _:line1 po:quantity ?quantity .
        _:line1 po:price ?price . }
```

The user wrote this query by "cut and paste" from a serialization of graph 1. The ability to cut-and-paste like this is one of the design goals of SPARQL.

This query will not find the desired answer, because of the existential interpretation of the blank node identifier _:line1.

Note that I am talking about the interpretation of the _:line1 in the query, not in the data, at this point. The example is interpreted as "Find pairs (?quantity, ?price) such that there exists _:line such that { :po po:lineitem _:line1 . _:line1 po:quantity ?quantity . _:line1 po:price ?price. }." With the sample data, there is only one pair (1, 12.99) such that there is a line item whose quantity is 1 and whose price is 12.99.

The breakdown occurs because of the blank node in the query. It would not help if the data were graph 3, which has no blank nodes. (Of course, if the data has no blank nodes, then the user can't cut-and-paste them to start writing his query, but if the user on his own places a blank node identifier in the query, he runs into this issue.) As for graph 2, SPARQL has no ability to walk a linked list, so the user could not even get this far with graph 2.

The user can avoid the implicit existential treatment of blank nodes in the query by rewriting the query like this:

```
SELECT ?quantity ?price
WHERE { :po po:lineitem ?line .
        ?line po:quantity ?quantity .
        ?line po:price ?price . }
```

This rewrite substitutes the variable ?line for the blank node identifier _:line. Variables are not treated existentially. The rewritten query is performed conceptually in two stages:

1. Find (?line, ?quantity, ?price) such that { :po po:lineitem ?line . ?line po:quantity ?quantity . ?line po:price ?price . }

   This stage finds two results: (_:line1, 1, 12.99) and (_:line2, 1, 12.99).

2. Project the solution sequence from stage 1 to retain only (?quantity, ?price).

   This stage finds two results: (1, 12.99) and (1, 12.99).

Again, this may be a question of user education. The distinction is pretty clear: use a blank node identifier in a query if you want duplicates factored out, use a variable if you want duplicates retained. On the other hand, I wonder if users might inadvertently use a blank node identifier in a situation in which the user really wants a variable.

1.3 Summary questions

My questions to BioRDF:

1. Do you view blank nodes as a device to create data structures without bothering with IRIs, or do you view them as existential assertions?
2. Do you expect to use linked lists? Do you expect to use what I call flattened lists?
3. Do you expect to use RDF in an “open world” fashion, or a “closed word”?
4. Do you expect to assume “distinct names”?
5. Do you want SPARQL to treat blank node identifiers existentially, or just the same as variables?

- End of paper -