Delta: an ontology for the distribution of differences between RDF graphs

Tim Berners-Lee and Dan Connolly*

MIT Computer Science and Artificial Intelligence Laboratory (CSAIL)

Abstract. The problem of updating and synchronizing data in the Semantic Web motivates an analog to text diffs for RDF graphs. This paper discusses the problem of comparing two RDF graphs, generating a set of differences, and updating a graph from a set of differences. It discusses two forms of difference information, the context-sensitive "weak" patch, and the context-free "strong" patch. It gives a proposed update ontology for patch files for RDF, and discusses experience with proof of concept code.

1 Introduction

The use of text files to record programs, documents, and other artifacts is supported by version control systems such as RCS[1] and CVS[2] that are based on the ability to compute the difference between two text files and represent it as diff[3], i.e. a set of editing instructions. The use of database tables to record bank accounts and records of all sorts is supported by the relational calculus[4] and its expression as SQL statements. In both cases, the data goes thru a sequence of states; not only are the states represented explicitly (as text files or database tables) but also the transitions from one state to the other can be represented explicitly (either as editing instructions or SQL insert/update statements). Difference (Δ) and sum (Σ) functions are ubiquitous in computing and, like differentiation and integration, are inverse in the sense that:

$$v1 = \Sigma(v0, \Delta(v0, v1)) \tag{1}$$

Since the transitions can be represented much more compactly than the pairs of states, and the sigma function is straightforward to compute, the deltas are useful for efficiently updating data distributed among two or more peers.

We are developing a Semantic Web Application Platform (SWAP) including tools and applications to manipulate RDF graphs much like traditional tools manipulate text files. It includes `cwm`, a command-line tool for processing RDF in both the standard XML encoding[5] and an experimental encoding, Notation3 (n3)[6].

* This work is supported in part by funding from US Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0593, "Semantic Web Development".

As we build the Semantic Web, using RDF graphs[7] to represent data such as bibliographies[8], syndication summaries[9] and medical terminology[10], we see a need for difference and sum functions for RDF graphs. The use of RDF to represent test results[11],[12] motivates better ways to compare the actual results of software tests with the intended results and isolate the differences.

1.1 The Synchronization Problem

One of the most stubborn problems in practical computing is that of synchronizing calendars and address books between different devices. Various combinations of device and program, from the same or different manufacturers, produce very strange results on not-so-rare occasions.

The problem has three parts. There is the syntactic problem of extracting the data from the strange device or its storage medium and turning into something manageable, such as RDF. There is the semantic problem of understanding what the fields mean: can one have two home phone numbers? There is the problem of actually synchronizing changes, particularly in the general case that changes have been made on both devices.

Because the direct syntactic conversion to RDF often leaves something which has strained and awkward semantics, it is often necessary or tempting to mix the semantic and syntactic conversions. Because the merging of changes requires more application knowledge than the bare RDF data provides, it is tempting to mix the conversion and sync algorithm. However, this mixing reduces the modularity and testability of the resulting program. Perhaps if the three stages were separated, then a more robust system would result, and one more extensible by the addition of information in new ontologies.

In the semantic web architecture, the application constraints on the data can be represented in the ontology, and so can be used by a a generic synchronization system.

On the one hand, the syntactic problems are straightforward, if tedious, and the much harder semantic problems may explain why many existing synchronization packages break down. But on the other hand, perhaps it is the combination of the two that result in so many failures; perhaps software that separates the problems, treating synchronization generically, will be more robust. We hope this work contributes to further work on specifications such as SyncML[13].

And while in the general case, concurrent changes may be completely irreconcilable, the diff mechanisms discussed here solve an interesting part of the problem space.

1.2 Problems with the line-oriented approach

RDF graphs can be serialized and used with traditional line-oriented tools. In the general case, with no constraints on how the graphs are serialized, line-oriented deltas can be as large as the data itself, even between files representing the same graph. However, when files are edited by hand, small changes to the

data naturally result in small textual diffs. But since the difference is expressed as the difference between two text files, not the difference between two graphs, the delta is dependent on the graph serialization. It's not enough to have the original graph to use the delta; one needs a copy of the particular serialization.

Pretty-printing algorithms reduce the large number of possible serializations of an RDF graph to a few actual serializations. The difference engine[14] produces human-readable difference descriptions using an algorithm analogous to comparing pretty-printed graphs; its descriptions are not sufficient to reconstruct one graph from the other, however.

We find it practical to use CVS to manage both hand-edited and machine-serialized RDF data files in many cases. A notable exception is the reference results for tests: comparison of experimental test results versus reference results yield many false mismatches every time we change the pretty-printing algorithm in the slightest. The cost of managing the reference results this way is barely tolerable.

The straightforward pretty-printing algorithm works in the obvious way when all the nodes are named (either with URIs or literals): triples are sorted by subject, and those that share a subject are grouped together. Notation3 has syntax for grouping triples that shared predicates. Unlabelled nodes (*blank nodes* or *bnodes*) that have no incoming triples are treated like named subjects. Bnodes that have one incoming link serve as internal nodes in the pretty-printing tree. Bnodes that have more than one incoming triple are given arbitrary labels for the purpose of serialization and are hence treated like named subjects. For example, the triples

Example 1.
```
:Bob :pet _:p.
_:p :size "small".
:Bob :brother :Pete.
_:p :mother _:p2.
:Pete :pet _:p2.
```

are pretty-printed as

Example 2.
```
    :Bob     :brother :Pete;
         :pet  [
             :mother _:g0;
             :size "small" ] .

    :Pete     :pet _:g0 .
```

The ordering and the identification of bnodes are the two ways which serializations of the same graph can arbitrarily differ. `Cwm` not only attempts to find a serialization which minimizes the number of arbitrarily named nodes but often happens to regenerate arbitrary names consistently across runs. Even so, diffs of pretty-printed RDF are still unsatisfactory, since changes as small as one triple

can lead to arbitrarily large textual diffs if that triple changes the set of bnodes that need arbitrary labels.

To completely eliminate the arbitrary choices in how to serialize an RDF graph, we could employ a canonicalization algorithm such as the one[15] in Jena[16], or cant.py from our own SWAP toolkit. One problem with this approach is that the canonical form is expressed in the N-Triples[17] representation. Deltas between N-Triples files are verbose and tedious to read for most practical graphs. Further, the problem of large textual diffs resulting from small changes remains: these canonicalization algorithms work by computing a signature for each blank node based on nearby triples and sorting the results; adding or removing one triple near a blank node will change its signature and hence potentially the labelling of many bnodes.

2 Delta and Sigma for RDF Graphs

SQL statements and text file diffs are attractive because they succinctly represent the difference between two states. If the difference between two text files were not much smaller than either of the text files, it would be of little use. The essential feature of a difference algorithm, then, is *economy*: small differences between input states should result in small deltas.

An RDF graph is a set of (subject, predicate, object) triples, i.e. a set of typed links between nodes. Each node may or may not be named (either by a URI or a literal). As a measure of the size of the difference between two RDF graphs $G1$ and $G2$, one can use the sum of the size of the set differences $|G1-G3|$ and $|G2-G3|$ where $G3$ is the largest common subgraph of $G1$ and of $G2$.

2.1 Computing differences between RDF graphs

In the case in which all the nodes are named, computing the difference between two graphs is simple and straightforward:

Definition 1. *If* $G1$ *and* $G2$ *are ground RDF graphs, then the* ground graph delta *of* $G1$ *and* $G2$ *is a pair* $(insertions, deletions)$ *where insertions is the set difference* $G2 - G1$ *and deletions is* $G1 - G2$.

This form of delta is reasonably economical: the storage cost is linear in the size of the difference between the graphs. Straightforward extensions with slightly improved economy might be more specific in expressing differences in which only one or two parts of the triple have changed.

In the case where not all of the nodes are named, finding the largest common subgraph becomes a case of the graph isomorphism problem. The arc labels do have names (in a very large set of practical cases, including all those which can be serialized as RDF/XML). Graph isomorphism is in fact a class of difficult problem that cannot be solved in polynomial time but which has not been shown to be NP complete[18]. While the general graph isomorphism problem has readily

available solutions[19][20], they do not seem to be a good match for the practical cases of RDF graph diff.

There is an interesting subset of real cases in which there are a mixture of named and unnamed nodes, but none of the unnamed nodes is very far from a named node. In this case, the unnamed nodes can be indirectly identified by giving a path from a named node. The difference is then expressed by giving this local context and the related changes.

2.2 A patch file format for RDF deltas

By analogy to the text diff, there is a need not only for a difference-finding algorithm, but for a patch file format. Such a format needs:

- a way to uniquely identify what is changing
- a way to distinguish between the pieces added and those subtracted

It is straightforward to pinpoint the parts of the graph that have changed when all nodes are named, but less so in the presence of anonymous nodes.

To identify what is changing, we use Notation3 expressions for quoted RDF graphs with schema variables, and we introduce three new terms. For example:

Example 3.

```
@prefix diff: <http://www.w3.org/2004/delta#>.
{ ?x  bank:accountNo "1234578"; bank:balance 4000}
 diff:replacement
{ ?x  bank:accountNo "1234578"; bank:balance 3575}.
```

This one new property `replacement` can express any change. Deletions can be written {...} `diff:replacement` {} and additions can be written {} `diff:replacement` {...}.

The second alternative is very similar but involves two properties, one for inserting and one for deleting:

Example 4.

```
{ ?x  bank:accountNo "1234578"}
  diff:deletion  { ?x  bank:balance 4000};
  diff:insertion { ?x  bank:balance 3575}.
```

The form using `diff:insertion` and `diff:deletion` is implemented in cwm.

The first and second form are related by

Definition 2.

```
{ ?F replacement ?G }    <=>  { ?F deletion ?F; insertion ?G }
```

2.3 Weak and Strong diffs

Much of the popularity of CVS is due to its support of concurrent development. It makes a patch file[21] representing the changes each party has made. These changes are made, in order, to the repository file to generate new versions. In

the event that two agents take a copy of the same version $v0$ and make different changes to it ($v1a$ and $v1b$), the party that commits last attempts to make $v1$ which incorporates both diffs:

$$v1 = \Sigma(\Sigma(v0, \Delta(v0, v1a)), \Delta(v0, v1b)) \tag{2}$$

Note that $\Delta(v0, v1b)$ is applied to something other than v0. The context diff and unidiff formats are sufficiently robust that it does work in most practical cases. When it does not work, then the user is left with the problem of manually reconcilling the conflicts. This happens when, for example, one party moves the date of a meeting at the same time as someone else moves or deletes the meeting. It may be that the criterion that a problem needs human involvement is very application-dependent.

There are two failure modes:

1. Inconsistent changes were made. This failure mode is not automatically soluble.
2. The patch was incapable of finding the appropriate points in V1a at which to make the change Δ(v0, v1b). This form of failure we can eliminate for RDF graph deltas.

To address the latter failure mode, we distinguish two types of RDF graph deltas: a *weak* delta gives enough information to apply it to exactly the graph it was computed from, but a *strong* delta specifies the changes in a context-independent manner. The difference is not in the patch file format, but in the information a particular patch gives.

Returning to the bank example, if bank account numbers are globally unique, then the replacement pattern will bind ?x to a node identifying a particular bank account. In OWL[22] terms, if `bank:accountNumber` is an `owl:InverseFunctionalProperty`, then the node must be the `owl:sameAs` any other node with the same account number. In that case, the patch will be strong.

If however, many accounts can have the same number, applying that patch to another knowledge base may inadvertently alter the wrong account. The patch is weak.

The pattern for terms goes as follows:

Definition 3. *Given a background ontology W and a graph G, if a blank node b in G is the object of a triple whose subject v is* functionally ground *and whose predicate p is an* `owl:FunctionalProperty` *according to W, then $v.p$ is a* functional term label *for b in G with respect to W. Likewise, $v\hat{q}$ is a functional term label for b if q is an* `owl:InverseFunctionalProperty`*, b is the subject, and v is the object. Recursively, v is functionally ground if it is a name (URI or literal) or a bnode with a functional term label.*

Then we can rewrite certain graphs:

Definition 4. *With respect to a background ontology W, a graph G is* fully labelled *iff every node in G is functionally ground. A* functional RDF graph *is*

a set of triples whose terms are URIs, literals, or functional terms. A functional RDF graph F is a functional analog *of an RDF graph G iff G is fully labelled and F can be obtained from G by replacing each bnode b in G with a functional term label for b.*

The diffs of functional RDF graphs are just as simple to make as ground RDF deltas:

Definition 5. *Given a background ontology W, a* strong *delta between fully labelled graphs $G1$ and $G2$ is a pair $(insertions, deletions)$ where insertions is the set difference $F2 - F1$, deletions is $F1 - F2$, and $F1$ and $F2$ are functional analogs of $G1$ and $G2$ respectively.*

A strong delta is like a context diff that cannot be mis-applied.

Proposition 1. *If D is a strong delta between fully labelled graphs $k1$ and $k2$, and $k3$ is a subset of $k1$, then $\Sigma(k3, D)$ is consistent with $k2$.*

One advantage of a strong patch is, then, that one can take a patch from any true knowledge base change and apply it to a subset knowledge base, and the result will be true. For example, if changes to a knowledge base are represented by a sequence of strong diffs, one can subscribe to the diffs from any given point on, and acquire a subset of the final knowledge base.

As a practical matter, achieving fully labelled graphs requires care in building and using the ontology. As a supplement to the good practice of using URIs to distributing data, it is useful to identify things indirectly by using terms with published ontologies that say whether they are many-many, many-1, 1-many or 1-1. The diff.py program from SWAP will generate a strong diff between two files, provided it can find sufficient information in the Web to fully label the input graphs.

3 Application to Update and Sync

Though we have made small scale tests, we are interested in pursuing strong diffs are useful in a variety of applications.

3.1 Peer-peer update and sync

The algorithm for synchronizing two databases can be straightforwardly generalized to N. In a decentralized peer-peer network such as Network News Transfer Protocol[23] (or many others), messages are timestamped and distributed eventually to every party, though a message may be received by different parties at different times. When the network is reliable, there may be a well-defined maximum delivery time.

A crude algorithm is to apply the patches in order of the timestamp. If a message arrives with a timestamp preceding the recent ones already taken into

account, they are unwound so that the new version can be built in the proper order. A patch which fails (as in a CVS conflict) is rejected. In the case of RDF graphs, failure can be a pre-agreed form of consistency, such as (for example) OWL-DL consitency. The sender of the failed patch will realize this as they will be running the same algorithm on the same patches, and will have to take recovery action.

A new version can be given a version id by hashing the version id of it predecessor with the message id of the patch used to make the new from the old. The community can refer to versions by these ids, and if they want to refer to a commonly held document, then one only has to wait for the maximum delivery time to know that everyone in the community will know the value of the knowledge base for that version. Even without waiting, anyone who knows of a version with that ID will know they have the same contents.

3.2 Patches as knowledge

The idea of the strong patch file format is interesting because a patch is a little bit of knowledge. A patch for example that where my phone number was 1234 it should now be 5678, when in the context in which it is known to be a change to a valid knowledge base between one week and the next, indicates that my phone number has actually changed. One might conclude, say, that I moved or changed jobs. A strong patch has meaning in itself, and distributing and filtering these becomes an interesting way of processing knowledge. In some areas (like houses for sale) it is the new changed information which is of most interest, and in some areas (like currency rates) if you listen to a stream of changes you will in fact accumulate a working knowledge of the area.

3.3 Patches as news

From the historical *NCSA Mosaic What's New* page to the current syndication of RSS streams [9], the interest in news on (or off) the Web demonstrates that there is great interest in changes to the status quo. We speculate that this will also be the case on the Semantic Web. When the state is represented in RDF, then RDF diffs represent news. The W3C Technical Reports list is available as RDF, and the W3C RSS feed is partly, effectively, a list of changes to the Tech Reports list. This could be formalized by explicitly distributing RDF diffs.

4 Future directions

The algorithm developed to date produces difference files only on graphs which are labelled directly with URIs or indirectly with functional properties or inverse functional properties.

It may be useful to extend the algorithm to cope with graphs which are not completely labelled, but where the unlabeled bits are the same in each graph,

and so a strong diff can still be produced. Another avenue would be to look at using more than one property to label a node when one is not sufficient.

To cater for applications when a weak patch file will suffice, the algorithm could be extended to do more of a canonicalization-style signature-based match to optionally give a weak diff where a strong diff cannot be given.

RDF does not contain the notion of an unordered set, though one can with OWL create a class which has an enumerated set of members. If the use of unordered sets becomes common, which the authors suspect would be wise in the long run, then a difference engine should be aware of such sets and be able to express differences between them.

This application, like the rule language, demonstrates the usefulness of the quoted formulae of n3. The authors believe that many applications will need this ability to quote RDF graphs within graphs. As n3 becomes a language of communication, difference files will of course have to express changes to nested formulae. As these are graphs, this is basically a straightforward recursive use of the difference system for single graphs. A simple though verbose alternative is to reify the n3 before building differences.

With these extensions, the simple difference file format may lose the elegance of its current simplicity. However, even with these extensions, most data and ontologies shipped around the web – the bottom layers of the semantic web layer cake – will be plain RDF graphs and so have simple difference files.

Clearly there are many algorithms which can be imagined for efficiently generating deltas for RDF graphs. The ones written are not particularly efficient, being designed as proof of concept.

5 Conclusions

There are many uses for technology of communicating differences between graphs or changes to a graph. While in general the generation of differences is basically a graph isomorphism problem, in a wide set of practical cases, one can efficiently generate a difference, or patch file. So-called strong patch files are particularly interesting, and open up a new series of applications based on the syndication of change information. However, to be able to generate them, one needs either a well-labelled graph, a ontological knowledge of inverse functional properties to allow nodes to be indirectly labelled. The patch file format proposed is simple, being a new ontology of only two (or three) new properties, and directly uses Notation3 syntax and semantics, which itself is a simple extension of RDF. This format can be generated by all sorts of difference-finding algorithms. It can be absorbed by any system capable of matching RDF subgraphs. The patch file ontology is a candidate for a future standard for remote update of RDF data.

References

1. Tichy, W.: Rcs–a system for version control. Software Practice & Experience **15** (1985) 637–654

2. Berliner, B.: Cvs ii: Parallelizing software development. In: USENIX Conference Proceedings, Washington, D.C. (1990) 341–352
3. Miller, W., Myers, E.W.: A file comparison program. Software—Practice and Experience **15** (1985) 1025–1040
4. Codd, E.F.: A relational model of data for large shared data banks. Communications of the ACM **13** (1970) 377–387
5. Beckett, D.: Rdf/xml syntax specification (revised). http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ (2004)
6. Berners-Lee, T., Hawke, S., Connolly, D.: Semantic web tutorial using n3. Twelfth International World Wide Web Conference (2003)
7. Klyne, G., Carroll, J.J.: Resource description framework (rdf): Concepts and abstract syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (2004)
8. Beckett, D., Miller, E., Brickley, D.: Expressing simple dublin core in rdf/xml (2002)
9. Beged-Dov, G.e.a.: Rdf site summary (rss) 1.0. http://web.resource.org/rss/1.0/ (2000)
10. Golbeck, J., Fragoso, G., Hartel, F., Hendler, J., Parsia, B., Oberthaler, J.: The national cancer institute's thesaurus and ontology. Journal of Web Semantics **1** (2003)
11. Chisholm, W., Palmer, S.B.: Evaluation and report language (earl) 1.0. http://www.w3.org/TR/2002/WD-EARL10-20021206/ (2002)
12. Carroll, J.J., De Roo, J.: Owl web ontology language test cases. http://www.w3.org/TR/2004/REC-owl-test-20040210/ (2004)
13. : Syncml specifications, version 1.1. http://www.openmobilealliance.org/tech/affiliates/syncml/syncmlindex.html (2002)
14. Klyne, G.: Semantic web inference scripting in haskell. http://www.ninebynine.org/RDFNotes/Swish/Intro.html (2004)
15. Caroll, J.J.: Signing rdf graphs. Technical Report HPL-2003-142, Hewlett-Packard (2003)
16. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the semantic web recommendations. Technical Report HPL-2003-146, Hewlett-Packard (2003)
17. Grant, J., Beckett, D.: Rdf test cases. http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/ (2004)
18. Köbler, J., Schöning, U., Torán, J.: The Graph Isomorphism Problem: Its Structural Complexity. Progress in Theoretical Computer Science. Birkhäuser, Boston, MA (1993)
19. Skiena, S.: The Algorithm Design Manual. Telos Pr, New York (1997)
20. Skiena, S.: Graph isomorphism. In: Stony Brook Algorithm Repository. Stony Brook University (2001)
21. Wall, L.e.a.: patch. http://www.gnu.org/software/patch/patch.html (2000)
22. Schreiber, G., Dean, M.: Owl web ontology language reference. http://www.w3.org/TR/2004/REC-owl-ref-20040210/ (2004)
23. Kantor, B., Lapsley, P.: Network news transfer protocol. Technical Report RFC977, IETF (1986)