

# Harvesting RDF triples

Joe Futrelle

Natioanl Center for Supercomputing Applications  
1205 W. Clark St., Urbana IL 61801, US  
futrelle@ncsa.uiuc.edu

**Abstract.** Managing scientific data requires tools that can track complex provenance information about digital resources and workflows. RDF triples are a convenient abstraction for combining independently-generated factual statements, including statements about provenance [1]. Harvesting is a strategy for asynchronously acquiring distributed information for the purposes of aggregation and analysis [2]. Harvesting typically requires that information be temporally scoped and attributed to some creator or information source. An RDF triple asserts a fact without attributing it to any actor or period of time, so the abstraction must be extended to support typical harvesting scenarios. This paper compares standard, conventional, and non-standard means of extending RDF triples to associate them with attribution and timing information. Then, it considers the implications of these techniques for harvesting and presents some implementation sketches based on a journaling strategy.

## 1. Introduction

In NCSA's Cyberenvironments project (<http://www.ncsa.uiuc.edu/Projects/>), the need to capture provenance from multiple, distributed, heterogeneous portal and workflow tools has led to the development of an RDF harvesting strategy. Because it is impractical to retool every aspect of the complex technical infrastructure used in science to support RDF API's, tools, and implementations, our approach is to wrap these tools in middleware that makes minimal assumptions about its environment, building consensus around simple abstractions such as log files and journals. The combination of a bottom-up implementation strategy with RDF's descriptive power has given our social networking and data mining efforts diverse new sources of provenance information with relatively little investment in new frameworks, protocols, and API's.

## 2. Conceptual overview

**Facts and contradictions.** Resolving contradictions across multiple, independently-produced RDF graphs requires rules that sometimes depend on second-order descriptions. For instance, imagine a reasoner that resolves contradictions between pairs of statements by ordering them alphabetically and rolling loaded dice. The dice-loading parameters are "global" in that they are independent of the statement pairs. Now imagine a reasoner that resolves contradictions between pairs of statements by selecting the statement that was made by the most trusted actor. In that case the trust parameters are "local" in that they are derived in part from second-order information about which actor made which statement [3].

**Actors and statements.** Actors in a distributed environment produce and consume information. Since it often matters which actor produced which information, representing

the information produced by actors as triples is insufficient to enable reasoning about the information, since the triples alone do not contain the context required to parameterize rules that depend on who said what. If actors can be uniquely identified, managing that contextual information amounts to associating an actor's identity with each triple or set of triples. Contradictions can then be resolved in any number of ways, e.g. trust ranking.

**Time and negation.** In general, models change over time [4]. RDF has no standard means to scope assertions temporally, which would enable contradictions to be resolved based on sequencing and other temporal logic. Time is an especially useful kind of contextual information, because it is “global” and therefore does not require coordination between data producers beyond clock synchronization, a solved problem [5].

### 3. Representation

RDF reification provides a standard way of associating arbitrary contextual information with any triple [6], including attribution and timing. However, representing attribution and temporal information about triples *efficiently* requires constructs that aren't available in RDF.

**Reification.** Second-order descriptions in RDF are achieved via *reification*, in which an RDF triple  $p(S, O)$  is represented by, at minimum, four triples: `rdf:type(T, rdf:statement)`, `rdf:subject(T, S)`, `rdf:predicate(T, p)` and `rdf:object(T, O)` where T is a URI uniquely identifying the triple. Any other contextual information can be represented by triples in the form  $p'(T, O')$  for any  $p'$  and  $O'$ . This is inefficient to implement, because it multiplies the number of triples that must be processed at least fivefold, and requires the generation and management of a globally unique ID for each triple.

**Journaling.** In an application, an RDF graph must be built procedurally, one triple at a time. In a distributed environment, a number of actors may modify a graph over time by adding and deleting triples. By logging each modification in a journal (e.g., log file), sufficient contextual information can be gathered to enable attribution and time-based decisions. For instance, the actor responsible for each modification, the type of modification (add/delete), and the time of the modification can be recorded along with the triple being modified. Computing time and actor-scoped subgraphs from the journal is simple, but its worst-case performance is linear with the number of modifications. A journal is a convenient source of time and actor-scoped triples, but not an efficient means of accessing them by attribution and time.

**n-tuples.** A simple way of adding information to a tuple is to add one or more terms. Declarative systems such as Prolog allow arbitrarily many terms per statement. RDF allows a fourth term for literal types, but is not generally extensible to n-tuples.

Attribution and time can be added to RDF tuples by adding an actor term and a time interval term, or an actor term and two time terms for the start and end of the time interval. Since these n-tuples cannot be represented as RDF statements, a different representation is required which can be processed to produce time and actor-scoped subgraphs. Given such a representation, useful classes of problems concerning attribution and time can be resolved directly against it using abstractions that do not support efficient graph traversal, such as SQL.

## 4. Harvesting

Harvesting is a strategy for distributing data in which clients typically retrieve data from servers asynchronously and store it for later processing. Harvesting clients make decisions about which data to retrieve based on contextual information about how likely it is to have changed since it was last retrieved (e.g., HTTP caching directives) or by partially exposing their decision criteria to a server (e.g., OAI-PMH date range queries).

**Harvesting triples.** It is not generally possible to harvest every triple from a triple store without contextual information, for the same reason that it is not generally possible to index every page on the web—both the web and RDF graphs are not guaranteed to be completely connected. Minimally, a graph store must expose information about which subgraphs may have changed over a given time interval to enable a harvesting client to walk the graph and find all information newer than the start of the time interval. Maximally, a graph store could expose a complete change log, as in the journaling strategy.

**Multi-tier approach.** To improve efficiency, several processing stages can be interposed between actors modifying a triple store and reasoning engines carrying out high-level operations such as rules-based inference. Figure 1 introduces a three-tier approach.

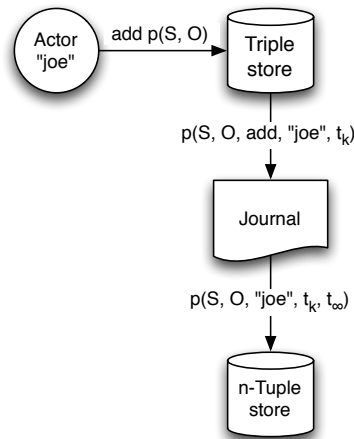


Figure 1. Harvesting n-tuples from a journaling triple store

In this example, an actor (“joe”) adds a triple  $p(S, O)$  to a triple store. The triple is journaled along with operation (“add”) actor (“joe”) and timing ( $t_k$ ) information. An n-tuple store, consuming the journal entry, adds an entry recording an open time interval ( $t_k$  to  $t_\omega$ ). If the actor then deletes  $p(S, O)$ , a journal entry  $p(S, O, \text{delete}, \text{“joe”}, t_{k+1})$  will be written and the n-tuple store will close the interval by modifying the relevant tuple as  $p(S, O, \text{“joe”}, t_k, t_{k+1})$ .

The n-tuple store can make decisions about what triples to harvest using simple range and set membership queries. For instance it could trivially reject modifications from untrusted actors or delete all tuples with closed time intervals as a means of discarding non-current information.

Actors can write journal entries directly to a file or network stream instead of modifying a triple store. Any number of simple text formats suffice for representing and trans-

porting journal entries, and journals generated by independent actors can be merged using simple, generic operations. The scenario is illustrated in Figure 2.

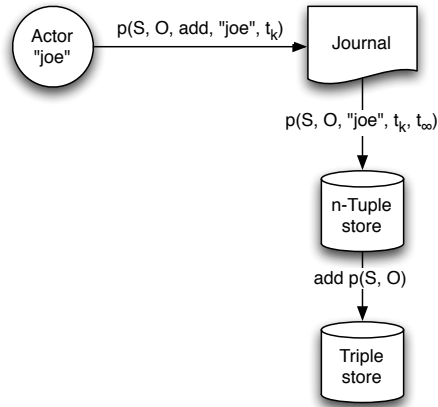


Figure 2. Harvesting triples from a journaling actor

In this example, the triple store is populated with a triple added by means of an actor (“joe”) writing a journal entry. The advantage of this scenario is that because the journal is rolled up into an n-tuple store containing attribution and timing information about each triple, the triple store can be populated only with the triples relevant to some reasoning task with respect to attribution and timing.

## 5. A practical implementation

Harvesting triples can be accomplished through the multi-tier approach outlined above. In this section, a practical proposal for implementing that approach is outlined. The strategy outlined in this section is currently being used in NCSA’s Cyberenvironments project (<http://www.ncsa.uiuc.edu/Projects/index.html>) to link workflow provenance to social networking analysis codes.

**Journaling.** In this section we present an RDF representation of journal entries and several format realizations, including standard RDF/XML and nonstandard extensions to NTriples.

The proposed RDF representation is based on reification, which is used in conjunction with a proposed vocabulary representing attribution and timing. We propose that attribution and timing information for each triple be represented using Dublin Core `creator` and `date` properties, using an actor URI for the value of the `creator` element and an ISO 8601 timestamp for the value of the `date` element. To denote the operation the actor performed on the model (e.g., add or delete) we introduce the `wsww:operation` property where `wsww` is a prefix for the proposed “who said what when” namespace URI <http://tupeloproject.org/wsww>. Possible values for the `wsww:operation` property are `wsww:add` and `wsww:delete`.

Figure 3 shows an RDF/XML representation of a journal entry in which an actor identified with the URI <http://example.org/example#joe> has deleted a triple at 15:09 UTC on December 1, 2005:

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:wsw="http://tupeloproject.org/wsw">
<rdf:statement rdf:about="http://example.org/example#genid927">
  <rdf:subject rdf:resource="http://example.org/example#someSubject"/>
  <rdf:predicate rdf:resource="http://example.org/example#somePredicate"/>
  <rdf:object rdf:resource="http://example.org/example#someObject"/>
  <dc:creator rdf:resource="http://example.org/example#joe"/>
  <wsw:operation rdf:resource="http://tupeloproject.org/wsw#delete"/>
  <dc:date rdf:dataType="http://www.w3.org/2001/XMLSchema#dateTime">
    2005-12-01T15:09:00Z
  </dc:date>
</rdf:statement>
</rdf:RDF>

```

Figure 3a. A journal entry in RDF/XML

One problem with an RDF representation of journal entries is that any application writing a journal must produce a unique ID for the each statement being reified. To solve this problem, we extend the NTriples format so that each line contains not just the subject, predicate, and object but also the actor URI, operation, and an ISO 8601 timestamp.

In this proposed notation, the example in Figure 3 is written on a single line as follows (← represents the continuation of a line):

```

<http://example.org/example#someSubject> ←
<http://example.org/example#somePredicate> ←
<http://example.org/example#someObject> ←
<http://example.org/example#joe> ←
<http://tupeloproject.org/wsw#delete> ←
2005-12-01T15:09:00Z .

```

This representation does not require generating a unique ID and can be compiled into standard representations should an application require it.

**n-Tuples in SQL.** This section outlines an SQL implementation of an n-tuple store which can be used to index a stream of journal entries. This example implementation has been designed for conceptual correctness and is not optimized.

The n-tuples can be represented using the following table definition:

```

CREATE TABLE ntuples (
  subject VARCHAR(255) NOT NULL,
  predicate VARCHAR(255) NOT NULL,
  object VARCHAR(255) NOT NULL,
  actor VARCHAR(255) NOT NULL,
  start_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
  end_time TIMESTAMP,
  CHECK (end_time IS NULL OR start_time <= end_time),
  CHECK (NOT EXISTS
    (SELECT * FROM ntuples nt
     WHERE nt.subject = subject
       AND nt.predicate = predicate
       AND nt.object = object
       AND nt.actor = actor
       AND (start_time, end_time) OVERLAPS
         (nt.start_time, nt.end_time)))));

```

The table corresponds closely to the abstract n-tuple model. In this representation, the `start_time` and `end_time` columns represent a half-open interval (containing the start time but not the end time) over which the triple is asserted to be a member of the set of all non-deleted assertions by the actor. A NULL `end_time` indicates that the actor has not deleted the triple since `start_time`. The CHECK constraints reject intervals whose end times precede start times, as well as overlapping identical statements by the same actor.

Adding a triple consists of performing an INSERT:

```

INSERT INTO ntuples
(subject, predicate, object, actor, start_time)
VALUES ('http://example.org/example#someSubject',
       'http://example.org/example#somePredicate',
       'http://example.org/example#someObject',
       'http://example.org/example#joe',
       '2005-11-28T03:09:00Z');

```

Deleting a triple consists of performing an UPDATE:

```

UPDATE ntuples
SET end_time = '2005-12-01T15:09:00Z'
WHERE subject = 'http://example.org/example#someSubject'
AND predicate = 'http://example.org/example#somePredicate'
AND object = 'http://example.org/example#someObject'
AND actor = 'http://example.org/example#joe'
AND end_time IS NULL;

```

Retrieving n-tuples based on trust is simple. Suppose the table `trusted_actors` contains a column called `actor` containing the URI of each trusted actor. The following query returns n-tuples created by trusted actors:

```
SELECT * FROM ntuples
WHERE actor IN (SELECT actor FROM trusted_actors);
```

Retrieving all n-tuples that were added and not yet deleted at any given point in time can be accomplished with a straightforward query (in this example, the point in time is represented as ?t):

```
SELECT * FROM ntuples
WHERE start_time <= ?t
AND (end_time IS NULL OR end_time > ?t);
```

**Open Archives implementation.** The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is a popular harvesting framework. This section outlines how the n-tuple abstraction can be adapted to OAI-PMH's model of a metadata collection.

OAI-PMH requires that a service expose one or more *sets* of *records*. A record is a time-stamped, identified item that can be retrieved in one or more *metadata formats*. It can also be deleted, and an OAI-PMH service is required to report that fact in response to queries about the record rather than acting as if the record never existed.

The OAI-PMH model matches the n-tuple model in several important ways. First, the n-tuple model contains information about the time at which triples were added and/or deleted. Second, sets of triples can be extracted from the n-tuple store and represented in several metadata formats, including RDF/XML and N3.

The most important *mismatch* between the n-tuple model and the OAI-PMH model is the granularity of description. In the n-tuple model, each triple is associated with attribution and timing information, but not otherwise identified. In OAI-PMH, a record is generally a *collection* of statements (e.g., a set of Dublin Core fields) associated with a single identifier and set of timing information. A closer match in the n-tuple model to OAI-PMH's concept of a record is the subject. The set of all non-deleted triples on a subject can be considered an OAI-PMH record, whose identifier is the subject URI and whose timestamp is the most recent add or delete time from among the set of all triples, deleted or not, on the subject.

In SQL, the following query will retrieve all the non-deleted triples on a given subject (where the subject is ?s):

```
SELECT subject, predicate, object FROM ntuples
WHERE subject = ?s AND end_time IS NULL;
```

This is based on the simplifying assumption that the n-tuple store does not allow an actor to delete a triple at some specific time in the future; that case could be handled by adding the clause `OR end_time > CURRENT_TIMESTAMP`.

Determining OAI-PMH timestamp for a record in SQL from the n-tuple table requires some temporal arithmetic, which can be accomplished with the following view:

```
CREATE VIEW oai_ts
AS SELECT subject, MAX(upd_time) upd_time
FROM
  (SELECT subject, MAX(start_time) upd_time FROM ntuples
  GROUP BY subject UNION
  SELECT subject,
    MAX(COALESCE(end_time, CURRENT_TIMESTAMP)) upd_time
  FROM ntuples GROUP BY subject)
GROUP BY subject;
```

Again, triples explicitly deleted in the future can be handled with a more complex query.

Actors in the n-tuple model make reasonable OAI-PMH sets, although a practical issue is how to map actor URI's to OAI-PMH set identifiers. A record's membership in a set corresponding to an actor can be determined by finding any n-tuple with the record's subject and the actor URI. The set of all sets corresponding to actors is also simple to compute in SQL from the n-tuple model using `SELECT DISTINCT(actor)`.

## 6. Conclusion

Harvesting heterogeneous information from multiple sources is critical to enabling collaborative e-science, and RDF provides a convenient abstraction for integrating heterogeneous information. To harvest RDF triples, it is useful to know "who said what when." Implementing this second-order information using RDF reification scales poorly. Instead, extending the representation of RDF triples to include information about attribution and timing can enable harvesting decisions without the need for a fast triple store.

In this paper, I have outlined a three-tier approach to harvesting RDF triples in which a journal is "rolled up" into an n-tuple store before being compiled into an RDF graph. The three tiers in the approach correspond to example implementations based on files, relational databases, and triple stores. The practical feasibility of the three-tier approach is demonstrated by harmonizing it with OAI-PMH, a standard protocol for metadata harvesting. The resulting approach supports reasoning in a dynamic, loosely-coupled, collaborative environment.

This strategy is currently in use as part of the CLEANER / CUAHSI CyberCollaboratory project at NCSA, a collaboration, data management, and workflow portal for environmental scientists and engineers. We use a logging API to capture user actions from workflow execution as well as asynchronous and synchronous collaboration and use the harvested triples to perform social network analysis and provide customized recommendations to users to help guide them through complex sets of resources and tools. For more information see <http://cleaner.ncsa.uiuc.edu/home/>.

## References

1. Wong, S. C., Miles, S., Fang, W., Groth, P., and Moreau, L. *Provenance-based validation of e-science experiments*. In Proceedings of 4th International Semantic Web Conference (ISWC'05), volume 3729 of Lecture Notes in Computer Science, pages 801-815, Galway, Ireland, November 2005. Springer-Verlag.
2. Lagoze, C. and de Sompel, H. V. 2001. The Open Archives Initiative: Building a low-barrier interoperability framework. <http://www.cs.cornell.edu/lagoze/papers/oai-jcdl.pdf>. <http://citeseer.ist.psu.edu/lagoze01open.htm>
3. Heymans, S., Nieuwenborgh, D.V., Vermeir, D. "Preferential Reasoning on a Web of Trust." Fourth International Semantic Web Conference, Galway, Ireland, 2005.
4. Huang, Z., and Stuckenschmidt, H. "Reasoning with Multi-Version Ontologies: a Temporal Logic Approach." Fourth International Semantic Web Conference, Galway, Ireland, 2005.
5. "Network Time Protocol," IETF RFC 958.
6. "RDF Semantics." W3C Recommendation, 10 February 2004. <http://www.w3.org/TR/rdf-mt/#Reif>