Representing and Querying Historical Information in RDF with Application to E-Discovery

Brian McBride and Mark Butler

HP Laboratories
HPL-2009-261

Keyword(s):
semantic web, temporal RDF, temporal query, temporal modeling, temporal application

Abstract:
Some semantic web applications must represent a domain as it changes over time. This raises two questions: how do we describe a changing domain using the Resource Description Framework (RDF) and how do we query that representation? Here we review several techniques for modeling change over time and querying those models and suggest a new novel technique. We illustrate this discussion with a real world example application which maintains a historical knowledge base containing a summary of information about employees, organizational structure, business activities and products, which enables users to rapidly identify employees who may hold information relevant to an e-discovery request.
Representing and Querying Historical Information in RDF with Application to E-Discovery

Brian McBride and Mark Butler
Hewlett Packard Laboratories, Filton Road, Stoke Gifford, Bristol BS34 8QZ, UK
{brian.mcbride@hp.com, mark.butler2@hp.com}

Abstract. Some semantic web applications must represent a domain as it changes over time. This raises two questions: how do we describe a changing domain using the Resource Description Framework (RDF) and how do we query that representation? Here we review and several techniques for modeling change over time and querying those models and suggest a new novel technique. We illustrate this discussion with a real world example application which maintains a historical knowledge base containing a summary of information about employees, organizational structure, business activities and products, which enables users to rapidly identify employees who may hold information relevant to an e-discovery request.

Keywords: semantic web, temporal RDF, temporal query, temporal modeling, temporal application.

1 Introduction

RDF graphs often present a static representation of the domain they model; that is they do not model how the domain changes over time. A static model may represent only eternal information, that is information that does not vary with time such as the identity of the authors of this paper. Alternatively, a static model may represent the state of a domain at a point in time such as ‘now’. Such static models are sufficient for some applications, but not for others. We describe an application to support the legal process of e-discovery that requires a data model that represents the state of a domain as it changes over decades. We also describe a technique for representing a changing domain in a single RDF graph.

E-Discovery is the process of collecting, preparing, reviewing and producing evidence in the form of electronically stored information (ESI) during litigation [1]. The e-discovery process is implemented by lawyers and paralegals who need to find all the systems and people in possession of all the ESI relevant to a legal matter. Legal matters often involve events that occurred several years in the past. The lawyers and paralegals must find the people who were involved in those past events. We have developed a prototype of an application to help find these people. This application is built on an information store containing historical information about organizations, activities, people and the roles they performed.
Our information store represents information as an RDF graph. We use the term *temporal RDF graphs* to refer to RDF graphs that represent the state of a domain as it changes over time. We discuss several proposals from the literature for how to structure temporal RDF graphs and describe a novel technique used in our store. Our technique has the advantages that it is more compact than some others and is able to model a changing domain within a single RDF graph. We describe how queries of our temporal RDF graphs can be represented in SPARQL and suggest a syntactic extension to SPARQL that allows such queries to be more conveniently expressed.

2 Background

A changing domain can be represented either as events or states [2]. A *state* describes the domain at a particular time or period of time, whereas an *event* is a change in the state of a domain. A state oriented temporal data model describes the states of a domain over time and could use *versioning*. On the other hand, an event oriented temporal data model describes changes in the state of a domain. Two different times are commonly associated with a change in a domain. The time when a change occurred in the domain or when the domain had a particular state is known as *valid time*. The time when a change occurred in a model of the domain or when that model was in a particular state is known as *transaction time* [3]. In a state oriented model, identical states or parts of states with overlapping or adjacent valid intervals can be coalesced into a single state with a single valid interval. A data model has undergone *maximal interval normalization* if it is not possible to coalesce any of its states.

Another key distinction between approaches to constructing temporal data models lies between endurantist approaches and perdurantist approaches [4]. In an *endurantist* approach a fundamental distinction is made between objects such as people that are *endurants* and events or processes such as a rain shower that are *occurrants*. Endurants have properties that change with time. In a *perdurantist* approach, there is no fundamental distinction between endurants and occurrants. Instead, all objects are regarded as having temporal parts i.e. as being four dimensional objects with a time dimension. Each temporal part is a separate object existing at a moment in time with its own fixed properties.

3 Related Work

Several different approaches for representing time in RDF have been proposed. These can be divided up into three broad classes [5]: *versioning* approaches, shown in Figure 1, maintain distinct RDF graphs that represent the state of the domain at different points in time, *abstract syntax extension* approaches, shown in Figure 2, extend the RDF abstract syntax to incorporate notions of time and define corresponding semantics, while *conventional RDF* approaches, shown in Figure 3 work within the current RDF abstract syntax and semantics and model the changing
state of the domain explicitly as triples in the RDF graph, perhaps using reification or N-ary relations [6].

**Fig. 1.** Using graph versioning to represent properties that vary over time. Here temperature is 10 in the period $t_0$ to $t_1$ and 12 in the period $t_2$ to $t_3$.

**Fig. 2.** Extending the RDF abstract model by allowing annotations of properties to model the data in Figure 1.

**Fig. 3.** Using blank nodes based on the N-ary relation proposal to represent properties that vary over time, to model the data in Figure 1.

Reviewing the literature, we have identified a number of previous investigations into representing temporal information in RDF, specifically Temporal RDF, Multidimensional RDF, Annotated RDF, Time-OWL, OWL Fluents and named graphs.
Temporal RDF (tRDF) [5] extends the RDF abstract syntax to associate with each triple a temporal label which represents the time, i.e. the instant, interval, initial time or final time at which the triple is. tRDF specifies new syntax and semantics for temporal RDF graphs, proposes a way to represent temporal RDF as a standard RDF graph and specifies rules to implement entailment on tRDF graphs represented as RDF.

Multidimensional RDF (MRDF) [7] extends the RDF abstract model so that the objects of triples may have values that are context dependent. Contexts form an n-dimensional space, so a context is specified by defining its coordinates in that space, one of which may be time. In contrast to other approaches, only the object of the triple is labeled with the context, not the whole triple.

Annotated RDF (aRDF) [8] attaches an annotation to the property of a triple, such as a time interval, a probability or indicator of the provenance associated with the triple. The authors do not specify a particular representation but suggest this could be done either using a quad-based data model or by using reification.

Time-OWL [9] is an OWL vocabulary for describing instants, intervals and timezones in OWL, but does not propose a standard way of using these constructs to represent valid times or transaction times in RDF models.

Welty et al [4] review several approaches to representing time explicitly in OWL before proposing an approach of their own. They characterize the problem as one of representing fluents, i.e. relations that hold within a certain time interval and not others, within a representation like RDF which is limited to binary relations. They consider using reification, but outline a number of known problems with this approach: first, it leads to a proliferation of objects and triples, because a new object is created for each tuple, and an additional triple for every element of the tuple. Second, it is possible to create multiple objects that reify the same tuple. Third, because the triples are represented in a different structure, the use of OWL reasoning becomes more difficult. They describe their solution as being a four dimensional ontology based on the perdurantist viewpoint outlined earlier, where entities are four dimensional objects. However, they note this approach also leads to proliferation of objects.

Tappolet et al [10] propose an approach using named graphs. Each named graph has an associated time interval and contains triples that are valid for that time interval. An additional graph is used as a catalog of the named graphs and the interval associated with each. This approach reduces the number of triples required over some other approaches. The normal treatment of blank nodes (b-nodes) is problematic in this approach. B-nodes are normally treated as existentially qualified variables where the scope of quantification is the graph in which they are contained. This approach puts statements about the same b-node for different time intervals into different named graphs and thus requires a scope of quantification over multiple graphs. To avoid this problem, b-nodes are replaced by URIs thus allowing the use of off the shelf software for processing RDF.

RDF has some similarities with earlier semi-structured data models [11, 12] such as OEM. There has also been work on extending these to representat temporal information, for example by annotating the graph with tags indicating when a node was created or updated, and when an arc is added or removed [13], or annotating nodes and arcs with a transaction time or valid time [14, 15].
4 Our Approach

For our application we wanted an approach for representing valid time that complies with the existing RDF abstract model and semantics and could be implemented using Jena [16]. We preferred an endurantist approach for two reasons. First, we believe this approach fits better with the way most people think. Second, our application was being developed as part of a larger program of work which was using an upper ontology derived from Dolce [17] and used an endurantist approach. We considered the use of reification or N-ary relations, as shown in Figure 3, but rejected them because of the number of extra triples generated and the way this approach changes the structure of the domain model. We also noted that because two reified triples with the same subject, predicate, and object are not necessarily identical [11], this could potentially make coalescing adjacent identical statements more difficult. Instead, we decided to adopt the approach shown in Figure 2, i.e. property annotation, which is similar to the approaches in [9, 14, 15]. Our chief innovation is the way we encode this in RDF, avoiding the need to extend the abstract syntax or model theory. We do this by creating a new property for each time interval for which a base property is asserted. For example, we represent the assertion that an employee was a member of HPLabs from the 27th November 2000 to the 1st October 2009 as follows:

```
```

The notation `f:memberOf:(2000-11-27--2009-10-01)` is shorthand for a URI that identifies a temporal property. A temporal property has a base property, in this example `f:memberOf`, that is asserted for a time interval, in this example [27 Nov 2000, 01 Oct 2009]. The temporal property in this example has the following properties:

```
f:memberOf:(2000-11-27--2009-10-01) a rdf:Property;
  tb:property f:memberOf;
  tb:begin "2000-11-27"^^xsd:date;
  tb:end "2009-10-01"^^xsd:date .
```

The `tb:property` triple identifies the base property of the time bounded property, the `tb:begin` property identifies the start of the interval for which it is valid and the `tb:end` property identifies the end of the interval for which it is valid. Adding these properties to the graph simplifies expressing and processing temporal queries.

We might have used a b-node rather than a URI to represent bounded temporal property had not the RDF abstract syntax prohibited b-nodes as predicates [18]. We compute a URI for a time bounded property by concatenating a fixed namespace URI, the URL encoding of the URI that identifies the base property, the lexical forms of the XML Schema datatype values that represent the beginning and end of the interval, along with some formatting characters to aid readability and parsing. The URI for the time bounded property in this example is:
We considered generating a hash for each new unique temporal property e.g. \texttt{n5ddpj} or \texttt{n5ddp6}. This would decrease the serialization length and placed less stress on RDF/XML readers and writers, but would impact human readability for debugging.

The interval of a time bounded property is closed at its beginning and open at its end so that properties may have only one value at any given time. Intervals may not be fully specified. Either or both of the start time or the end time may be omitted. To improve the performance of some queries, if a time bounded property has no specified begin time, it is given an \texttt{rdf:type} property with object \texttt{tb:OpenBeginProperty}. Similarly if it has no specified end time, it is given an \texttt{rdf:type} property of \texttt{tb:OpenEndedProperty}. Whilst queries that select resources that lack a given property can be expressed in SPARQL, it is generally faster to query for properties that are present rather than those that are absent.

The temporal representation proposed here has a number of advantages: first, it does not require new RDF abstract syntax or semantics. Second, compared with reification or an N-ary relation, it requires fewer triples, and it provides some compression when there is more than one occurrence of a particular property being valid over the same time period. We note that with the named graph approach, the graph catalog contains the start and end times once for each named graph, whereas with the temporal properties approach this information is repeated for each temporal property. The temporal properties approach will therefore require more triples, however, for practical applications we expect the different to be small. Our approach models a time varying domain within a single RDF graph, supports the use of b-nodes, allows the merge of temporal RDF graphs in the normal way and allows the use of existing tools. Jena Rules \cite{16} are currently restricted to operating on a single RDF graph and do not support expressing rules on multiple graphs. Since in our approach, a temporal graph is a single RDF graph, Jena Rules can be used to perform simple inference on temporal graphs.

## 5 Semantics

The semantics of RDF and RDF Schema are defined in \cite{11}. Here we sketch an approach to extending the semantics of RDF to account for temporal properties. A simple interpretation \( I \) of a vocabulary \( V \) is defined as:

1. A non-empty set \( IR \) of resources, called the domain or universe of \( I \).
2. A set \( IP \), called the set of properties of \( I \).
3. A mapping \( IEXT \) from \( IP \) into the powerset of \( IR \times IR \) i.e. the set of sets of pairs \( \langle x,y \rangle \) with \( x \) and \( y \) in \( IR \).
4. A mapping \( IS \) from URI references in \( V \) into \( (IR \cup IP) \).
5. A mapping \( IL \) from typed literals in \( V \) into \( IR \).
6. A distinguished subset \( LV \) of \( IR \), called the set of literal values, which contains all the plain literals in \( V \).
We define a simple temporal interpretation by adding the following to a simple interpretation:

1. A subset $T$ of $IR$ representing times.
2. A value $NT$ that represents no time.
4. A subset $BP$ of $IR$ which is the set of base properties.
5. A mapping $PT$ from $BP \times (T \cup \{NT\}) \times (T \cup \{NT\})$ into $IP$. This mapping ensures that temporal properties are properties in the simple interpretation and defines the structure that relates a temporal property to its base property and interval.

Further, we constrain a simple temporal interpretation such that for all $tp = PT(bp, t_0, t_1)$ defined by $PT$:

1. $IEXT(tb$:property$)$ contains $\langle tp, bp \rangle$.
2. If $t_0$ is unequal to $NT$, then $IEXT(tb$:begin$)$ contains $\langle tp, t_0 \rangle$ otherwise $IEXT(tb$:begin$)$ contains no pair $\langle tp, t \rangle$ for any $t$.
3. If $t_1$ is unequal to $NT$, then $IEXT(tb$:end$)$ contains $\langle tp, t_1 \rangle$ otherwise $IEXT(tb$:end$)$ contains no pair $\langle tp, t \rangle$ for any $t$.

It is clear from this definition that any simple temporal interpretation is also a simple interpretation and therefore consistent with the simple interpretations of RDF defined in the RDF semantics recommendation. We leave extending the definitions of RDF interpretations and defining temporal variants of RDFS interpretations for future work as discussed in section 8.

6 Querying Temporal RDF Graphs Using SPARQL

Our temporal RDF graphs are ordinary RDF graphs and so can be queried in the normal manner using SPARQL. However, it is helpful to the writer of temporal queries to provide some extra syntax to enable queries to be written more compactly and to hide the details of the underlying representation. Following the notation we used earlier, we describe queries using a syntax that allows predicates in triple patterns to specify a begin time and an end time. For example, to describe all the people who worked in HPLabs in 2008:

```
DESCRIBE ?labbie
WHERE {
  ?labbie f:memberOf:(2008-01-01--2009-01-01) f:HPLabs
}
```

The URI of the temporal property is fully specified and this query consists of a simple triple pattern.

More complex queries may require variables for base properties, begin and end times. For example, consider a query to find the names and current email addresses of the managers of all the engineers who committed code to a particular software module in the first half of 2008:
SELECT DISTINCT ?mngrName ?mngrEmail
WHERE {
    ?module rdfs:label "module name" .
    ?person f:hasManager:(?mBegin--?mEnd) ?mngr .
    FILTER (tb:intervalsIntersect(?uBegin, ?uEnd, ?mBegin, ?mEnd))
    ?mngr f:contactDetails:(*--) ?vc .
    ?vc vc:FN ?mngrName .
    ?vc vc:EMAIL ?mngrEmail .
}

The first triple pattern selects the appropriate module. The next triple pattern and associated filter selects the persons who updated the code module in the period in question. The next triple pattern, with the f:hasManager predicate, finds the managers of those persons, and the following filter restricts the managers to those who managed the persons in the time period during which they updated the module. The next triple pattern selects the current contact details for the managers and the last two select the manager’s current name and email address.

The function tb:intervalsIntersect is one of several provided to enable basic reasoning about intervals. The "*" in the f:contactDetails triple pattern is a don’t care symbol. It specifies that the temporal property may have, but need not have, a begin time. The lack of end time specifies that the temporal property must not have an end time.

This query can be expressed in standard SPARQL as follows:

SELECT DISTINCT ?mngrName ?mngrEmail
WHERE {
    ?module rdfs:label "module name" .
    ?hasManager tb:property f:hasManager .
    ?hasManager tb:begin ?mBegin .
    ?hasManager tb:end ?mEnd .
    FILTER (tb:intervalsIntersect(?uBegin, ?uEnd, ?mBegin, ?mEnd))
    ?person ?hasManager ?mngr .
    ?contactDetails tb:property f:contactDetails .
    ?contactDetails rdf:type tb:OpenEndedProperty .
    ?mngr ?contactDetails ?vc .
7 Use Case: The History Store

In the US, The Federal Rules of Civil Procedure place obligations on organizations to produce to a court or litigating parties evidence in the form of Electronically Stored Information. The process by which this evidence is produced is known as e-discovery.

We investigated how this process works in Hewlett Packard (HP) and learned that a key step in the process is identifying employees who were either in key roles relevant to the legal matter, or were in roles that indicate they might be in possession of relevant ESI. E-Discovery often takes place several years after the events concerning the legal matter and so we set out to develop a History Store, a system that would maintain a historical record of information that would aid the e-discovery process. Our History Store is intended to include, a historical record of the HP organizational structure, what products were developed, who filled key roles such as project manager and product manager on product developments and who designed or modified hardware or software components.

We have developed a prototype implementation of the History Store using temporal properties to represent historical information. The prototype was developed using the Jena [16] semantic web application library and TDB [19] for persistent storage of RDF graphs. We extended the Jena API with a new API to support the representation of temporally varying information. This temporal API has operations to support adding new properties to a resource at some time, changing the value of a property of a resource at some time and closing the interval for which a property of a resource had a specific value. The API is designed to be independent how temporal information is represented. Nine months history of personal and organizational data was loaded into the store along with five months history of updates to the code base of a major software product. This information was represented in a modest 14M triples.

The information in the history store was obtained from structured data sources within HP. For example, information about employees, organizational structure and reporting relationships was obtained from the corporate LDAP directory. This corporate directory maintains no historical information so on any given day it only reflects the state of the enterprise on that day. We used the LDAP directory as an information source for our prototype history store because we could easily get access to the data. Information in the directory is sourced from other applications, and as noted below, for a real deployment it may be better to source the information directly from those applications.

A dump of the LDAP directory on a given start date was ingested into the history store. All temporal properties produced by this initial ingest had intervals that were both open beginning, and open ended as neither the date on which they first became
valid nor the date on which the ceased to be valid are known. On each subsequent day, the dump of the directory for that day and the previous day were compared and the history store updated to reflect all changes. As illustrated in Fig 4 for example, if an employee’s organizational affiliation changed, the open ended temporal property of the employee that described the old relationship would be replaced by a temporal property with a defined end time, and the new organizational affiliation relationship would be represented by adding a new temporal property of the employee with a defined begin time, but no defined end time.

Fig. 4. Updating a temporal model to represent a change in organizational affiliation. The history store uses the VCard standard [20] to represent employee names and contact details and an HP authored ontology, based on an upper level foundational ontology described in [21], to represent other information about the employee, such as their email, location, manager, host organization, job family etc.

The LDAP directory does not contain an explicit representation of the organizational hierarchy. However, it does associate an organization with each employee. This information, taken together with employee reporting relationships, is sufficient to construct an organization chart for the enterprise. HP uses a four level hierarchy for representing organizational structure, consisting of top level Business Groups, Business Units, Organizational Entities, and then Work Groups. Business Groups and Business Units have names and abbreviations in the LDAP directory, whereas Organizational Entities and Work Groups just have names. We created unique URLs for Business Groups and Units from their abbreviations, and for Organizational Entities and Workgroups from their names.

Because we are generating the organizational hierarchy from employee records, some complexities arise. For example, if the name of an employee’s Organizational Entity changes, does this mean that organizational entity remains the same but has changed its name, or does it mean the employee has moved to a new organization? If the Organizational Entity has changed its name, then we will see many name changes, for all the member employees so it is possible to distinguish these two cases. A practical consequence for the design of the application that ingests the LDAP data is that it cannot process each employee record independently. Some properties of an employee can be processed on an employee by employee basis, but some changes must be aggregated over all employees and then applied to the history store.
Another issue is the difference between changes and corrections. A change in the LDAP directory, such as a name change, may reflect a genuine change in an employee’s circumstances or it may be a correction to erroneous data entered previously. Unfortunately it is not possible to tell from the LDAP directory, except possibly by the use of standardization screen [22] that identifies obvious erroneous values. We have not explored the use of a standardization screen in this case. For our prototype, we regard the LDAP data as definitive. For a real deployment, where information is sourced from authoritative applications, it may be possible to distinguish changes and corrections when the data is sourced and reflect these accurately in the history store.

Another source of information for the history store was the SVN repository containing the source code for a major product. The SVN log command was used to create a summary of which users modified which software modules each month and this information was added to the history store. The SVN data, integrated with the LDAP data, allows queries such as “Who were the managers responsible for the Query module in January 2009?” and “Are there any managers that also commit code changes?” to be expressed in SPARQL and readily answered. We believe that the integration of information from multiple structured or semi-structured information sources in the enterprise will be a powerful tool for supporting e-discovery.

An issue that arose integrating the SVN data was identifying the employees in the LDAP directory corresponding to the SVN login ids returned by the SVN log command. It was possible however, to map all but one of the SVN login ids to an employee id using a number of simple heuristics. For example, many users’ SVN login ids were the same as their HP email address.

Although, the heuristics were not perfect, if the history store can enable an e-discovery paralegal to find, in 5 minutes, a useful number of the engineers who worked on a code module five years in the past, whether they still work for HP and if so how they can be contacted, that is a considerable improvement over what they have today. Whilst the information in the history store is itself discoverable, it is not intended to be part of the chain of evidence but merely a tool for helping to identify evidence.

An RDF knowledge base is not the only way to build a history store. An alternative would be to use a relational database with extra columns to record the interval for which the row is valid. There are however key advantages to using RDF over a relational database. First, a history store, by its nature, is intended to be around for a long time. There will be many changes over that time, not just in the properties and relations of resources, but in the conceptual model or ontology that defines those properties and relations. Since the history store has no schema that defines its storage layout, it has greater flexibility to cope with such changes.

Secondly, using RDF means it is possible to represent more complex time patterns [23] than in a relational database. Specifically, if only a single property describing an employee changes, for example the manager of an employee, in the relational database approach it would be necessary to duplicate the whole row containing the manager column whereas in the RDF solution it is only necessary to update one property.
8 Conclusions and Future Work

Many potential semantic web applications need to model the state of a domain as it changes over time. There are several different techniques for representing the changing state of a domain in RDF, and they have different strengths and weaknesses. We have proposed an approach for representing temporal information within a single RDF graph using temporal properties and illustrated how such graphs can be queried.

We have also described the History Store, an application to support the process of e-discovery that relies on the ability to represent change over time. We have illustrated how integrating information from multiple structured sources into the history store can enable users to rapidly answer questions that will take days or weeks to answer using current methods.

We anticipate three streams of future work on temporal RDF, engineering, modeling and formal semantics. There is engineering work to be done to optimize the performance of our implementation. Our prototype system was implemented using the off the shelf implementation of TDB [19]. We anticipate extending TDB to use optimized structures for representing temporal properties. There is also considerable redundancy in the way the current prototype represents the summaries of SVN updates and thus scope for optimizing this also.

An important issue for further exploration is the interaction between temporal properties and the RDF Schema language and the OWL ontology language. RDFS [24] defines the concepts of class, domain and range constraints, and subclass and subproperty relations. How are these affected when time is introduced into the conceptual model? Is a class a temporal object, i.e. can its instances vary with time or are classes necessarily atemporal with instances fixed for all time? How do domain and range constraints interact with temporal properties? If classes are temporal objects whose extension can vary with time, then a temporal domain constraint may state that if a resource R has a temporal property P with interval I, then R is an instance of some class for interval I. An eternal domain constraint may state if a resource R has a temporal property P with interval I, then R is an instance of some class C without temporal restriction. For example, consider the base property hpEmployeeId. A resource with which has an hpEmployeeId property for some interval I may be instance of the class HPEmployee for the same interval I, but also an instance of the class Person without temporal restriction.

References