Resource-Level Versioning in Administrative Geography RDF Data

Alex Lohfink, Duncan McPhee

University of Glamorgan, faculty of Advanced Technology, Dept. of Computing and Maths, Pontypridd, CF37 1DL
Tel. +44443 482950
(alohfink, dmcphee)@glam.ac.uk

ABSTRACT

The Resource Description Framework (RDF) is a base technology of the semantic web, providing a web infrastructure that links distributed resources with semantically meaningful relationships at the data level. An issue with RDF data is that resources and their properties are subject, as with all data, to evolution through change, and this can lead to linked resources and their properties being either removed or outdated. In this paper we describe a simple model to address such an issue in administrative geography RDF data using RDF containers. The model version-enables such data at the resource level without disturbing the inherent simplicity of the RDF graph, and this translates to the querying of data, which retrieves version-specific properties by matching simple graph-patterns. Further, both information and non-information resources can be versioned by utilizing the model, and its inherent simplicity means that it could be implemented easily by data consumers and publishers. The model is evaluated using some UK administrative geography RDF datasets.

Keywords: Key words: administrative geography; semantic web; versioning; RDF; linked data

1. Introduction

The publication of data in Linked Data (RDF) form has increased significantly recently and in the UK this process has been accelerated through the Government's 'Making Public Data Public' initiative, which is encouraging Government Agencies to publish data in linked form. However, no real solution currently exists to enable data to be versioned other than at dataset level, and being able to version at the resource level is likely to provide significant advantages to both publishers and consumers of Linked Data by facilitating access to different versions of specific data.

The Ordnance Survey[1](OS) have published its UK administrative geography data in RDF format, and an issue that has arisen with this data is that the administrative units represented change frequently (boundaries, for example, are released twice a year), but these changes are not adequately represented in the data as there is no logical organisation between different versions of units. Because of this, different users of the administrative geography datasets could link to different versions of administrative units represented in whichever version of the data they are accessing, meaning that inconsistencies will be apparent between different RDF datasets. There is also a requirement for applications to link to other versions of a resource that may not necessarily be the latest version. In this paper we describe a mechanism that could solve this problem by version-enabling RDF administrative geography data at the resource level. This has been in the form of a practical and simple Linked Data model that can potentially provide solutions to the issues described here. The model uses RDF container and collection elements to represent versioned resources within a RDF graph, representing collections of resources as single entities. The model does not interfere with the inherent simplicity of the graph-based structure imposed by the RDF data model, and this simplicity is evident when formulating queries about versioned resources, where standard SPARQL (W3C 2008) queries can be used to retrieve versioned data by matching simple graph patterns. We believe the model offers the potential to provide a practical and scalable solution to versioning in Linked Data datasets whilst retaining sufficient simplicity to be implemented by data publishers and consumers. It also has the potential to be extended to incorporate other issues in resource-level versioning such as ambiguity (where two or more resources share the same name) and variance (different representations of the same resource).

The rest of this paper is organised as follows: in the next section we describe the problem which our model addresses and previous work, in section 3 we describe our model, and in section 4 we describe an implementation and evaluation of versioned datasets based on our. Finally, in section 5 we present our concluding remarks.

2. Previous work

Versioning for RDF can be viewed from two perspectives: web ontology versioning (classes) and instance versioning. Instance versioning can be further divided into model-based, statement-based (triple), or resource-level versioning. Model-based versioning applies to a group of triples that form part of a logical unit. Statement-based versioning applies to individual statements (triples), while resource level (or datum-level) versioning applies to the versioning of individual resources within an RDF graph. It is within the field of resource-level versioning that this paper is primarily concerned.

[1] This research is sponsored by Ordnance Survey

Previous work in the field of versioning in Linked Data has been centred mainly on web ontology versioning (versioning of classes), and on the managing of change in RDF graphs, and to a lesser degree on resource-level versioning. The SemVersion (Volkel 2005) model focuses on managing change in ontologies where users can suggest different classes to include in the ontology. SemVersion can manage such changes and reconcile them into a new version of the ontology. SemVersion employs model-based versioning. Delta (Berners-Lee and Connolly 2001) is a system designed to identify differences between RDF graphs, and uses functions to compute these differences. Differences between graphs are produced in the form of a delta which represents the changes only. This means that a delta derived from a knowledge base can be applied to a subset of this knowledge base and update it, with accurate results. RDF Difference Models (deVos 2002) also uses groups of triples to describe changes that have occurred between two graphs, but goes further in specifying "forward difference statements" and "reverse difference statements", and adds preconditions that must be present in both the current graph and the difference statements before the desired version can be derived. This can be used as a type of version management or locking. RDFSync (Tummarello, Morbidoni et al. 2007) provides an algorithm to synchronise locally held RDF graphs with updates to linked RDF datasets across a network, but although updates are preserved there is no access to previous versions. Evolution patterns are used by (Soren Auer and Herre 2005) to represent changes in ontologies. Here types of changes that can occur in a RDF graph are defined and applied to ontology evolution by the specification of graph patterns representing the change. Graph patterns are graphs where variables appear in place of URIs. This concept is taken further in the EvoPat system (Christoph RieB, Norman Heino et al.) where evolution patterns are applied as a form of 'software refactoring'. Evolution patterns can be basic, accommodating atomic change, or compound, where atomic changes or other compound evolution patterns are sequenced to represent complex changes in ontology. A basic evolution pattern is composed of a SPARQL select query and a SPARQL update query (this uses a modified SPARQL query processor). These are applied according to a library of pre-defined patterns. Another version model described by (Ludwig, Kuster et al. 2008) uses an extension to the Topic Maps data model (ISO 2008) to potentially implement versions in RDF. Topic Maps represent topics (or subjects), attributes, and associations as an entity-relationship model. This model uses a structure called the VersionInfo Object, or VIO, to record start and end dates for a specific version of a topic map object. The model is stated as being applicable to RDF triples by grouping triples into logical units and linking them to a VIO, although no example or evaluation of this technique is specified. Changesets (Talis 2011) is a resource-centric approach, and uses RDF reification to describe changes to a resource. A Changeset applies to a named resource, and describes triples that are added and removed in the updated resource. The Changeset can also describe the reason for the change. This is a powerful representation but is complex, requiring a large number of triples to represent a single change, and would be correspondingly difficult to query. The UK Government data developers (Tennison 2010; HMGovernment 2011) use named graphs to record a set of changes to a group of resources. (Named graphs are groups of triples that share a common identifier). Each named graph links to what it replaces. A set of changes is therefore a

set of named graphs, and a dataset is the result of an RDF merge of the individual graphs. HTTP content negotiation has been described by (Sompel, Sanderson et al. 2010) as a method to represent versioned resources, where the default linkage is always the current version. Previous versions are timestamped and accessed by specifying a time in the HTTP request, using a timegate, which supports date-time content negotiation.

Delta, SemVersion, and RDF Difference models are aimed at managing change to web ontologies or managing and reconciling differences between RDF graphs, rather than addressing the need to be able to reference different versions of the same statements or resources within the same RDF dataset. RDFSync is able to handle updates to RDF graphs and synchronise locally help graphs across a network, but provides no capability to store or retrieve previous versions of data. The evolution patterns approach is again aimed specifically at ontology evolution and attempts to use patterns to interpret structure and therefore define change. However, this contradicts the inherent, semi-structured nature of the RDF data model, and relies on mathematical definitions of change. The implementation of evolution patterns in the EvoPat system provides a practical implementation of evolution patterns and claims to be relatively simple, but it relies on a modified SPARQL query engine and a pre-defined catalog of patterns, which could easily be problematical if this was not applicable to a particular domain. The model described by (Ludwig, Kuster et al. 2008) attempts to provide a method for versioning resources, by linking logical units of triples to VIOs. In this case, the suggestion is that the VIO would contain start and end dates relevant to the statements in the referenced logical unit, in effect extending the graph to incorporate versioning objects. However, no implementation is specified, and it is not clear how alternatives (that is versions where the representations are not necessarily time dependent) would be handled. It also organises VIOs according to a sequence, the organisation of which is not specified. ChangeSets focus on versioning at the resource level, but achieves this by recording different graph configurations representing the resource's properties for a given version, and does not provide direct access or linkage to previous versions of a resource. Also the technique of this approach and that of RDF Difference Models, that of using collections of triples to represent changes in particular graphs, is verbose, leading to more statements being added to the graph than the triples being described. The named graphs technique used by UK Government data developers versions at the graph level, and even at the dataset level, and has no capability to handle resource-level versions. This means that current RDF graphs about a particular study area are derived by merges, and direct linkage to previous/other versions is not possible. The model described by (Sompel, Sanderson et al. 2010) versions at the resource level, and has the advantage that the default URI is always the latest version. However, this would be problematical if linkage was required to a non-default version. Also, this model only distinguishes versions using date-time values, and would therefore not handle versions not distinguished by time properties. To our knowledge no model can adequately represent versions at the resource level, and provide direct linkage or access to versions other than the latest version, whilst retaining the inherent simplicity of the RDF data model.

3. Versioning administrative geographic data

The data used of the basis of our model were OS UK Administrative Geography data containing information resources describing administrative units (such as boroughs, regions, and districts) and their associated properties (such as areas, identification numbers, and topological relationships to other administrative units). Consequently, our model aimed to be applicable to these data and also fulfill our objectives in versioning resources. These objectives were as follows:

- The data should provide easy access for consumers who are only interested in the latest or most recent version of a resource.
- Previous versions of resources can be easily accessed and queried using standard SPARQL queries.
- Non-information resources can be incorporated into the model if required and similarly accessed and queried

To achieve these objectives we intended to devise an appropriate model using existing rdf elements defined by (W3C 2004) and based on the example dataset. Our sample data contained two versions of a RDF dataset, the most recent data containing resources with newly minted URIs.

There are several mechanisms and features within RDF and RDF Schema (RDFS (W3C 2004), the specification of the classes and properties of RDF) that offer the potential to model resource-level versions in administrative geography data, and provide the features of linkage to default or alternative versions that may or may not be time dependant. The following were considered.

Inferencing

RDFS allows inferencing based on defined properties such as *subClassOf* and *subPropertyOf* . At the simplest level, this provides a mechanism to create a version hierarchy based on inheritance, where new versions of an item are defined using the *subClassOf* (or 'is_a') relationship. Inferencing allows RDF to infer from the *subClassOf* relationship that the resource is a member of the superclass. For example, In Ordnance Survey Administrative Geography data, a Civil Parish is defined as a *subClassOf* a Civil Administrative Area. It can thus be inferred that the Civil Parish of Chelmsley Wood is also a Civil Administrative Area. This feature provides type propagation, and could be used to define a version hierarchy of RDFS classes.

It is also possible with some RDF implementation environments to define inferencing rules. This kind of inferencing goes beyond the scope of the RDFS inferencing capabilities, and allows the definition of specific, text-based rules by which implicit relationships can be inferred. This could allow version-specific rules to enhance a version hierarchy, such as *version_of, derivative_of, alternative_to* based on criteria derived from the differences between versions of an administrative unit.

Named graphs

Named graphs allow groups of triples to be identified as belonging to a specified graph within a larger RDF graph, and as described in the previous section, have been employed by (HMGovernment 2011) to version at the graph and dataset level. This is achieved by tagging the triples with an identifier that specifies the named graph to which it is associated, in effect making the triples

"quads". This means that a group of triples could be coupled together as a named version graph. The RDF query language, called the SPARQL protocol and RDF query language (SPARQL (W3C 2008)), has a *FROM NAMED* clause which can query named graphs.

RDF containers and collections

RDF containers and collection elements provide the capability to represent collections of resources as single entities, and link to them forming new triples. This means that it is possible to model relationships between multiple versions of data by defining appropriate RDF container or collection classes. An RDF container simply uses a blank node from which to link resources that belong to that container. Of particular interest here is the *rdf:Alt* container, which is used to describe a list of alternative values of a resource.

It is our contention that of these three mechanisms, RDF containers and collections provide the most appropriate structures to represent versions. RDFS inferencing provides a simple mechanism to deduce versions, but does not provide any version-specific relationships between versioned resources. Rule-based inferencing, on the other hand, is mainly aimed at getting more meaning from existing relationships between resources, and would require the definition of specific conditions upon which inferences could be made, which would not be expressed within triples. Named graphs disturb the inherent simplicity of the triple by tagging each one with an identifier to identify it with a particular graph, and would also require some kind of logical naming convention to facilitate querying. In addition, relationships between versions within the named graph would still need to be defined, negating the need to name the version graph. RDF containers and collections, on the other hand, provide a simple mechanism that integrates seamlessly into an RDF graph without disturbing its simplicity, and version-specific properties can be defined within this structure. Containers and collections also provide flexibility in the mode of the representation. For example, containers can be defined as rdf:Alt, rdf:Bag, or rdf:Sequence. Rdf:Alt denotes that the contained resources are alternatives, rdf:Bag denotes that there is no significance to the order in which the contained resources are represented, and rdf:Sequence denotes that the contained resources are set in sequential order. (It should be noted that there is no implicit behaviour associated with any of these containers, and that these conventions exist to provide consistency in their use.) Containers and collections may also be nested. Further, the simplicity offered by the use of containers and collections should be reflected in the queries used in retrieving version-specific properties. Also, this simplicity could offer the possibility of take-up by data consumers as its implementation should be less complex than the methods discussed earlier.

Information and non-information resources

The type of resource being versioned, specifically whether it is an information resource or a non-information resource, has an impact on how the resource is updated. For information resources, a new URI can be minted for the new version, preserving the URI of the replaced version, which can still be linked to. Alternatively the resource can be updated and keep its existing URI, the previous version being lost (destructive update). Non-information resources usually imply a finer granularity in the extent of the change ((Tennison 2010) provides the

example of school statistics where class sizes, and staff change frequently) and so require special provision. (Tennison 2010) also concludes that timestamping individual property versions and holding them in a multi-value element becomes prohibitively complex. We, therefore, have taken a broader approach to modeling the versioning of both information and non-information resources to negate this complexity, with some notable compromises. More specifically, our approach does not explicitly distinguish between information resources and non-information resources. Rather, it distinguished between versions where the new version has a newly minted URI, and versions where the new version retains the existing URI.

Versioning resources where the new version has a newly minted URI

The datasets used provided newly minted URIs for the updated resources. According to our criteria for versioning, the model must provide easy linkage or access to the latest version of the resource. In this case we used an rdf:Alt container to represent versions of a resource, and used Dublin Core metadata (DCMI 2010) version properties to establish version- specific properties. The use of the rdf:Alt container to represent versioned information resources within an RDF graph is shown in Figure 1. The model shown represents two versions of a resource described in two separate RDF datasets produced by the OS. The diagram uses the following prefixes:

dcterms: *http://purl.org/dc/terms/*
admingeo: *http://www.ordnancesurvey.co.uk/ontology/AdministrativeGeography/v2.0/AdministrativeGeography.rdf#*
rdfs: *http://www.w3.org/2000/01/rdf-schema#*
rdf: *http://www.w3.org/1999/02/22-rdf-syntax-ns#*

The versioned resource has the URI *http://data.ordnancesurvey.co.uk/id/7000000000010769*, and the property rdfs:label with the value of *The London Borough of Bexley*. The property *dcterms:hasVersion* links the versioned resource to the rdf:Alt container, specifically a blank node defined as type *rdf:Alt*. The property *rdf:_1* of this container specifies the alternate version for the *The London Borough of Bexley* resource, which has the URI *admingeo:osr7000000000010759*. By the convention defined by (W3C 2004), all container members are identified by the properties rdf:_1, rdf:_2, and so on, but the significance of the ordering is defined by the container type, as specified in the previous section. Further versions would be represented using such properties. The property *dcterms:isVersionOf* gives the reverse property, showing which resource this resource is a version of. The versioned resource is given the property *dcterms:isPartOf* to show which administrative authority this version was/is a part of. In this example, it denotes that this version of *The London Borough of Bexley* is part of the Greater London authority which has the URI *admingeo:osr7000000000041441*.
Although in the example OS data used in this work the alternative version of *The London Borough of Bexley* is part of the same Greater London authority, the model allows for a version to be part of a different administrative unit.

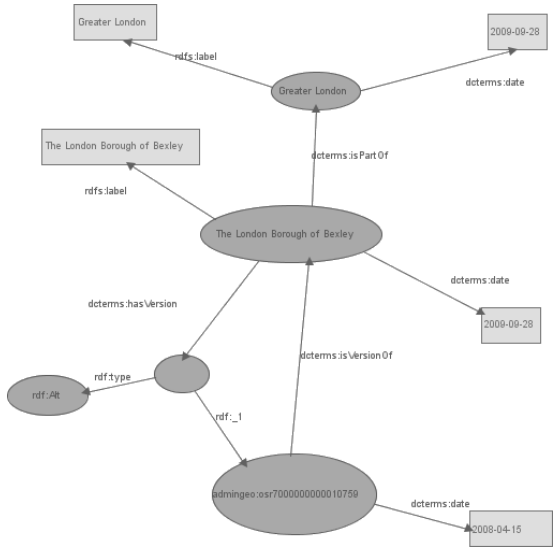

Figure 1RDF graph showing the use of the rdf:Alt container to represent two versions of the resource called 'The London Borough of Bexley'

It can be seen that date values have been linked to each administrative unit resource using the property dcterms:date. There is currently little consistency or agreement on the definition of date values that are included in OS administrative data. The values provided here represent the dates on which the datasets were first assembled. Boundary lines for administrative units are typically released twice a year, although it can be seen that there is seventeen months between our two datasets. This is in part due to the absence of a suitable mechanism to represent versions.

Versioning resources where the new version retains the existing URI

Although not present in our example data, new versions which retain the URI of the resource can be represented by the use of nested rdf:Collection elements. Figure 2 shows such an example. The same prefixes are used as in the previous example. The figure shows of an rdf:Alt container being used as the dcterms:hasVersion property of the resource (admingeo:7000000000010759), with a nested rdf:Collection element as its first (in this case only) property representing a multi-value collection of properties and their values.

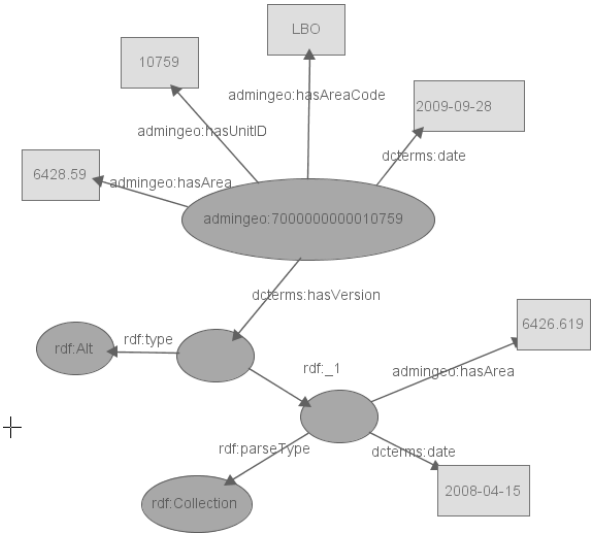

Figure 2 RDF graph showing the use of the rdf:Alt container with nested rdf:Collection element to represent versioned properties of a resource

For clarity, only a representative subset of the properties of the resource is shown in the diagram.

The property dcterms:hasVersion links to the rdf:Alt container, which contains all the resource's previous versions of its properties. The versioned properties in this example are represented by the rdf:_1 element. It can be seen that the non-information properties dcterms:date and admingeo:has Area are different versions from the parent version. The parent properties admingeo:hasAreaCode and admingeo:hasUnitID are not present in the version as these were newly added in the parent. This highlights a restriction in the model: there is no recording of any data to

determine properties that have been added or removed, nor is there data concerning how or why the change occurred. The representation does offer, however, simple querying. An alternative model would be to represent only those properties that exhibit change in the versioned properties, but this would make querying at best far more complex (and possible impossible), although the forthcoming SPARQL 1.1 (W3C 2011) may mitigate this.

4. Implementation

To evaluate the version model we implemented triplestores containing versioned resources based on the models described in the previous section. Two such triplestores were implemented, one for resources where the new version has a newly minted URI, and one for resources where the new version retains the URI of the parent version. A record linkage algorithm was developed to generate the version-specific RDF links. The datasets to be produced were formed by merging versions from the two example datasets: the most recent dataset (referred to henceforth as the base dataset) comprised of UK administrative geography data from September 2009, and the second dataset (henceforth referred to as the update dataset) comprised of UK administrative geography data from April 2008. As stated previously, the date values refer to the dates on which the datasets were first compiled, and will be represented in the versioned resources by a dcterms:Created property.

The algorithm followed these stages:

1. Iterate over the base dataset URIs and identify resources of type *LondonBorough*
2. Iterate over the update dataset URIs and identify resources of type *LondonBorough*
3. Create the appropriate mulit-value rdf element (either rdf:Alt or rdf:Collection) for each base dataset *resource* URI that has a matching ID in the update dataset URI.
4. Update the base dataset with the newly created container.

The datasets were implemented in an AllegroGraph (Franz 2009) triple store, chosen for its strong Java and Python APIs and the strength of its documentation. It also has a triplestore browser that allows the examination and verification of implemented triplestores, which will be used to show versioned resources in the versioned datasets.

Figure 3 shows the resultant structure of a versioned information resource in the implemented dataset.

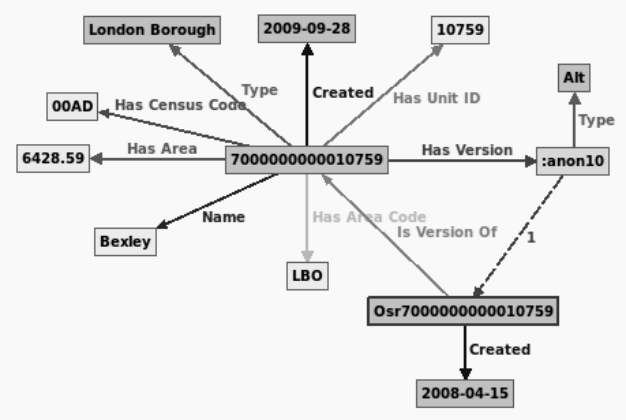

Figure 3 Screenshot of a versioned resources with distinct URIs as viewed in the triplestore browser

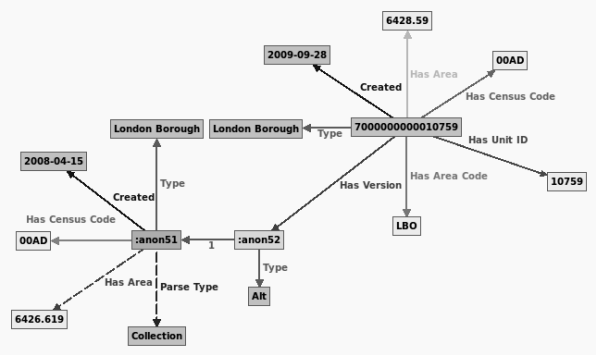

Figure 4 Screenshot of a versioned resource where the URI is retained as viewed in the triplestore browser

Although our example datasets comprised of resources that had new URIs for the different versions, as a proof of concept we implemented the two datasets as versioned resources that retained the same URI in both versions, according to our model outlined in section 3. In this case we attached the properties and their values of the version to the nested rdf:Collection element representing the rdf:_1 property of the hasVersion property of the parent version. *Figure 4* shows a versioned resource as viewed in the triplestore browser. Some of the properties have been omitted for clarity. It can be seen that the hasArea and Created properties are different than in the parent version, but the type (LondonBorough) and hasCensusCode vales are duplicated, illustrating the limitation of the model.

Potential to scale

The sample datasets used for the implementation were small by semantic web standards. To provide a guide as to how our model would scale we timed the implementation of the two triplestores and measures their sizes before and after the merges.

Table 1 shows the results we obtained. The merge times of over an hour show that the implementation would get considerably time consuming in datasets containing millions of triples, although this could be mitigated by more regular and/or phased updates. Also, the update where the new versions retain the existing URI doubles the size of the store, which might be a problem with large datasets.

Table 1 Merge times for combining the two datasets into one dataset containing versioned resources

Update type	Base size (triples)	Resources to update	Merged size (triples)	Merge time (secs)
URIs distinct in both versions	150k	11,761	207k	4017.94
URI retained by both versions	150k	11,761	325k	4043.35

Querying data in the versioned datasets

Querying the dataset demonstrates some of the proposed benefits of the model. Firstly, because the model uses standard RDF containers, the implemented data could be queried using standard SPARQL queries. Secondly, version-related properties can be retrieved along with other properties, meaning that version-specific predicates need not necessarily be defined in the query. Thirdly, because the inherent simplicity of the data structure imposed by the RDF triple has not been disturbed, version-specific attributes can be retrieved by matching simple graph patterns.

Querying the dataset in which versions have distinct URIs

For example, the query below retrieves all the properties and their values for the resource with the URI *http://data.ordnancesurvey.co.uk/id/7000000000041441* :

```
SELECT ?x ?y
WHERE
{<http://data.ordnancesurvey.co.uk/id/7000000000041441> ?x ?y};
```

In this example the variable *?x* retrieves all properties including the *hasVersion*, *isVersionOf*, and *isPartOf* properties if present. These can in turn be queried using a similarly simple query. For example, the following query retrieves the version of a resource and its parent version (prefixes are as previously defined):

```
SELECT ?x ?y
WHERE {?x dcterms:isVersionOf ?y};
```

Here the variable ?x retrieves the version, and the variable ?y retrieves the parent version. The exception is when querying the *hasVersion* property, which must access the version of the resource via the container, which is a blank node. This query is shown in the following example:

```
SELECT ?y ?z
WHERE {?y dcterms:hasVersion ?x.?x rdf:_1 ?z};
```

This query retrieves the URIs of a versioned resource, and its first version, represented by the variables *?y* and *?z* respectively. The variable *?x* represents the blank node of the container, and the graph pattern matches two triples, delineated by a period (*?y dcterms:hasVersion ?x and ?x rdf:_1 ?z)*. Replacing the *rdf:_1* value with a variable would retrieve all versions of the resource.

Querying the dataset in which new versions retain the URI

Querying this data is slightly more complicated due to the nested rdf:Collection element. The following query retrieves the first version (denoted by the rdf:_1 property) for a resource:

```
Prefix adgeo: <http://www.ordnancesurvey.co.uk/ontology/AdministrativeGeography/v2.0/AdministrativeGeography.rdf#>
SELECT ?p ?n WHERE
{ adgeo:7000000000010759 <http://purl.org/dc/terms/hasVersion> ?node_variable_1 .
 ?node_variable_1 rdf:_1 ?node_variable_2 .
 ?node_variable_2 ?p ?n . }
```

The implementation demonstrates that the model is practical and offers the potential to scale. The simplicity of the structure means that a simple pattern-based algorithm can be used to merge versions, and the simplicity evident in the queries should impact minimally on performance. It could be argued that the use of blank nodes in the rdf:Alt and rdf:Collection containers introduces a degree of vagueness or indirection into the graph as blank nodes have no URI and therefore cannot be directly linked to. It is also claimed by (Bizer, Cyganiak et al. 2008) that the use of blank nodes hinders the merging of data from different sources. (Bizer, Cyganiak et al. 2008) also discourages the use of containers (presumably because they utilize blank nodes), and suggests using multiple triples with the same predicate instead. In

the case of versions where each version has its own URI, this would mean omitting the blank node from the model and having multiple *hasVersion* links between the default version (in our case the most recent) and its versions. In the case where versions retain the URI of the parent, the rdf:Collection element's blank node would need to be retained. We argue that our model retains the semantics of the container that defines the versions as alternatives to the linked resource, and reduces complexity in that the default version will have only one *hasVersion* property. Also, the use of multi-value elements enables the implementation of nested collections, providing the mechanism to version collectively the properties of a resource.

We maintain that the versioning mechanisms described offer the facility to version resources for small to medium datasets, and retain the inherent simplicity imposed by the RDF data model, by making certain compromises. These are:

- Data redundancy, as in the versioning of resources which retain the existing URI of the previous version.
- An absence of information relating to the nature of changes to a resource.
- The use of blank nodes which may impede the data's suitability to be merged with other datasets.

5. Conclusions

In this paper we have given a brief background to RDF, and discussed some of the possible methods which may be adopted for the purposes of versioning RDF at the resource level in administrative geography data. We have described a versioning model based on RDF containers and collection elements that addresses a specific issue that has arisen in the RDF representation of administrative geographic data, that is, the requirement of a linkage to a default version of an administrative unit, and logical and easy access to previous or alternative versions of that unit. The proposed model utilises standard RDF container and collection elements in its structure, and exploits industry standard Dublin Core metadata for its version-specific properties, negating the need to mint new properties specific to versioning. The model represents versions without interfering with the simplicity of the graph structure imposed by the RDF triple, and this simplicity is evident in the queries, which are able to utilise standard SPARQL syntax and retrieve version-specific properties by matching simple graph patterns. The evaluation establishes that the model is practical and relatively simple to implement, but that scaling might be problematical in larger datasets. For this reason, we would only recommend our method for versioning small to medium size datasets with update frequencies of at least six months. The model is also potentially applicable to two issues related to versioning, that of variants and ambiguity. Variants differ from versions in that rather than being an updated representation of a resource they are a different representation of a current resource that may or may not necessarily be distinguished by time. For example a resource may have both administrative and financial representations. Ambiguity applies to resources that are physically distinct but are commonly identified by the same name. For example, there are two places in Hampshire called Hook. Someone searching for Hook on the internet would retrieve URIs for both places with no distinction between the two. Both these scenarios will be investigated in future work.

References

Berners-Lee, T. and D. Connolly. (2001). "Delta: an ontology for the distribution of differences between RDF graphs." Retrieved 3/11/2009, from http://www.w3.org/DesignIssues/Diff.

Bizer, C., R. Cyganiak, et al. (2008). "How to Publish Linked Data on the Web." from http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/.

Christoph RieB, Norman Heino, et al. "EvoPat - Pattern-Based Evolution and Refactoring of RDF Knowledge Bases." Retrieved 7th July, 2011, from http://svn.aksw.org/papers/2010/ISWC_Evolution/public.pdf.

DCMI. (2010). "The Dublin Core Metadata Initiative." Retrieved 26th April, 2010, from http://dublincore.org/.

deVos, A. (2002). "RDF Difference Models." from http://www.langdale.com.au/CIMXML/DifferenceModelsR05.pdf.

Franz. (2009). "AllegroGraph RDFStore." Retrieved 3/11/2009, from http://www.franz.com/agraph/allegrograph/.

HMGovernment. (2011). "data.gov.uk." Retrieved April 18, 2011, from http://data.gov.uk/.

ISO. (2008). "Information Technology - Topic Maps - Part 2: Data Model " Retrieved 3/11/2009, from http://www.isotopicmaps.org/sam/.

Ludwig, C., M. W. Kuster, et al. (2008). Versioning in Distributed Semantic Registries. iiWAS2008**:** 493-499.

Sompel, H. V. d., R. Sanderson, et al. (2010). An HTTP-Based Versioning Mechanism for Linked Data. LDOW2010.

Soren Auer and H. Herre. (2005). "A Versioning and Evolution Framework for RDF Knowledge Bases." Retrieved 7th July, 2011, from http://uni-leipzig.academia.edu/S%C3%B6ren Auer/Papers/242787/A_Versioning_and_Evolution_Framework_for_Rdf_Knowledge_Bases.

Talis. (2011). "Changesets." Retrieved April 25th, 2011, from http://n2.talis.com/wiki/Changesets.

Tennison, J. (2010, 27-02-2010). "Versioning (UK Government) Linked Data." Retrieved 07-07-2011, from http://www.jenitennison.com/blog/node/141.

Tummarello, G., C. Morbidoni, et al. (2007). RDFSync: efficient remote synchronization of RDF models. 6th International and 2nd Asian Semantic Web Conference

Volkel, M. (2005). "SemVersion – Versioning RDF and Ontologies." Retrieved 3/11/2009, from http://semversion.ontoware.org/kwebd233a.pdf.

W3C. (2004). "RDF Vocabulary Description Language 1.0: RDF Schema." Retrieved 3/11/2009, from http://www.w3.org/TR/rdf-schema/.

W3C. (2004). "RDF/XML Syntax Specification (Revised)." Retrieved 3/11/2009, from http://www.w3.org/TR/rdf-syntax-grammar/.

W3C. (2008). "SPARQL Query Language for RDF." Retrieved 3/11/2009, from http://www.w3.org/TR/rdf-sparql-query/.

W3C. (2011). "SPARQL 1.1 Query

Language." Retrieved August 9th, 2011, from http://www.w3.org/TR/sparql11-query/.

About the authors

Alex Lohfink is a lecturer at the University of Glamorgan, and gained a PhD there in December 2008. His current research interests are the semantic web and spatio-temporal databases.

Duncan McPhee is a senior lecturer at the University of Glamorgan. His research interests are in computer-based learning, databases, data mining, and the semantic web.