The National Archives Digital Records Infrastructure Catalogue: First Steps to Creating a Semantic Digital Archive

Rob Walpole
Devexe Limited / The National Archives
London
United Kingdom
rob.walpole@devexe.co.uk

Abstract
This paper describes the context and rationale for developing a catalogue based on Semantic Web technologies for The National Archives of the United Kingdom as part of the Digital Records Infrastructure project, currently in progress at Kew in London. It describes the original problem that had to be resolved and provides an overview of the design process and the decisions made. It will go on to summarise some of the key implementation steps and the lessons learned from this process. Finally, it will look at some of the possible future uses of the catalogue and the opportunities presented by the use of Semantic Web technologies within archives generally.

1 Background

1.1 The National Archives

The National Archives (TNA) are the official archives of the UK Government. TNA holds over 11 million historical government and public records [1] in the form of documents, files and images covering a thousand years of history. The vast majority of the documents currently held are on paper. However, as the digital revolution continues, this will soon be overtaken by a tsunami of digital files and documents for which a new and permanent home must be found that will allow controlled access for future generations.

These documents can take many forms including standard office documents, emails, images, videos and sometimes unusual items such as virtual reality models. Digital preservation brings a myriad of challenges including issues such as format recognition, software preservation and compatibility, degradation of digital media and more. Some of these issues were clearly demonstrated by the problems encountered by the BBC Domesday Project [2].

1.2 The Digital Records Infrastructure

TNA have been at the forefront of meeting this digital preservation challenge and have made great strides in finding solutions to many of the issues along with colleagues from other national archives, libraries and academia. In 2006, they deployed the Digital Repository System (DRS) which provided terabyte scale storage. Unfortunately DRS can no longer meet the vastly increased volumes of information produced by the Big Data era or the “keep everything” philosophy that cheap storage allows.

Now a new and far more extensible archive system, the Digital Records Infrastructure (DRI), is being built on the foundations of DRS to provide a quantum leap in archive capacity. This new system will allow long term controlled storage of a huge variety of documents and media. Digitised Home Guard records from the Second World War were used for the proof of concept and many more record collections, such as the Leveson Enquiry and 2012 Olympic Games (LOCOG), are now awaiting accession into the new system. At its core DRI provides its massive storage using a robot tape library. Although tapes provide highly resilient storage if treated and monitored carefully, they are not suited to frequent access. Therefore, the archive is designed to be a “dark archive”. In other words, it is powered down until an access request is received.

Although there will be frequent demands for access to the data in the archive, many of these requests can be met by substitutes from a disk cache. For example, scanned documents can be substituted with a lower quality JPEG file from disk, instead of the original JPEG 2000 held on tape. Whenever data is retrieved it will be cached on disk for the next time so that frequently requested items are always promptly available.

1.3 The DRI Catalogue

The DRI Catalogue is perhaps best described as an inventory of the items held within the repository. It is distinct from the TNA Catalogue. The latter is a comprehensive accessioning, editorial management and public access system spanning both paper and digital documents.

As the tape archive cannot easily be searched, it is vital that rich metadata is readily available to tell archivists and other users what data is being held. Some of this metadata comes from the records' providers themselves, usually a government department. Some is generated as part of the archiving process while some is obtained by inspecting or transcribing the documents. With each collection of data stored, a comprehensive metadata document is built up. A copy of this metadata is placed in the archive and another copy is sent to Discovery [3], TNA’s public access search portal, provided the record is open to the public.

Controlled release of material from the archive is of paramount importance. Although the majority of data in the archive is open to the public, some is not. This may be for reasons of national security, commercial interest or simply because it would breach someone's privacy. For example, a service and medical record is held for each member of the Home Guard. Service records are opened to the public when the soldier in question is known to be deceased. Medical records on the other hand are only released some time later, usually not until the record itself is 100 years old. Because some members of the Home Guard were very young when they served, it is possible they would still be alive today.

This crucial need to provide fine-grained control over record closure lies at the heart of the DRI Catalogue's requirements and has provided some of the key challenges during implementation, which will be discussed in more detail further on.

2 Requirements

Archiving records is only of value if those records can be accessed and viewed by researchers and members of the public. TNA's search portal for the archives is Discovery which holds over 20 million descriptions of records. Once a user has located a record of interest from searching Discovery they can request to see the original item. In cases where the record has been digitised a built-in image viewer allows the record to be viewed on-line. Until recently, the majority of records were on paper and painstaking work by the cataloguing team provided this metadata which was stored in the original electronic catalogue system (PROCAT) which has now been replaced by Discovery. In future the majority of records will come via DRI. DRI has two fundamental responsibilities with regard to Discovery which we can classify as closure and transfer which are explained in more detail below.

2.1 Closure

Until recently, most public records were closed for 30 years, however the government is now progressively reducing the closure period to 20 years. Some records, particularly those containing personal data, are closed for longer periods - up to 100 years. However the justifications for closing records for longer periods are scrutinised by a panel of academics and other experts.

Closure of records has two possible forms: the record itself can be closed but the description may be open or, alternatively, the record and the description may both be closed. In either situation it is very important that DRI does not release any closed records to the public.

Closure can apply at various levels. In one case a document may be open whereas in another only the metadata could be open. In some cases, even the metadata could be closed or possibly a whole collection of data, depending on the content and the reasons for closure.

2.2 Transfer

DRI transfers information on all of the records it holds to Discovery for public access. In the case of a closed record what the public sees depends on whether just the record, or the record and the description are closed. If the record is closed but there is a public description this will be shown, albeit with no possibility to see the actual record. In the case of a closed record and description they will be able to see that there is a record but not what the record is about. In other words, whilst providing as much public access to the records as possible, closed information must be filtered from the public view at all times.

2.3 Initial Approach

In order to establish the most effective way of dealing with the closure problem, three different approaches were prototyped simultaneously. These approaches were based on three fundamentally different models of the catalogue data. These models can be categorised as the **relational**, **hierarchical** and **graph** approach.

Relational – this approach was to retain the existing relational database management system for storing catalogue data but to rewrite the SQL queries used to establish record closure status. On the face of it this would seem to be the most obvious and expedient solution.

Graph – the second approach was to re-structure the catalogue as a graph using RDF and then query it using SPARQL. This was the least well understood approach of the three but its increasingly common usage in large scale applications suggested it was a viable solution.

Hierarchical - the third approach was to use XML to describe the DRI catalogue, store the catalogue data in a native XML database and query the data using XQuery. The nature of the catalogue is strongly hierarchical and so this seemed a good solution.

Before any of these approaches could be tested extra data needed to be added the existing DRI catalogue. It was essential that items in the catalogue knew their ancestry, or at least the item that represented their parent. To achieve this a simple Scala program was written which connected to the database, built up a map of catalogue entries and their descendants and then added a parent reference column to the main catalogue table by looking up the entry from the map.

2.4 Results

Relational

Rewriting the SQL used to extract catalogue data led to a dramatic improvement in query response times. Whereas the original query took hours to complete, the new query using the extra parent column information completed in minutes. Optimising this query may well have further reduced the query response times.

Graph

Considerable effort had to be made to create an environment where the graph approach could be tested. These steps will briefly be described here and covered in more detail later on.

1. Create a mapping from the relational database column entries to triples using D2RQ [5]
2. Export the data from the relevant relational tables into triples using D2RQ.
3. Load these new triples into a triple-store (Jena TDB [6]) which could be accessed via a SPARQL endpoint (Jena Fuseki [7]).
4. Write SPARQL queries using SPARQL 1.1 property paths [8] to establish closure status.

Once all of this was done the results of this experiment were stunning. It was possible to establish the closure status of any catalogue item based on its ancestors and descendants in seconds or split-seconds. The performance far outstripped that of the relational database queries. It was also possible to write queries that showed the ancestors and descendants of any item and verify beyond doubt that the results were correct.

Hierarchical

Testing of the hierarchical approach was abandoned; the reasons for abandoning this approach were threefold:

1. It was felt that the graph approach offered a good solution to closure problem.
2. The graph tests had led to a better understanding of this approach and, with this understanding, a number of new and interesting possibilities had arisen in terms of what could be done with the catalogue. It was felt that the hierarchical approach did not offer these same possibilities.
3. Finally, and sadly, project deadlines and cost overheads meant that, although it would have been interesting to complete the hierarchical test, the fact that a good solution had been found obliged the project to move on.

2.5 Conclusion

The issue of closure had meant that it was necessary for the DRI project team to fundamentally question the nature of the DRI catalogue. Which of the available models best represented the catalogue? While relational tables may be very good at representing tabular data such as you find in a financial institution they were found to be less suited to describing the complex relationships within the catalogue.

Because TNA accessions data from the full range of government activities it is difficult to predict what structure this data will have and what information will need to go in the catalogue. The hierarchical model offers a good solution for documents and publications but falls down when attempting to describe the poly-hierarchical structures that you find in archives. For example a scanned document may contain data about many people. How do you nest this information in a hierarchy without repeating yourself many times over?

Fundamentally the archive holds information about people, their relationships and activities over the course of time. These things are complex and varied – they are after all the nature of the world around us. The conclusion of the experiment was that the graph approach not only solved the immediate problem of closure but also most closely modelled our complex world and would in the future provide a powerful tool for discovering information held within the archive.

3 Design

3.1 Technology

The catalogue trials had provided valuable experience in a number of technologies. This included tools for working with RDF such as D2RQ and Apache Jena plus experience of new standards-based formats and languages such as Turtle [9] and SPARQL. An important factor in the technology choices made was the preference for using open source and open standards specified by the UK Government in the Government Services Design Manual:

> *"...it remains the policy of the government that, where there is no significant overall cost difference between open and non-open source products that fulfil minimum and essential capabilities, open source will be selected on the basis of its inherent flexibility."* [10]

And also for using open standards as stipulated by criteria 16:

> *"Use open standards and common Government platforms (e.g. Identity Assurance) where available"* [11]

All of these technologies met this criteria as being either open source (D2RQ, Jena) or open standards (Turtle, SPARQL).

Furthermore the trials had given developers a head start with these particular tools and there was felt there was no particular benefit to be gained by switching to an alternative solution at this stage. Having said that, the use of open standards means that, should the existing open source technology cease to meet TNA's requirements, the overhead in moving to a new tool-set should be kept to a minimum.

Another significant reason for choosing the Apache Jena framework was the excellent Java API provided. DRI is predominantly a Java based system. Java was chosen originally because the underlying technology of DRI (Tessella's Safety Deposit Box – SDB [12]) was written in Java and therefore Java was the natural choice for extending SDB functionality. The DRI development team naturally had strong Java skills and Jena's API provided a straightforward way for developers familiar with Java to start working with RDF.

3.2 The Catalogue Services

The DRI catalogue is required to provide a number of key services:

Accessioning
Firstly, it must accept new entries in the catalogue when data is initially accessioned into DRI. For each item accessioned it must provide a unique identifier which is persisted with the entry. In fact the identifiers generated must be globally unique identifiers [13].

Currently the catalogue recognises a number of item types. These are:

- **Collection** – this describes a large related group of documents, for example the Durham Home Guard records are represented by a single collection.
- **Batch** – this represents a batch of records on disk. This is how records are initially received by TNA and usually represents a substantial volume but it may or may not be a whole collection.
- **Deliverable Unit** – this represents a single item of information. It is a term coined from the paper archive world and represents something that can be handed to someone else. This may be a box or records, a folder or a single document. Similar criteria are used to for digital records.
- **Manifestation** – there are different types of manifestation. For example images have preservation and presentation manifestations. Preservation manifestations of these would be the highest quality images while presentation ones are a lower quality for display purposes.
- **File** – these are the actual digital files held within the archive.

Each of these types comes with a set of properties which must be retained in the catalogue, including things like TNA Catalogue references and closure information.

Closure Updates
The catalogue must provide the functionality to alter the closure status of a record (subject to review by an human operator).

Export
The catalogue must provide the functionality for exporting records. This is normally done in order to transfer the record to Discovery. The export process itself involves numerous steps in a controlled work-flow. The catalogue's responsibility is to allow a review of items for export and to maintain a record of the status of the export work-flow.

Lists
The catalogue must also provide the ability to persist lists of records. These are generic lists of records which may be used for a variety of purposes. Currently they are used to group records due for opening and export but there are likely to be other uses in the future.

From these requirements it can be gathered that the catalogue must provide read, write, update and delete access to the triple-store. The Apache Jena framework provides a straightforward approach to these requirements.

- **Reading** data can be done using the SPARQL Query Language [14].
- **Writing** data can be done by creating and persisting new triples
- **Updating** and **deleting** can be done using the SPARQL 1.1 Update Language [15].

There are a number of different ways of performing these tasks including:

1. Using the Jena command line utilities
2. Using the Jena API to work directly with the triple-store.
3. Using a SPARQL server such as Fuseki to perform these tasks

The SPARQL server provides an attractive solution which allows queries to be executed quickly and conveniently over HTTP as well as allowing new sets of triples to be posted to the triple-store. The Jena Fuseki SPARQL server also includes a built in version of the TDB triple-store. As TDB can only be accessed safely from within one Java Virtual Machine [16] it makes sense to use this built-in version with the SPARQL server approach. This server has a number of endpoints built in including a SPARQL query endpoint which can be accessed via a web browser. This not only provides a useful tool for developers but could in the future be exposed to staff within TNA, or even to the general public, who could then query the data for themselves given a little SPARQL knowledge.

Jena Fuseki with embedded TDB was chosen as the solution.

One hurdle to acceptance of this semantic technology stack within TNA however was the need to develop skills around semantic search and in particular in terms of learning RDF syntax and the SPARQL query language. One solution to this problem is the Linked Data API [17]. This API offers a way for SPARQL queries to be pre-configured and then accessed via RESTful URLs. For example you can configure a SPARQL query that locates a catalogue entry's descendants and then access this via a pre-set URL structure e.g.

```
      http://{server-name}/{catalogue-identitifer}/descendant
```

Elda [18] is an open source implementation of the API written in Java. The SPARQL queries within Elda are configured using the Turtle format so in this case the configuration for this specific URL would look something
like this:

```
spec:catalogueItemDescendant a apivc:ListEndpoint
      ; apivc:uriTemplate "/catalogue/{uuid}/descendant"
      ; apivc:variable [apivc:name "uuid"; apivc:type xsd:string]
      ; apivc:selector [
            apivc:where """
                  ?catalogueItem dcterms:identifier ?uuid .
                  {
                  ?item dri:parent+ ?catalogueItem .
                  }
            """
      ];
      .
```

Once this endpoint is registered (also within the configuration file) any requests that match this URI template will execute a SPARQL SELECT statement returning any matching catalogue items represented by the variable `?item`. The string value of the UUID passed in via the URI is allocated to the `?uuid` variable when the statement is executed.

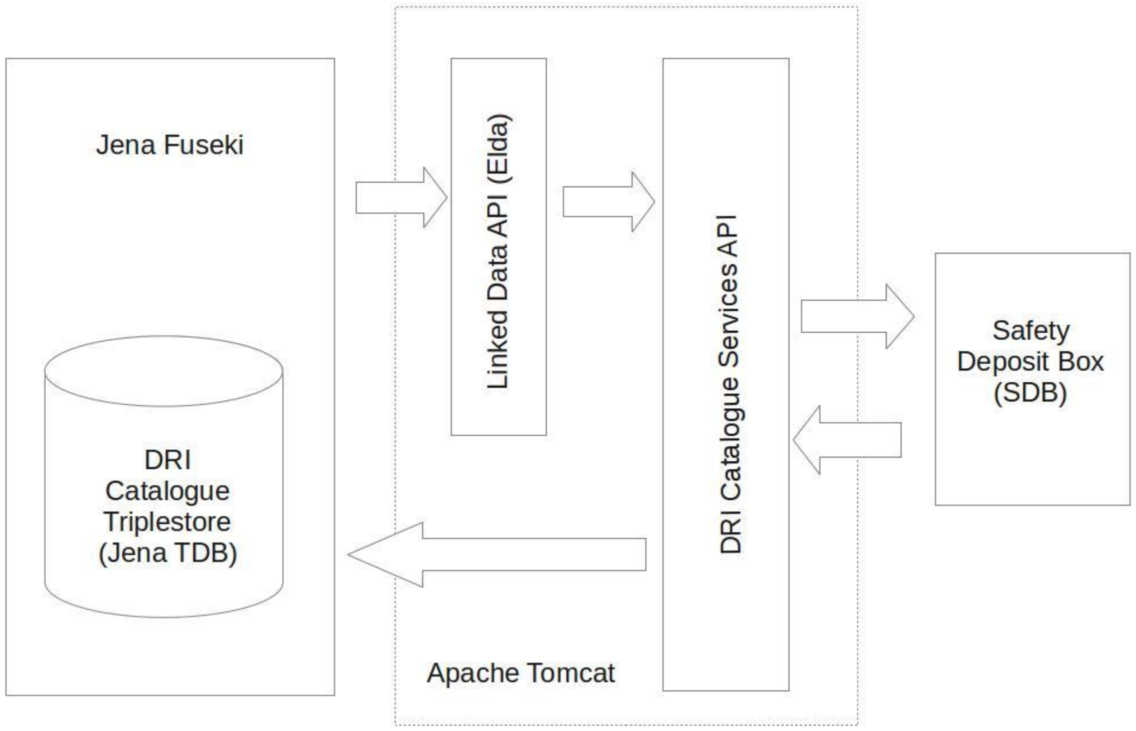
Within this statement you will notice the dri:parent+ property. The + signifies a SPARQL 1.1 property path, in other words it will find the parent and the parent's parent and so on until there are no more matches. The dri prefix indicates that this parent property is part of the DRI vocabulary which is discussed in more detail later.

Elda was chosen as the read API in front of Jena Fuseki. This meant that all reads would go via Elda and all writes, updates and deletes would go directly to Fuseki.

One final problem remained with regards to the acceptance of this solution within TNA. It was perceived that there was some risk involved in using such a (relatively) new technology stack and therefore the impact on other systems, in particular SDB, had to be kept to a minimum. To solve this problem it was decided to develop a simple bespoke Catalogue Service API between SDB and the DRI Catalogue. Having SDB talk to this API meant that if the new DRI Catalogue failed to deliver the expected results for some reason then it could be swapped for another solution with only the Catalogue Service API needing to be re-engineered.

Both Elda and the Catalogue Service API would be run within an Apache Tomcat web container in common with other components of DRI and SDB. Jena Fuseki however would need to be run as a standalone application as there is currently no mechanism for deploying Fuseki as a web archive within Tomcat, although it is a feature that has been discussed in depth [19].

The final design of the catalogue system is shown below.



3.3 *DRI Vocabulary*

During the initial testing of a graph based catalogue it was quickly discovered that a vocabulary for catalogue terms would need to be developed in order to be able to describe catalogue specific entities and relationships. This vocabulary would need to be established, as far as possible, in advance of the initial import of catalogue data. Although it would be possible to reclassify terms once the data was converted into RDF, it is more expedient to resolve these terms up front and, as far as possible, avoid renaming later.

In keeping with the W3C's guidelines laid out in the Cookbook for Open Government Linked Data [20] existing vocabularies are re-used as much as possible. Extensive use is therefore made of OWL [21], Dublin Core [22], RDF Schema [23] and XML Schema [24] however quite a few terms are very specific to the Catalogue. Although catalogue items have “parents” suggesting use of the FOAF vocabulary it was decided that catalogue items are emphatically not people and the rules around people's parents (one mother and one father) do not apply in this case. Use of the FOAF vocabulary could therefore cause confusion at a later date. A DRI parent term was therefore created.

The full vocabulary is described in the appendix.

3.4 *Implementation*

The first stage of the implementation was to extract the existing catalogue data from the RDBMS where it was held. D2RQ was to be used for this as had been done for the initial trial. The difference now was that we had established a vocabulary the terms to be used in the new catalogue. With this in place it was possible to map the columns in the database to the terms that would be used in the RDF. This was done using the D2RQ mapping file, a sample of which is shown below.

```
# Table TNADRI.COLLECTION
map:collection a d2rq:ClassMap;
	d2rq:dataStorage map:database;
	d2rq:uriPattern
"http://nationalarchives.gov.uk/dri/catalogue/collection/@@TNADRI.COLLECTION.UUI
D@@";
```

```
      d2rq:class dri:Collection;
      d2rq:classDefinitionLabel "TNADRI.COLLECTION";
      .
```

In this example a row from the TNADRI.COLLECTION table is mapped to an instance of the dri:Collection class and assigned a URI based on the UUID column of the table. This means that we end up with a resource described by the following RDF triple (in Turtle format):

```
<http://nationalarchives.gov.uk/dri/catalogue/collection/example1234>
      rdf:type dri:Collection .
```

In other words, a resource of type collection.

Using this mapping technique the contents of the RDBMS catalogue were dumped into text files in the Turtle format, which totalled approximate 1Gb of data. Approximately 8 million triples.

The second stage was to load these triples into TDB using the **tdbloader** command line tool which is part of the Apache Jena framework. However this raw data was still not in the desired format for use within the new catalogue. For starters the closure information was not linked to the resources it referred to. Closure information is comprised of five elements:

- **Closure period** – how long the record must remain closed
- **Description status** – whether the description is open or closed
- **Document status** – whether the document is open or closed
- **Review Date** - when the closure is due for review or when the record was opened
- **Closure type** – gives more information about the type of closure

However closure cannot be said to be a resource in its own right as it only exists in the context of a catalogue item. RDF has a concept for this type of information which seemed highly appropriate, the **blank node** or **bNode**. Creating these blank nodes would require some transformation of the data however. While there is no equivalent of the XML transformation language XSLT for RDF the SPARQL language itself allows transformation through use of CONSTRUCT queries. In this case new triples can be created based on existing triples.

By loading the data into TDB and then using the SPARQL query endpoint in Fuseki to run construct queries, it was possible to generate new triples in the desired format that could be downloaded in Turtle format from Fuseki and then reloaded into a new instance of TDB. The following CONSTRUCT query shows how the new triples could be created. In this case by combining file and closure information into a new set of triples relating to a single resource with the closure information held in a blank node signified by the square brackets in the construct query:

```
PREFIX closure:  &lt;http://nationalarchives.gov.uk/dri/catalogue/closure#&gt;
PREFIX dcterms: &lt;http://purl.org/dc/terms/&gt;
PREFIX dri: &lt;http://nationalarchives.gov.uk/terms/dri#&gt;
PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt;
PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt;
PREFIX xsd: &lt;http://www.w3.org/2001/XMLSchema#&gt;

CONSTRUCT
{
      ?file rdf:type dri:File ;
            rdfs:label ?name ;
            dri:directory ?directory ;
      dcterms:identifier ?identifier ;
            dri:closure [
                  rdf:type dri:Closure ;
                  dri:closurePeriod ?closureCode ;
                  dri:descriptionStatus ?descriptionStatus ;
            dri:documentStatus ?documentStatus ;
                  dri:reviewDate ?openingDate ;
            dri:closureType ?newClosureType
            ] ;
      .
}
WHERE
{
      ?file rdf:type dri:File ;
            rdfs:label ?name ;
            dri:directory ?directory ;
      dcterms:identifier ?identifier .
      ?closure rdfs:label ?identifier ;
            dcterms:creator ?creator ;
            dri:closureCode ?closureCode ;
      dri:closureType ?closureType ;
      dri:descriptionStatus ?descriptionStatus ;
      dri:documentStatus ?documentStatus ;
      dri:openingDate ?openingDate ;
      BIND(IF(?closureType = 1, closure:A, closure:U)  AS ?newClosureType)
   .
}
```

With the data cleaned, refactored and accessible via Fuseki, development of the Catalogue Service API could begin.

3.5 Catalogue Service API

The Catalogue Service API is a RESTful Jersey JAX-RS [25] application that reads and writes data to the TDB triple-store. As per the design, reading data is done via Elda and writing data is done via Fuseki. The actual API itself is extremely simple and returns data in an XML or JSON format. For example, in the case of creating a new catalogue entry it simply takes in the TNA catalogue reference as a request parameter and generates an entry in the DRI catalogue of the appropriate type (this depends on the URI called) and, if successful, returns the unique identifier in a snippet of XML as follows:-

```
<result xmlns=”http://nationalarchives.gov.uk/dri/catalogue”>
      <uuid>e9d8f987-5d49-40f2-869b-a2172e3d362c</uuid>
</result>
```

In the process it generates a new unique ID, writes out the triple to file using the Jena API to then it to Fuseki over HTTP and, if all goes well, returns the UUID in the response as shown above.

To create the new triples it first creates an ontology model using the Jena API which is populated with the required classes from our vocabulary:

```
protected Model createModel() {
      OntModel model = ModelFactory.createOntologyModel(OntModelSpec.RDFS_MEM);
      collectionClass = model.createClass(DRI_TERMS_URI + "Collection");
      batchClass = model.createClass(DRI_TERMS_URI + "Batch");
      deliverableUnitClass = model.createClass(
            DRI_TERMS_URI + "DeliverableUnit");
      preservationManifestationClass = model.createClass(
            DRI_TERMS_URI + "PreservationManifestation");
      exportClass = model.createClass(DRI_TERMS_URI + "Export");
      recordListClass = model.createClass(DRI_TERMS_URI + "RecordList");
      model.setNsPrefix("dri", DRI_TERMS_URI);
      model.setNsPrefix("dcterms", DCTerms.getURI());
      return model;
}
```

It then creates the necessary resources and literals which are added to the model.

```
private String addCollection(Model model, String collectionRef, String label) {
      UUID uuid = UUID.randomUUID();
      Resource collection = model.createResource(
            COLLECTION_URI + uuid.toString());
      collection.addLiteral(RDFS.label, collectionRef);
      collection.addProperty(RDF.type, collectionClass);
      collection.addLiteral(
            DCTerms.created, new XSDDateTime(Calendar.getInstance()));
      collection.addLiteral(DCTerms.identifier, uuid.toString());
      collection.addLiteral(DCTerms.description, label);
      return uuid.toString();
}
```

The model is then written to a file in Turtle format which is posted to Fuseki via HTTP. In the case of a SPARQL update it writes out a SPARQL file and posts this to Fuseki:

```
public ResultWrapper createRecordListAddItemFile(String recordListUuid,
            String itemUuid) {
      Model model = createModel();
      Resource recordList = model.createResource(
            RECORD_LIST_URI + recordListUuid);
      Literal itemUuidLiteral = model.createTypedLiteral(itemUuid);
      QuerySolutionMap parameters = new QuerySolutionMap();
      parameters.add("recordList", recordList);
      parameters.add("itemUuid", itemUuidLiteral);
      ParameterizedSparqlString paramString =
            new ParameterizedSparqlString(getQueryProlog() +
             getRecordListItemAddString(), parameters);
      UpdateRequest update = paramString.asUpdate();
      File queryFile = getFileHandler().writeUpdateFile(
            update, "insert" + "_" + getDtf().print(new DateTime()) + ".rq");
      ResultWrapper rw = new ResultWrapper(queryFile, null);
      return rw;
}
```

In the above example the getQueryProlog() and getRecordListItemAddString() methods generated the necessary text for the SPARQL update as follows:

```
protected String getQueryProlog() {
      String prologString =
            "PREFIX dcterms: &lt;" + DCTerms.getURI() + "&gt; \n" +
            "PREFIX dri: &lt;" + DRI_TERMS_URI + "&gt; \n" +
            "PREFIX rdf: &lt;" + RDF.getURI() + "&gt; \n" +
            "PREFIX rdfs: &lt;" + RDFS.getURI() + "&gt; \n" +
            "PREFIX owl: &lt;" + OWL.getURI() + "&gt; \n" +
            "PREFIX xsd: &lt;" + XSD.getURI() + "&gt; \n";
      return prologString;
}

private String getRecordListItemAddString() {
      StringBuilder deleteBuilder = new StringBuilder();
      deleteBuilder.append(
            "INSERT { ?recordList dri:recordListMember ?item . }");
      deleteBuilder.append("WHERE { ?item dcterms:identifier ?itemUuid . }");
      return deleteBuilder.toString();
}
```

In the case of reading the data it accesses Elda, requesting XML format (for which a schema has been developed) and unmarshalls this into JAXB objects from which it extracts the required information. The results are then marshalled into the much simpler XML format described above.

3.6 Insights, Issues and Limitations

3.6.1 Elda

Whilst Elda and the Linked Data API provide enormous benefits for users in terms of simplifying access to triple-stores it has provided some challenges to the developers wanting to implement SPARQL queries and make use of the XML result format.

Elda extension
Early on in the development process it came to light that Elda had a significant limitation in the type of SPARQL queries it could run. Although Elda provides read access to the underlying triple-store it was found to be impossible to create new triples through the use of CONSTRUCT queries. There was an early requirement to know whether a record was open, closed or partially closed. This information is not held within the knowledge-base but has to be generated as the result of a query. Although you can embed SPARQL SELECT queries within Elda there was no working mechanism for a CONSTRUCT query. As Elda is open source and written in Java, it was feasible for TNA to add this functionality as an extension, which was subsequently done.

As the project has continued however, we have found this functionality to be of limited benefit. Although there is a temptation to use CONSTRUCT queries for all kinds of things, quite often there is a more efficient way to achieve the desired result, for example by configuring a viewer within the API to control the properties returned.

Furthermore it was found that complex construct queries that relied on a number of SELECT sub-queries became difficult to debug as there was limited visibility of what was happening within the query. This led to a rethink whereby more complex queries were built up gradually using the Jena API calling a combination of simpler sub-queries from Elda. This enabled us to embed logging within the Java which could show the results of sub-queries and this also gave us a structure for providing much better error handling within the application.

Whilst the Elda construct extension is still in use, it is likely to be gradually phased out in the future.

Elda caching
Elda is very efficient at caching query results. Whilst this reduces query times for frequently called queries it can cause some confusion and frustration when the data in the underlying triple-store is being frequently updated. The cache can however be cleared by calling...

```
http://{server-name}/catalogue/control/clear-cache
```

...and so the catalogue API calls this before any query which is likely to be affected by updates.

Elda XML
An early design decision was to use the XML results from Elda and then unmarshall the results as JAXB objects with the Catalogue Service from where the required values could be extracted and returned. This meant creating our own schema for the XML as there is no publicly available one and, in any case, the XML returned is dependent on the underlying vocabularies being used. Because we had created our own vocabulary we had no choice but to create our own schema.

An important point to note with Elda is that in the case of item endpoints (i.e. where only one result will ever be returned) a **primaryTopic** element is used to contain the result. In the case of a list endpoint, which can return zero to many results, an **items** element is returned containing one **item** element for each result. Understanding this enabled us to write generic services for reading and writing these two types of result.

Elda Turtle
It is proposed that in future the DRI Catalogue Service will make use of Turtle format results from Elda instead of XML. Turtle response times seem to be considerably faster than the equivalent XML (presumably because this is closer to the native format of the data) and Jena provides a well developed API for reading RDF, meaning that Plain Old Java Objects could be used within the Catalogue Service rather than specifically JAXB objects.

3.6.2 TDB

One point that was established early on in the design process was that TDB could only be safely accessed from one Java Virtual Machine. We had therefore chosen to use Fuseki. This presented us with a problem however when it came to performing backups. We wanted the backups to be controlled by a CRON job run on the server itself. How could the server safely carry out a backup if TDB was being controlled by Fueski?

The answer was provided by the Fuseki management console. A little documented but extremely useful feature of Fuseki which meant that the backup could be controlled safely via the management console by calling the relevant URL from the server shell:

```
wget -O - --post-data='cmd=backup&amp;dataset=/catalogue' http://{server-
address}/mgt
```

3.6.3 Xturtle

Working with a semantic web technology stack means that you frequently have to manually edit RDF files in Turtle format. For example the mapping files for D2RQ and configuration files for Elda are written in Turtle. Without a syntax highlighter, errors in these files can be difficult to spot, especially as they get larger.

Xturtle [27], which comes as a plug-in for Eclipse, provided us with a very useful tool for editing these files, especially as the majority of our developers were already using Eclipse as a Java IDE.

3.6.4 Scardf

Scala is being used increasingly within the DRI project. The reasons for this are its scalability, compatibility with Java (Scala programs compile to JVM byte-code) and concise syntax which results in less lines of code, better readability and fewer opportunities for errors.

For this reason, **scardf** [28] is being considered for future development of the Catalogue Service API within DRI. The **ScardfOnJena** API appears to offer a slot in replacement for the Java API currently being used. If the project were being started over again now this may well have been the preferred language for the Catalogue Service API, rather than Java.

3.6.5 Scale and performance

So far the DRI Catalogue has only been used for the modest amount of data currently held within DRI (the Durham Home Guard Records). Whilst no performance issues have been encountered so far it is likely that this will become more of a concern as data accumulates. Actual volumes are very difficult to predict and the nature of an archive is that it will always grow and never shrink. For this reason extensibility has been a primary concern in all aspects of DRI development and the catalogue is no different. Whilst we cannot predict future performance we are encouraged by a number of factors:

Firstly the graph solution was chosen because of its superior performance.

Secondly we are confident that the existing catalogue architecture can be scaled horizontally. This has already been done with other components. Additional servers could be added, running further instances of the triple-store, possibly containing completely separate graphs. Indeed there is a certain logic to one graph per catalogue collection. With a SPARQL endpoint configured on each collection it would be possible to have a “catalogue of catalogues” which would provide us with a pan-archive search capability.

Finally, if the existing open source framework fails to meet expectation the standards-based approach means that we can move to an alternative framework. For example, TNA has previously worked with Ontotext in the development of the UK Government Web Archive [29] which uses an OWLIM [30] triple-store containing billions of triples.

We are confident that whatever issues arise with scale in the future, we are in a strong position to address them.

4 The Future

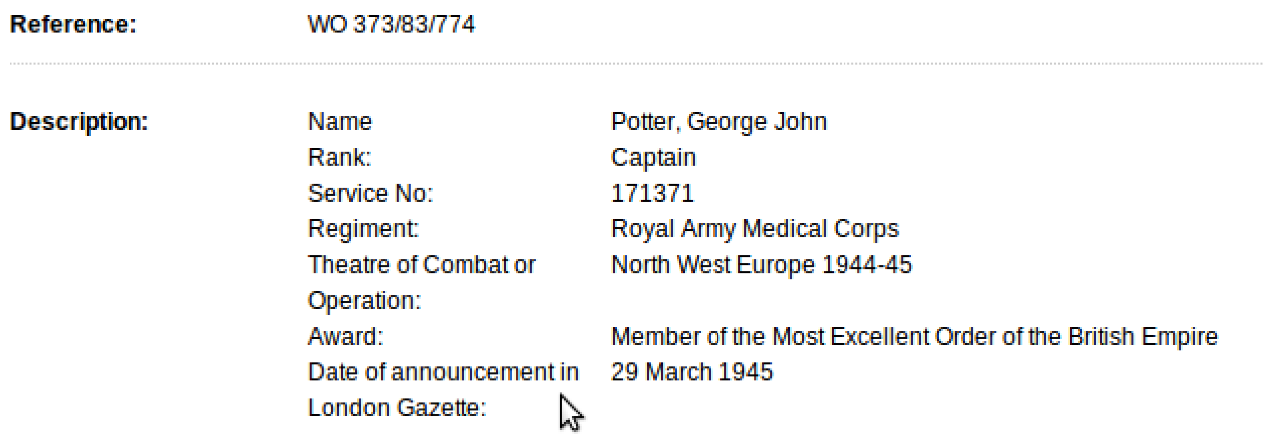
So far what has been achieved with the DRI Catalogue is a fundamental remodelling of the catalogue using Semantic Web technologies and a successful implementation of a solution to the problem of closure. With these new technologies in place further enhancements of the catalogue can be considered which were not previously possible.

4.1 Named Entity Recognition

Using a semantic approach enables an Open World Assumption. This means that it is "*implicitly assumed that a knowledge base may always be incomplete"* [31]. It is always possible to add more information as we learn it. As we extract more information from the content going into the archive we can add it to our knowledge-base and this information can be transferred to Discovery where it can be viewed and searched.

What could this really mean for people viewing the record on the Web?

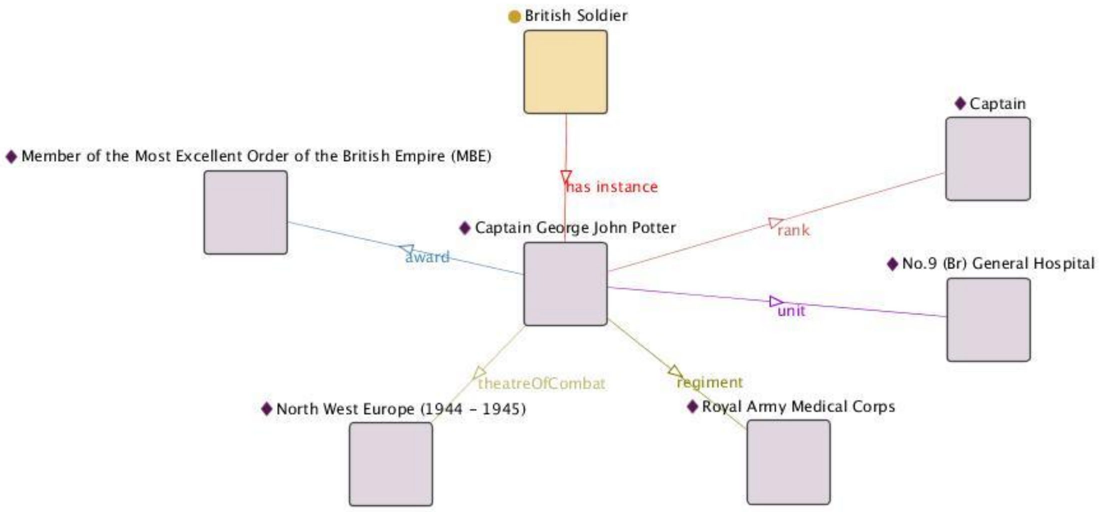
As a human being with a reasonable level of education we can look at the following entry and make a reasonable assumption that it refers to a soldier receiving an award for conduct during the Second World War.



Reference:	WO 373/83/774	
Description:	Name	Potter, George John
	Rank:	Captain
	Service No:	171371
	Regiment:	Royal Army Medical Corps
	Theatre of Combat or Operation:	North West Europe 1944-45
	Award:	Member of the Most Excellent Order of the British Empire
	Date of announcement in London Gazette:	29 March 1945

It is easy to forget that the machine has no idea about these concepts. As far as the computer is concerned this is just a collection of string values. If we want the computer to help us search for information we need to tell it about the concepts that we already understand. We need to tell it that George John Potter is a Captain which is a kind of officer, which is a kind of soldier, which is a kind of person who serves in the army. We can tell the computer that he served in a "regiment" which is part of an "army" and the regiment is called the "Royal Army Medical Corps". We can also tell it that we know this to be a regiment within the "British Army". If we identify the Royal Army Medical Corps as a single entity we can then say that other people also served in the Royal Army Medical Corps and point at the same resource. With the addition of these simple pieces of information suddenly the computer can help us enormously in our search. The "Royal Army Medical Corps" is no longer just a piece of text but a concept that the computer can relate to other entities it also knows about. It can now tell us whatever it knows about the Royal Army Medical Corps. For example who else served in the Royal Army Medical Corps? What theatres of combat or operations did it take part in and when.

The follow is a machine readable representation of Captain George John Potter's record previously discussed. The nodes in the graph represent either resources or literal values and the lines connecting the nodes are the properties.


British Soldier
Captain
Member of the Most Excellent Order of the British Empire (MBE)
has instance
Captain George John Potter
rank
No.9 (Br) General Hospital
award
unit
theatreOfCombat
regiment
North West Europe (1944 – 1945)
Royal Army Medical Corps

4.2 Ontology-driven NLP

Let's say that we build up a dictionary (or ontology) of these terms. As the documents are loaded into the archive they could be scanned for these terms using a Natural Language Processing tool such as GATE [32] Let's take a real example from Discovery. The following is the text of the recommendation for award for the aforementioned George John Potter:

> "*From 1 Aug 44 to 20 Oct 44 Bayeux, Rouen and Antwerp. During the period from 1 Aug to date this officer has carried the principal strain of establishing and re-establishing the hospital in three situations. His unrelenting energy, skill and patience have been the mainstay of the unit. His work as a quartermaster is the most outstanding I have met in my service. (A.R.ORAM)*
> *Colonel Comdg. No.9 (Br) General Hospital*"

I have underlined the terms that could reasonably be understood by the computer with the help of a dictionary. We have place names, a date range, a keyword "hospital" a rank (Colonel) the name of a unit (No.9 British General Hospital) and the name of a person (A.R.Oram). The computer could reasonably be expected to recognise A.R.Oram as he is himself the subject of a record (he also received an award). Although the computer would know him as Algar Roy Oram, it would be computationally simple to add another name value with this common abbreviation. Likewise "Br" could be recognised as an abbreviation for British. As a result of this process the computer could perform *reification*. In other words it could make statements that may or may not be true but which are plausible conclusions. For example, it could say "Colonel Algar Roy Oram served with Captain George John Potter" or "Colonel Algar Roy Oram was based in Antwerp". This is inferred knowledge and may not be factual but it could be marked as a theory in the knowledge-base, until such a time as it can be shown to be (or not to be) a fact.

4.3 Semantic Search

Keyword searches tend to deliver a large volume of irrelevant results because it is not usually possible to communicate the desired context to the search engine mechanism. Semantic searching on the other hand allows for selection of ontological terms when searching. For example if you search for "George John Potter" in Discovery you get 361 results whereas in fact, there is only one result that actually refers to a person with this name. Imagine if you were able to say that you were looking for a match on person's name. To paraphrase Bill Gates, **context is king** when doing Semantic Search. This technique is known as *query string extension*.

Semantic search offers other possibilities as well. If it knows you are searching for a person named George John Potter it would be possible for a semantic knowledge base to look for terms that were closely associated to matches. For example, George Potter's award was given by a Colonel A.R. Oram. This information could be held within the graph of knowledge and therefore a semantic search could bring back results linking to Colonel Algar Roy Oram as well. Through painstaking research it is possible to discover this information now and this reveals that Algar Oram's records describe in great detail the difficult and dangerous situation that he shared with George Potter in Antwerp during the Second World War. In this way we can discover new information about George Potter's experiences during the war that are not directly expressed but strongly implied. A semantic search could've provided this information in seconds. This technique is referred to as *cross-referencing*.

Because the knowledge base is a graph, it is possible to go further into searches than traditional keyword searching. For example in the case of George Potter you could continue the search using the link to Algar Oram. In this case you would find that he too was awarded a medal and that this medal was awarded by a very senior officer in the Royal Army Medical Corps who had himself been decorated many times. Some of the medals this individual received were for extraordinary acts of bravery that saved many lives and the letters of recommendation make gripping reading. This may not be what you were looking for originally but it provides interesting context that you would not otherwise have found. This is known as *exploratory search*.

Another possibility would be to allow machine reasoning within the catalogue. This would enable rules to be applied. For example if the machine knew that George Potter “served” with something called “No.9 (British) General Hospital” in the “theatre of combat” called “North West Europe (1944-1945)” it would be possible to reason that certain other people “served with” George Potter. This is new knowledge that is not currently available in the catalogue. It is an example of the using the graph and the ontology to do *reasoning*.

4.4 Linked Data (for the things the archive doesn't have)

Whilst TNA is a huge national source of information, there is lots of pieces of official information that are simply not there, but kept elsewhere. For example birth, marriage and death certificates from 1837 onwards are held by the General Register Office or at local register offices - prior to 1837 they are kept in parish registers; military service records for soldiers and officers after 1920 are maintained by the Ministry of Defence; the London Gazette [33] is the official UK Government Newspaper and has it's own knowledge-base of historical articles. When it comes to places, an organisation such as the Ordnance Survey would be a excellent source of information. As long as an archive is kept in a silo, its world view will remain incomplete.

Linked Data [34], the brainchild of World Wide Web inventor Tim Berners-Lee, provides a way of un-siloing data and it is based on the standardised technologies of the Semantic Web. By providing its information as Linked Data, i.e. in RDF-based machine readable formats, other organisations or individuals can connect the dots between items of data. In the case of George Potter we know from his record that he set-up field hospitals in Bayeux, Rouen and Antwerp. Whilst the archive may not be an authority on these places, someone could make a connection between a record held in the archive and a resource that gives much more information about these places. For example, DBPedia [35], which contains key information taken from Wikipedia provides a great deal of useful and increasingly reliable data. This could work both ways so not only could external users make these connection but the archive itself it could pull in information from other places that provide Linked Data. For example at TNA, UK location information could be pulled in from the Ordnance Survey, allowing relevant maps and other information to be shown within Discovery. At the same time the archive could contribute knowledge back to DBPedia, adding new entries and improving the quality of existing entries on which it is an authority.

4.5 Crowd-sourced linking

TNA has a vast amount of information and it is unrealistic to think that a machine is going to be able to read everything and make all the right connections. It is going to make mistakes and it is going to miss things. This is particularly the case with digitised documents, i.e. scanned paper documents. Even with modern OCR technology it is not possible to accurately read all these items. Providing a way for researchers, archivists and the general public to make connections would add enormous value to the knowledge base. Allowing users to add tags to a document is a common way to crowd-source information but the terms tend to be very specific and individualistic. Imagine if it was possible to select dictionary terms from a semantic knowledge-base to apply to these items, many new and previous overlooked connections could be made.

Acknowledgments

I am grateful to a host of people whose knowledge and expertise have made this work possible. In particular to Bob DuCharme for his excellent book on SPARQL (now worn out), Dr Harald Sack at the Hasso Plattner Institute for his thorough course on Semantic Web Technologies, the guys at Epimorphics who have given us invaluable support and of course, above all, the folks at The National Archives whose forward thinking has enabled this project, in particular David Thomas, Tim Gollins, Diana Newton, Peter Malewski and Adam Retter.

References

[1] Our new catalogue: the Discovery service http://www.nationalarchives.gov.uk/about/new-catalogue.htm - The National Archives
[2] Domesday Reloaded http://www.bbc.co.uk/history/domesday/story - BBC
[3] Discovery http://discovery.nationalarchives.gov.uk - The National Archives
[4] Freedom of Information Act 2000 http://www.legislation.gov.uk/ukpga/2000/36/contents
[5] D2RQ http://d2rq.org - Free University of Berlin
[6] Apache Jena TDB http://jena.apache.org/documentation/tdb/ - Apache Software Foundation
[7] Apache Jena Fuseki http://jena.apache.org/documentation/serving_data/ - Apache Software Foundation
[8] SPARQL 1.1 Property Paths http://www.w3.org/TR/sparql11-property-paths/ - W3C
[9] Terse RDF Triple Language http://www.w3.org/TR/turtle/ - W3C
[10] Choosing technology https://www.gov.uk/service-manual/making-software/choosing-technology.html#level-playing-field - GOV.UK
[11] Digital by Default Service Standard https://www.gov.uk/service-manual/digital-by-default - GOV.UK
[12] Safety Deposit Box http://www.digital-preservation.com/solution/safety-deposit-box/ - Tessella
[13] UUID http://docs.oracle.com/javase/6/docs/api/java/util/UUID.html - Oracle and/or its affiliates
[14] SPARQL 1.1 Query Language http://www.w3.org/TR/sparql11-query/ - W3C
[15] SPARQL 1.1 Update http://www.w3.org/TR/sparql11-update/ - W3C
[16] TDB Transactions http://jena.apache.org/documentation/tdb/tdb_transactions.html - The Apache Software Foundation
[17] Linked Data API http://code.google.com/p/linked-data-api/
[18] Elda Linked Data API Implementation http://code.google.com/p/elda/ - Epimorphics
[19] Deliver Fuseki as a WAR file https://issues.apache.org/jira/browse/JENA-201
[20] Linked Data Cookbook http://www.w3.org/2011/gld/wiki/Linked_Data_Cookbook#Step_3_Re-use_Vocabularies_Whenever_Possible - W3C
[21] OWL Web Ontology Language Overview http://www.w3.org/TR/owl-features/ - W3C
[22] DCMI Metadata Terms http://dublincore.org/documents/dcmi-terms/ - Dublin Core Metadata Initiative
[23] RDF Vocabulary Description Language 1.0: RDF Schema http://www.w3.org/TR/rdf-schema/ - W3C
[24] XML Schema http://www.w3.org/XML/Schema - W3C
[25] FOAF Vocabulary Specification 0.98 http://xmlns.com/foaf/spec/ - Dan Brickley and Libby Miller
[26] Jersey JAX-RS (JSR 311) Reference Implementation https://jersey.java.net/
[27] Xturtle: an eclipse / Xtext2 based editor for RDF/Turtle files http://aksw.org/Projects/Xturtle.html - Agile Knowledge Engineering and Semantic Web (AKSW)
[28] scardf Scala RDF API http://code.google.com/p/scardf/
[29] UK Government Web Archive http://www.nationalarchives.gov.uk/webarchive/ - The National Archives
[30] Ontotext OWLIM http://www.ontotext.com/owlim - Ontotext
[31] Foundations of Semantic Web Technologies; PascalHitzler, MarkusKrötzsch, SebastianRudolph; Taylor & Francis Group, LLC; ISBN 978-1-4200-9050-5; Chapman &amp; Hall/CRC
[32] GATE General Architecture for Text Engineering http://gate.ac.uk/ - The University of Sheffield
[33] The London Gazette http://www.london-gazette.co.uk/
[34] Linked Data - Connect Distributed Data across the Web http://linkeddata.org/
[35] DBPedia http://dbpedia.org/

Appendix – The DRI Vocabulary

This vocabulary has been laid out using the Turtle W3C format.

```
<http://nationalarchives.gov.uk/terms/dri>
  rdf:type owl:Ontology ;
  owl:imports &lt;http://purl.org/dc/elements/1.1/&gt; .

:Batch
  rdf:type rdfs:Class ;
  rdfs:label "Batch"^^xsd:string ;
  rdfs:subClassOf rdfs:Class .

:Closure
  rdf:type rdfs:Class ;
  rdfs:label "Closure"^^xsd:string ;
  rdfs:subClassOf rdfs:Class .

:ClosureType
  rdf:type rdfs:Class ;
  rdfs:label "Closure type"^^xsd:string ;
  rdfs:subClassOf rdfs:Class .

:Collection
  rdf:type rdfs:Class ;
  rdfs:label "Collection"^^xsd:string ;
  rdfs:subClassOf rdfs:Class .

:DeliverableUnit
  rdf:type rdfs:Class ;
  rdfs:label "Deliverable unit"^^xsd:string ;
  rdfs:subClassOf :Item .

:Directory
  rdf:type rdfs:Class ;
  rdfs:label "Directory"^^xsd:string ;
  rdfs:subClassOf :Resource .

:Export
  rdf:type rdfs:Class ;
  rdfs:label "Export"^^xsd:string ;
  rdfs:subClassOf rdfs:Container .

:ExportStatus
  rdf:type rdfs:Class ;
  rdfs:label "Export status"^^xsd:string ;
  rdfs:subClassOf rdfs:Class .

:File
  rdf:type rdfs:Class ;
  rdfs:label "File"^^xsd:string ;
  rdfs:subClassOf :Resource .

:Item
  rdf:type rdfs:Class ;
  rdfs:label "Item"^^xsd:string ;
  rdfs:subClassOf rdfs:Class .

:Manifestation
  rdf:type rdfs:Class ;
  rdfs:label "Manifestation"^^xsd:string ;
  rdfs:subClassOf :Item .

:PresentationManifestation
```

```
  rdf:type rdfs:Class ;
  rdfs:label "Presentation manifestation"^^xsd:string ;
  rdfs:subClassOf :Manifestation .

:PreservationManifestation
  rdf:type rdfs:Class ;
  rdfs:label "Preservation manifestation"^^xsd:string ;
  rdfs:subClassOf :Manifestation .

:RecordList
  rdf:type rdfs:Class ;
  rdfs:label "Record list"^^xsd:string ;
  rdfs:subClassOf rdfs:Container .

:Resource
  rdf:type rdfs:Class ;
  rdfs:label "Resource"^^xsd:string ;
  rdfs:subClassOf rdfs:Class .

:batch
  rdf:type rdf:Property ;
  rdfs:domain :Item ;
  rdfs:label "batch"^^xsd:string ;
  rdfs:range :Batch .

:closure
  rdf:type rdf:Property ;
  rdfs:domain :DeliverableUnit ;
  rdfs:label "closure"^^xsd:string ;
  rdfs:range :Closure .

:closurePeriod
  rdf:type rdf:Property ;
  rdfs:domain :Closure ;
  rdfs:label "closure period"^^xsd:string ;
  rdfs:range xsd:decimal .

:closureType
  rdf:type rdf:Property ;
  rdfs:domain :Closure ;
  rdfs:label "closure type"^^xsd:string ;
  rdfs:range :ClosureType .

:collection
  rdf:type rdf:Property ;
  rdfs:domain :Item ;
  rdfs:label "collection"^^xsd:string ;
  rdfs:range :Collection .

:completedDate
  rdf:type rdf:Property ;
  rdfs:label "completed date"^^xsd:string ;
  rdfs:range xsd:date ;
  rdfs:subPropertyOf dcterms:date .

:descriptionStatus
  rdf:type rdf:Property ;
  rdfs:domain :Closure ;
  rdfs:label "description status"^^xsd:string ;
  rdfs:range xsd:decimal .

:directory
  rdf:type rdf:Property ;
  rdfs:domain :Resource ;
  rdfs:label "directory"^^xsd:string ;
  rdfs:range xsd:string .

:documentStatus
  rdf:type rdf:Property ;
  rdfs:label "document status"^^xsd:string .

:exportMember
  rdf:type rdf:Property ;
  rdfs:label "export member"^^xsd:string ;
  rdfs:subPropertyOf rdfs:member .

:exportStatus
  rdf:type rdf:Property ;
  rdfs:label "export status"^^xsd:string ;
  rdfs:range :ExportStatus .

:file
  rdf:type rdf:Property ;
  rdfs:domain :Manifestation ;
  rdfs:label "file"^^xsd:string ;
  rdfs:range :File .

:parent
  rdf:type rdf:Property ;
  rdfs:domain :Item ;
  rdfs:label "parent"^^xsd:string ;
  rdfs:range :Item .

:recordListMember
  rdf:type rdf:Property ;
  rdfs:label "record list member"^^xsd:string ;
  rdfs:subPropertyOf rdfs:member .

:reviewDate
  rdf:type rdf:Property ;
  rdfs:domain :Closure ;
  rdfs:label "review date"^^xsd:string ;
  rdfs:range xsd:dateTime ;
  rdfs:subPropertyOf dcterms:date .

:username
      rdf:type rdf:Property ;
      rdfs:label "username"^^xsd:string ;
      rdfs:range xsd:string
```