Course: How to use Linked Data/Providing Linked Data/Linked Data Catalogs and Tools for Providing Linked Data

0.1 3.8.1 Linked data catalogs

As we saw in the previous section, data catalogs are platforms that provide access to a wide range of datasets from different domains. Below we describe CKAN that can be used to build data catalogs and The Data Hub, which is a public catalog of datasets.

CKAN [20] is an open source platform for developing a catalog for a number of datasets. CKAN may be used by an organisation for internally managing their datasets. These datasets need not be publically available as part of the Linking Open Data cloud. CKAN features a number of tools for data publishers to support:

- Data harvesting
- Creation of metadata
- Access mechanisms to the dataset
- Updating the dataset
- Monitoring the access to the dataset
Figure 16. CKAN.

CKAN has a schema for describing contributed datasets. This is similar to the VoID schema described in section 3.5.1.

**Figure 17: Overview of the CKAN portal (from [20]).**

The Data Hub [21] is a community-run data catalog that contains more than 5,000 datasets. Data Hub is implemented using the CKAN platform and can be used to find public datasets. In The Data Hub, datasets can be organised into groups each having their own user permissions. Groups may be topic based (e.g. archaeological datasets) or datasets in a particular language or originating from a certain country.

**Figure 18: The Data Hub.**

The group “Linking Open Data Cloud” catalogs datasets that are available on the Web as Linked Data.
Figure 19: The group Linking Open Data Cloud on The Data Hub.

Every bubble in the Linking Open Data cloud (shown in Figure 3.20) is registered with the Data Hub. For a dataset to be included in this cloud it must satisfy the following criteria:

- The dataset must follow the Linked Data principles (see section 3.3)
- The dataset must contain at least 1,000 RDF triples
- The dataset must contain at least 50 RDF links to a dataset that is already in the diagram
- Access to the dataset must be provided

Once these criteria are met, the data publisher must add the dataset to the Data Hub catalog, and contact the administrators of the Linking Open Data Cloud group.

Figure 20: Linking Open Data Cloud.
0.2 3.8.2 Linked data and commercial search engines

Search engines collect information about web resources in order to enrich how search results are displayed. Snippets are the few lines of text that appear underneath a search result link to give the user a better sense of what can be found on that page and how it relates to the query. Rich snippets provide more detailed information, by understanding something about the content of the page featured in the search results. For example, a rich snippet of a restaurant might show a customer review or menu information. A rich snippet for a music album might provide a track listing with a link to each individual track.

![Example of a Rich Snippet](image)

Figure 21: Example of a Rich Snippet.

Rich snippets are created from the structured data detected in a web page. The structured data found in a web page may be represented using RDFa as the mark up format and schema.org as the vocabulary. schema.org is a collaboration between Google, Microsoft and Yahoo! to develop a markup schema that can be used by search engines to provide richer results.

schema.org provides a collection of schemas for describing different types of resource such as:

- Creative works: Book, movie, music recording, ...
- Embedded non-text objects
- Event
- Health and medical types
- Organization
- Place, local business, restaurant
- Product, offer, aggregate offer
- Review, aggregate rating

Data represented using schema.org is recognized by a number of search engines such Bing, Google, Yahoo! and Yandex. schema.org also offers an extension mechanism that a publisher can use to add more specialised concepts to the vocabularies. The aim of schema.org is not to provide a top-level ontology, rather it puts in place core schemas appropriate for many common situations and that can also be extended to describe things in more detail.

Google knowledge graph uses structured data from Freebase to enrich search results. For example a search for the Beatles could include data about the band and its membership. In the snapshot of Figure 3.22 this can be seen in what is
called a disambiguation pane to the right of the search results. This additional information could help the user to disambiguate between alternative meanings of their search terms. Google knowledge graph can also allow users to access directly related web pages that would otherwise be one or more navigation steps away from their search results. For example, the search for a Beatles album could provide links giving direct access to tracks contained on the album.

Figure 22. Google search for The Beatles showing a disambiguation pane.

Bing is now providing similar functionality to Google Knowledge Graph but built on the Trinity graph engine. A Bing search for “leopard” would produce structured data and disambiguation as shown in Figure 23.

Figure 23: Bing search for Leopard showing disambiguation.

The above examples use data graphs that connect entities to enrich search results. The Open Graph Protocol, originally developed by Facebook, can be used to define a social graph between people and between people and objects. The Open Graph Protocol can used to express friend relationships between people and also relationships between people and things that they like: music they listen to, books they have read, films they have watched. These links between an object and person are expressed by clicking Facebook “like” buttons that can be added
by publishers to websites outside the Facebook's domain. RDFa embedded in the page provides a formal description of the “liked” item. The Open Graph Protocol supports the description of several domains including music, video, articles, books, websites and user profiles.

Figure 24: Example Open Graph relationships (from [22]).

The Open Graph Protocol can be used to express different types of actions for different types of content. For example, a user can express that they want to watch, have watched or give a rating for a movie of TV programme. For a game, a user may record an achievement or high score.

This social graph of people and objects can then be used in Facebook Graph Search. This can be used to search not only for objects, but for objects liked by friends that have other properties such as living in a particular location.

Figure 25: Facebook Graph Search.

1 3.9 Tools for Providing Linked Data

In this section we will look at some of the tools that can assist with the creation and interlinking of linked data, introduced in sections 3.3 and 3.4.
• Extracting data from spreadsheets: OpenRefine.
• Extracting data from RDBMS: R2RML.
• Extracting data from text: OpenCalais, DBpedia Spotlight, Zemanta, GATE.
• Interlinking datasets: Silk.

1.1 3.9.1 Extracting data from Spreadsheets: OpenRefine

First, we will look at how RDF data can be created from tabular data such as that found in spreadsheets. This relates to the part of the architecture shown in Figure 3.26. Tabular data can be represented in a number of common formats. CSV (Comma Separated Values) and TSV (Tab Separated Values) are two common plain text formats for representing tables. Tables can also be represented in HTML and in spreadsheet applications. Tabular data can also be represented in JSON (Javascript Object Notation), originally developed for use with the Javascript language and now a common data interchange format on the Web.

The transformation of tabular data to an RDF dataset will involve mapping items mentioned in tables to existing vocabularies, interlinking to entities for other datasets and to some extent data cleansing where alternative names or mistyped names for items mentioned in the tables need to be handled.

![Figure 26: Integrating chart data.](image)

Below we can see an example of data represented in CSV format. This shows sales data for a number of music artists. Each row is divided into two cells by the comma. So, for example, the first row tells us that The Beatles have sold 50 million records. Elvis Presley is not far behind with 203.3 million records.

The Beatles, 250 million
Elvis Presley, 203.3 million
Michael Jackson, 157.4 million
Madonna, 160.1 million
Led Zeppelin, 135.5 million
Queen, 90.5 million

Below we can see the first row of sales data represented in JSON format. Here the rank order of the artists based on sales is made explicit as an additional column in the data. The cells in the data also have labels, reflecting the column labels you might find in a HTML table, and which are also implicit in the CSV format.

```json
{
  "artist": {
    "class": "artist",
    "name": "The Beatles"
  },
  "rank": 1,
  "value": 250 million
},
```

Finally, in Figure 27 we see sales data represented as a HTML table. Here we have explicit column labels and also additional information related to the active period and first release of each artist.

![Figure 27: List of best-selling music artists (from [23]).](image)

We will now look at how OpenRefine [24] can be used to translate tabular formats to RDF. The OpenRefine tool was originally developed by the company Metaweb as a way of extracting data for Freebase, a vast collaborative knowledgebase of structured data. This tool later became GoogleRefine and was then renamed OpenRefine when released as an open source project. When using OpenRefine,
the first step is to create a project and import tabular data in a format such as Microsoft Excel or CSV.

Figure 28: Importing data to OpenRefine.

Your browser does not support the video tag.

Movie 5: Screencast of OpenRefine.

As illustrated in Figure 3.29, OpenRefine assists us in transforming a tabular format such as CSV to RDF data. A number of processes are involved in this transformation. First, we can see that the serialisation has changed from comma-separated rows of data to RDF data in Turtle format. Second, the artists listed in the first column have been transformed from text strings to MusicBrainz URIs. Third, the sales figures have been transformed to a number with an integer data type. Finally, we have a relation (totalSales) linking the artist to the number. We will now look step-by-step at how this transformation is carried out.
The first step involves defining the rows and columns of the dataset. This can involve deleting columns not required in the RDF data and splitting columns based on specified conditions. To help with this process OpenRefine provides a powerful expression language called OpenRefine Expression Language. This has a number of functions for dealing with different types of data such as Booleans, strings, and mathematical expressions. The OpenRefine Expression Language is also still often known by the acronym GREL. This dates back to when it was known as the Google Refine Expression Language.

In the example of Figure 3.30, we used GREL to split the word “million” from the number in Column 2 to create Columns 22 and 23. We then multiply Column 22 by the number 1,000,000 to create the Total Sales column.
We then need to map the artists listed in Column 1 to MusicBrainz URIs. For this we can use the RDF Refine plugin developed by DERI (www.deri.ie). The process of mapping between multiple representations of the same thing (in this case artists represented by a string and a MusicBrainz URI) is known as *entity reconciliation*. Entity reconciliation can be carried out against the SPARQL endpoint of an RDF dataset. Textual names for things (such as The Beatles) are matched against the text labels associated with the entities in the dataset. In the figure the artist names have been reconciled with the MusicBrainz URIs listed in a column headed musicbrainz-id.

**Figure 31: Entity reconciliation in OpenRefine.**
The data is then transformed into RDF triples. In the example of Figure 3.32, we have specified that a triple should be generated for each row in which the MusicBrainz URI is connected to the Total Sales data by the totalSales property. The RDF Preview tab shows what the first 10 rows of data will look like as RDF triples.

![Figure 32: Previewing RDF triples in OpenRefine.](image)

### 1.2 3.9.2 Extracting data from relational databases: R2RML

For data stored in multiple tables in a relational database we need a more expressive way of defining mappings to an RDF dataset. R2RML (Relational Database to RDF Mapping Language) [4] can be used to express mappings between a relational database and RDF that can then be handled by an R2RML engine. R2RML can be used to publish RDF from relational databases in two ways. First, the data could be transformed in batch as an RDF dump. This dump could then be loaded into a RDF triplestore. The endpoint of the triplestore could then be used to run SPARQL queries against the RDF dataset. Second, the R2RML engine could be used to translate SPARQL queries on-the-fly into SQL queries that can be run against the relational database.
In 2012, the W3C made two recommendations for mapping between relational databases and RDF [25]. The first recommendation defines a direct mapping between the database and RDF. This does not allow for vocabulary mapping or interlinking, just the publishing of database content in an RDF format without any additional transformation. The direct mapping recommendation is not relevant here as we also wish to map items in the database (for example music artists) to existing URIs even though those URIs are not included in the database itself.

The second recommendation is R2RML that provides a means of assigning entities in the database to classes and mapping those entities in subject-predicate-object triples. It also allows the construction of new URIs for entities and interlinking them with the rest of the RDF graph.

Figure 3.34 shows the core database tables and relationships in the MusicBrainz Next Generation Schema (NGS) that was released in 2011. In the diagram, the Primary Key (PK) of a table indicates that each entry in that column can be used to uniquely reference a row. A Foreign Key (FK) uniquely identifies a row in another table. MusicBrainz NGS provides a more complex way of modelling musical releases. For example, before NGS it was not possible to relate together multiple releases of the same album at different times and in different territories.

The NGS defines an Artist Credit that can be used to model variations in artist name. This can describe multiple names for an individual and different names for various groups of artists. For example, the song “Get Back” is credited to “The Beatles with Billy Preston” rather than “The Beatles”. This would be difficult to represent in MusicBrainz without NGS.

Another major change in MusicBrainz NGS is how musical releases are modelled. A Release Group is an abstract “album” entity. A Release is a particular product you can purchase. A Release has a release date in a particular county and on a particular label. Many releases can belong to the same Release Group. A Release
may have more than one Medium (such as MP3, CD or vinyl). On each Medium, the Release will have a tracklist comprising a number of tracks. Each track has a Recording that is a unique audio version of the track. This could be used, for example, to distinguish between the single and album versions of a track. Artist Credit can be assigned to each individual track as well as the Recording, Release and Release Group. Artist Credit can also be assigned to a Work, which represents the composed piece of music as distinct from its many recordings.

*Figure 34: Relational Schema for the Music Database.*

In Figure 3.35 we can see a few core classes in the Music Ontology to which we can map when generating RDF data from data represented in the MusicBrainz NGS Relational Database. The Music Ontology models a MusicArtist as composing a Composition, which is then produced as a MusicalWork. A Performance of a MusicalWork can be recorded as a Signal that can be Produced from Recordings of that Performance.

*Figure 35: The Music Ontology (from [26]).*

Mapping a database table to a class in an ontology is relatively straightforward. Here we will map the Artist table in MusicBrainz NGS to the MusicArtist class in the Music Ontology. Mappings in R2RML are mostly specified as instances of what are referred to *TriplesMaps*. The TriplesMap specified below has the
identifier lb:Artist. The logicalTable is the source data table from which the triples are derived. In this case the logicalTable is the table named “artist” in the relational database. As we shall see in the next example, the logicalTable can also be the result of a SQL query across a number of tables rather than a single table in the database.

The subjectMap defines the subject of the triple. The subject is constructed as a MusicBrainz URI with the entry from the gid column of the Artist table being inserted between http://musicbrainz.org/artist/ and the # symbol. The specified predicate is mo:musicbrainz_guid which links a MusicBrainz URI to its ID in the form of a string. The object of the triple is also the entry from the gid column but represented as a string.

```
rb:Artist a rr:TriplesMap ;
rr:logicalTable [rr:tableName “artist”] ;
rr:subjectMap
[rr:class mo:MusicArtist ;
rr:template
"http://musicbrainz.org/artist/\protect\char"007B\relaxgid\protect\char"007D\relax#="";]
rr:predicateObjectMap
[rr:predicate mo:musicbrainz_guid ;
rr:objectMap [rr:column “gid” ;
rr:datatype xsd:string]] .
```

Database columns can also be mapped to properties. In the example below we supply a name property to each of the MusicBrainz URIs generated in the previous example. In this case the logicalTable used in the TriplesMap is an SQL query that returns a table of results. This query joins two tables to link the gid of an artist to the artist’s name. The subject of the triple is the same as the subject specified using the TriplesMap from the previous example. The predicate is foaf:name. The object is the name column from the logicalTable

```
lb:artist_name a rr:TriplesMap ;
rr:logicalTable [rr:sqlQuery
"SELECT artist.gid, artist_name.name
FROM artist
INNER JOIN artist_name ON artist.name = artist_name.id";
rr:subjectMap lb:sm_artist ;
rr:predicateObjectMap
[rr:predicate foaf:name ;
rr:objectMap [rr:column “name”]] .
```

MusicBrainz Next Generation Schema (NSG) also provides Advanced Relationships as a way of representing various inter-relationships between key MusicBrainz entities such as Artist, Release Group and Track. The table l_artist_artist is
used to specify relationships between artists. Each pairing of artists would be represented as a row in the `l_artist_artist` table. Each pairing of artists refers to a Link. One link would be `member_of` that would specify a relation between an artist and a band of which they were a member.

**Figure 36: NGS Advanced Relations.**

The R2RML triplesmap below shows how we would specify that an artist is a member of a band. Here the `logicalTable` is the result of a complex query that associates artists with a band. The Music Ontology `member_of` predicate is used to associate an artist with a MusicBrainz URI identifying the band.

```plaintext
lb:artist_member a rr:TriplesMap ;
rr:logicalTable rr:sqlQuery "SELECT a1.gid, a2.gid AS band 
FROM artist a1 
INNER JOIN l_artist_artist ON a1.id = l_artist_artist.entity0 
INNER JOIN link ON l_artist_artist.link = link.id 
INNER JOIN link_type ON link_type = link_type.id 
INNER JOIN artist a2 on l_artist_artist.entity1 = a2.id 
WHERE link_type.gid='5be4c609-9afa-4ea0-910b-12ffb71e3821' 
AND link.ended=FALSE" ;
rr:subjectMap lb:sm_artist ;
rr:predicateObjectMap 
[rr:predicate m:member_of ;
rr:objectMap rr:template "http://musicbrainz.org/artist/\protect\char'007B\relaxband\protect\char'007D\relax#_" ;
rr:termType rr:IRI] .
```
1.3 3.9.3 Extracting data from text: DBpedia Spotlight, Zemanta and OBIE

The previous tools worked on data that was already in some tabular or relational structure. Work carried out by the tools largely involved transforming this existing structure to a triple structure as well as some mapping and interlinking. Text is more open and ambiguous and involves more than transformation. As we shall see later, text extraction is only correct to some level of precision and recall.

OpenCalais [27] can be used to automatically identify entities from text. OpenCalais uses natural language processing and machine learning techniques to identify entities, facts and events. OpenCalais is difficult to customise and has variable domain-specific coverage.

DBpedia Spotlight [28] can be used to identify named entities in text and associate these with DBpedia URIs. In the snapshot of Figure 3.38, recognised entities in the submitted text have been hyperlinked to their DBpedia URIs. DBpedia Spotlight is not easy to customise or extend and is currently only available in English.

Zemanta [29] is another general-purpose semantic annotation tool. Zemanta is used by bloggers and other content publishers to find links to relevant articles and media. Best results require bespoke customization.
Figure 39: Zemanta.

GATE (General Architecture for Text Engineering) [30] is an open-source framework for text engineering. GATE started in 1996 and has a large developer community. GATE can be more readily customized for text annotation in different domains and for different purposes. GATE is used worldwide to build bespoke solutions by organisations including the Press Association and National Archive. Information extraction is supported in many languages. GATE can also parse text as well as recognise entities and can therefore be used to identify entities depending on their function in the sentence. For example, GATE could be used to extract an entity only when used as the noun phrase, rather than the verb phrase, of the sentence.

LODIE [31] is an application built with GATE using DBpedia. LODIE uses Wikipedia anchor texts, disambiguation pages and redirect pages to help find alternative versions of things.

Figure 40: LODIE.

Precision and recall measures can be used to compare text annotation tools. Precision and recall are both values from 0 to 1. Precision indicates what proportion of annotations are correct. Recall indicates what proportion of possible correct annotations in the text are identified by the tool.

Figure 3.41 shows precision and recall figures for these tools. Precision and recall figures are shown in pairs, precision on the left, recall on the right separated by a slash. Precision and recall are shown for types of entity (person, location and organisation) as well as in total. We see that DBpedia Spotlight has relatively
good precision but relatively low recall, identifying 39% of entities in the text. Zemanta has similar precision but higher recall. LODIE has the highest recall but lower precision.

Good results can also be achieved by combining the methods. Only annotating entities suggested by both Zemanta and LODIE (i.e. the intersection of Zemanta and LODIE) gives very high precision. The union of Zemanta and LODIE gives high recall but lower precision.

![Figure 41: Comparison of DBpedia Spotlight, Zemanta and LODIE.](image)

An alternative to the generic services provided by DBpedia Spotlight and Zemanta is to build a GATE processing pipeline specifically for your domain. For this we take an RDF dataset and use this to produce what is called a GATE Gazetteer, which is a list of entities in a domain and associated text labels used to refer to those entities. We can produce a gazetteer using the RDF data produced from the R2RML transformation of the MusicBrainz NGS Relational Database (see section 3.7.2). A SPARQL endpoint to this data can be used to populate a custom gazetteer for the music domain. For example, the query in Figure 3.43 returns solo artists and music groups and their foaf:name. It also returns albums (represented by the SignalGroup class in the music ontology) and their dc:title. This provides a vocabulary of artists and albums with associated labels that can be used in the gazetteer.

![GATE Gazetteeer](image)
Figure 42: Producing a GATE Gazeteer.

A GATE pipeline can be run locally or uploaded to the GATE cloud. Once set up, text can be submitted and then annotated using the MusicBrainz data. The annotated text can then be output in a format such as RDFa.

Figure 43: GATE Cloud.

1.4 3.9.4 Interlinking datasets: SILK

As mentioned in section 3.4, manually interlinking large datasets is not feasible. SILK [32] is a tool that has been developed to support the interlinking of datasets. In our case we may wish to define links between artists mentioned in MusicBrainz and the same entities in DBpedia. This process fits specifically in the interlinking phase of the diagram of Figure 44.

Figure 44: Interlinking datasets with SILK.

SILK is an open source tool for discovering RDF links between data items within
differ free 1 3.9 Tools for Providing Linked Data 21 / 26
different Linked Data sources. The Silk Link Specification Language (Silk-LSL) is
tused to define rules for linking entities from two different datasets. For example, a
rule may express that if two entities belong to specified classes and have matching
labels then they should be linked by a certain property. This property could be
owl:sameAs or some other property such as skos:closeMatch (see section 3.4).

SILK can run in different variations. It can be run locally on a single machine,
on a server or distributed across a cluster. The SILK workflow is shown in Figure
3.44. The first step is to select the two datasets. Generally, one dataset would
be your own and another including some of the same entities but named with
different URIs. In the second step, we specify the two datasets by either loading
an RDF dump or pointing to a SPARQL endpoint. We also specify the types of
etentities to be linked. In the third step, we express the linkage rules in Silk-LSL.
The discovered links can then be published as a linkset (see section 3.5.1) with
your dataset.

Figure 45: SILK workflow based on [33].
The rules for comparing two entities can consider not only the two entities them-
selves but also additional data items found in the graph around each of those
entities. For example, a rule may compare the rdfs:label of one entity with the
foaf:name of another. The paths from the compared entities to these additional
data items are specified as RDF paths. Different transformations can be also be
performed on the compared data items. For example, if they are strings (such as
labels or names) then they may be both transformed into lower case to prevent
the use of lower and upper case leading to a mismatch. The linkage rules also de-
define the comparators used to compute the similarity of the two data items. When
comparing two strings, an exact match may be required. Alternatively, similarly
may be computed as a Levenshtein edit distance. This indicates the number of
single character changes that would need to be made in order to turn one string
into the other. This provides a way of matching data items that may contain
typos. Similarity metrics can be used for other data types such as dates. Finally,
aggregations can be computed from data items associated with the entity. For
example, two potentially matching albums could be compared in terms of the
number of track that they contain. If the number of tracks is equal then this
could be further evidence that these two entities refer to the same album.

The SILK Workbench is a web application built on top of SILK that can be used
to create projects and manage the creation of links between two RDF datasets.
The SILK Workbench has a graphical editor that can be used to create linkage
rules. Support is also provided for the automatic learning of linkage rules. Figure
3.46 shows a snapshot of the SILK Workbench. A new project has been created
with the name “MyLinkedMusic”. Two datasets have been added, labelled as DBpedia and MyMusicBrainz. The sections below this are concerned with the specification of linkage rules and the location and format of the output.

![Figure 46: Overview of a Silk Workbench project](image)

Figure 46: Overview of a Silk Workbench project

Figure 47 shows how the graphical editor can be used to specify linkage rules. In this example the foaf:name of the MusicBrainz entity and the rdfs:label of the DBpedia entity are both transformed to lower case and then compared in terms of their Levenshtein edit distance.

![Figure 47: Adding a linkage rule in the SILK workbench.](image)

Figure 47: Adding a linkage rule in the SILK workbench.

The linkage rules can then be used to generate a set of links as shown in Figure 48. Each of these links between a MusicBrainz and DBpedia identifier has a confidence score. The larger the Levenshtein edit distance between the foaf:name and rdfs:label, the lower the confidence. All of the examples listed have a confidence score of 100% indicating a zero edit distance between the two literals. Confidence can be accumulated from a number of sources. When comparing music groups, this could include other data such as their membership and formation date.
Figure 48: Generating links with the SILK workbench.

The SILK Workbench also provides an interface for examining automatically learned rules. These suggested rules can then be added to the set of linkage rules or rejected.

Figure 49: Rule learning with the SILK workbench.
2 Text and image sources, contributors, and licenses

2.1 Text

- Course: How to use Linked Data/Providing Linked Data/Linked Data Catalogs and Tools for Providing Linked Data Source: https://en.wikitolearn.org/Course%3AHow_to_use_Linked_Data/Providing_Linked_Data/Linked_Data_Catalogs_and_Tools_for_Providing_Linked_Data?oldid=10364 Contributors: Irene, WikiToBot, M.bona, Move page script and Anonymous: 1

2.2 Images


2.3 Content license

- [Project:Copyright Creative Commons Attribution Share Alike 3.0 & GNU FDL]
- Creative Commons Attribution-Share Alike 3.0