Automatic Interlinking of Music Datasets on the Semantic Web

Yves Raimond, Christopher Sutton and Mark Sandler
Centre for Digital Music
Queen Mary, University of London
{yves.raimond,chris.sutton,mark.sandler}@elec.qmul.ac.uk

ABSTRACT

In this paper, we describe current efforts towards interlinking music-related datasets on the Web. We first explain some initial interlinking experiences, and the poor results obtained by taking a naïve approach. We then detail a particular interlinking algorithm, taking into account both the similarities of web resources and of their neighbours. We detail the application of this algorithm in two contexts: to link a Creative Commons music dataset to an editorial one, and to link a personal music collection to corresponding web identifiers. The latter provides a user with personally meaningful entry points for exploring the web of data, and we conclude by describing some concrete tools built to generate and use such links.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Linked Data

Keywords

Semantic-Web,Linked Data,Music

1. INTRODUCTION

The Linking Open Data community project [3] aims at publishing and interlinking open datasets by following simple rules [2] for linking data. The publication step can to some extent be automated, using tools such as D2R or OpenLink Virtuoso (which both allow relational databases to be published as linked data) or P2R (which allows SWI-Prolog knowledge bases to be published in the same way). All these tools handle declarative mappings from a given data structure to corresponding web resources and associated RDF descriptions. Once this publication is achieved, we still have to create links to other datasets, in order for a user agent to navigate from one to another. A typical example of such interlinking would be the following one, where a music band in a Creative Commons label is linked to its location in the Geonames dataset, and to the corresponding resource in an editorial database[1]:

```
<http://dbtune.org/jamendo/artist/5>
  foaf:based_near <http://sws.geonames.org/2991627/> ;
  owl:sameAs <http://zitgist.com/music/artist/
              0781a3f3-645c-45d1-a84f-76b4e4decf6d>.
```

Then, you may access the actual audio content from the Creative Commons label, some extra information such as the birth dates of the members of this band from the editorial dataset, and detailed geographic information (latitude, longitude, hierarchy of geographical features) from the Geonames dataset.

For small datasets published manually (such as an individual's FOAF file), it is possible to create such links manually. However, doing so for large datasets is impractical: we need a way to automatically detect the overlapping parts of heterogeneous datasets. In this paper, we detail a few algorithms that have been developed, implemented and practically deployed to interlink different music-related datasets. We mainly focus on the most sophisticated one, applicable in a Linked Data context, and taking into account not only the similarities of single resources but also the similarities of their neighbours. We evaluate how this algorithm performs when applied to link a real-world Creative Commons dataset to an editorial one. We also show how a personal music collection can be treated as one such dataset, enabling a user to benefit from the growing body of knowledge on the Semantic Web in a personally meaningful way.

We define the mapping problem as follows. We consider two RDF datasets D_1 and D_2, respectively describing a number of web resources r_i and s_i. We consider the problem of matching resources—finding resources s_y in our target dataset, D_2, which identify the same object as a resource r_x in our seed dataset, D_1. For example, we want to find s_y = `http://zitgist.com/music/artist/0781a3f3-645c-45d1-a84f-76b4e4decf6d` for r_x = `http://dbtune.org/jamendo/artist/5`. All mapping problems in our context can be reduced to this one, even *literal expansion* where a resource r_x is linked to a literal l and we are looking for a resource s_y corresponding to l—for example, expanding "Moselle, France" into `http://sws.geonames.org/2991627/`. In these cases,

[1]We use the Turtle notation throughout the paper, with the namespaces defined in § 8

LDOW'08, April 28, 2008, Beijing, China

a simple transformation (creating a blank node between r_x and l) is enough to return ourselves to a resource matching problem.

2. NAIVE INTERLINKING

In this section, we describe some naïve approaches to tackling this resource matching problem, and identify their failings.

2.1 Simple literal lookups

Most datasets provide a literal search facility, either through a dedicated Web service (eg. Geonames, Musicbrainz), or through a SPARQL end-point on which we can use filters on literals, or in some cases built-in literal matching functionality. So one solution to link resources from our two datasets would be to first issue the following query on D_1:

```
SELECT ?l
WHERE { <r> ?p ?l } FILTER (isLiteral(?l))
```

and use the bindings of $?l$ to issue a literal search on D_2. We can then try to map the resulting resources to r.

We used this kind of approach to link the Jamendo dataset to the Geonames one[2]. Jamendo provides information about the location of artists, in the form of a literal string (such as "Moselle, France"). We use this to query the Geonames web service, and get back the corresponding Geonames resource. In practice, we found that the literal strings provided by Jamendo are specific enough for this approach to work well. In the two instances when more than one candidate was returned for a location string, no link was created.

2.2 Extended literal lookups

Although this simple approach can be suitable for interlinking particular datasets in cases where a literal string reliably provides sufficient disambiguation it is unlikely to discriminate suitably in most cases. For example, when trying to apply it to link musical works in the BBC John Peel Sessions to resources in the DBpedia dataset [1], we come across the literal "Violet"[3]. The actual song which the algorithm should link to is just one of the sixteen results of the corresponding literal lookup.

One solution is to add constraints on the resulting resources. For example, by using the DBpedia links to Yago [10], we can restrict our resources to be of a specific Yago type, or we can restrict it to be linked to a particular *infobox* URI (specifying a template for structured data on the corresponding Wikipedia page). This leads us to the following SPARQL query:

```
PREFIX p: <http://dbpedia.org/property/>
SELECT ?r
WHERE
{ ?r ?p "Violet"@en.
  {
    {?r a <http://dbpedia.org/class/yago/Song107048000>}
    UNION
    {?r p:wikiPageUsesTemplate
      <http://dbpedia.org/resource/Template:single_infobox>}
  }
}
```

[2]All links to mentioned datasets are available in § 7

[3]For the resource `http://dbtune.org/bbc/peel/work/1498`

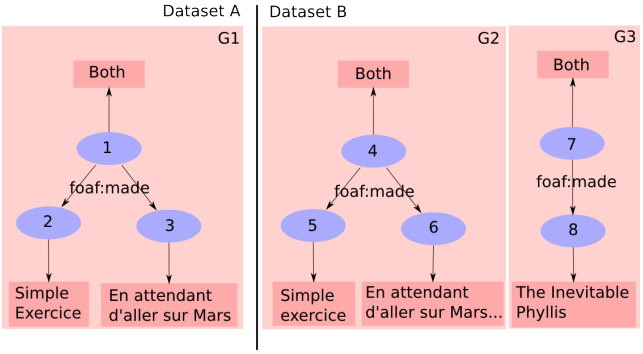

Figure 1: Example datasets—we here try to match elements from A and B

This approach was used to link the musical works and the artists in the BBC John Peel sessions dataset to corresponding resources in DBpedia. Constraints on the target resources were manually defined, and queries taking into account the seed literal and these constraints were issued to the DBpedia SPARQL end-point.

However, even with restrictions on the nature of target resources, a literal may not be discriminating enough. For example, the resource `http://dbtune.org/jamendo/artist/5` is linked to the literal "Both". Searching the Musicbrainz dataset for "Both" while restricting ourselves to artists gives us two results. Likewise, when looking for the Nirvana version of the song "Love Buzz" in the DBpedia dataset, we have to disambiguate between the original song and the Nirvana cover. There is therefore a need for a more sophisticated algorithm, capable of handling such disambiguation.

3. GRAPH MATCHING

An intuitive approach to disambiguate two artists with the same name would be to check the titles of their releases, and see if they match the titles of the releases in our seed dataset. If by any chance they have releases with the same title, we can check their track titles, and disambiguate using them, and so on. In this section, we develop this idea and give a formal specification of such an algorithm.

3.1 Offline graph matching

Consider the two datasets illustrated in fig. 1, with our seed dataset on the left containing a single artist with the name "Both", and the target dataset on the right containing two artists named "Both". We model the two datasets as graphs, with each edge represented as a triple (s, p, o).

As a first step, we compute initial similarity values between all pairs of resources (s_1, s_2) and (o_1, o_2) such that $(s_1, p, o_1) \in D_1$ and $(s_2, p, o_2) \in D_2$. Such similarity values might be calculated by a string similarity algorithm (such as the ones described in section 2 of [12]) comparing literals directly at-

Table 1: Initial similarity map

Resource 1	Resource 2	Initial Similarity
1	4	1
1	7	1
2	5	0.9
3	6	0.8
2	6	0.1
3	5	0.1
2	8	0.1
3	8	0.1

Table 2: Possible graph mappings—$(x, y) \equiv \{x$ *owl*:*sameAs* $y\}$**, and associated measures**

Graphs		Mapping	Measures
G1	G2	$M_{G1:G2}a = \{(1,4),(2,5),(3,6)\}$	0.9
G1	G2	$M_{G1:G2}b = \{(1,4),(2,6),(3,5)\}$	0.4
G1	G3	$M_{G1:G3}a = \{(1,7),(2,8)\}$	0.55
G1	G3	$M_{G1:G3}b = \{(1,7),(3,8)\}$	0.55

tached to these resources. In our example, this produces the results in table 1.

Next, we construct a *graph similarity* measure. In our example, we consider the possible graph mappings in table 2. Then, we associate a measure with such mappings: we sum the similarity values s associated with each pair (x, y) and we normalise it by the number of pairs in the mapping. In our example, the resulting measures are in table 2.

Finally, we choose the mapping whose similarity measure is the highest, optionally thresholding to avoid making mappings between graphs which are too dissimilar. In our example, we choose $M_{G1:G2}a$.

3.2 Linked Data context

Now, we apply this algorithm in a Linked Data [2] context, where we discover our graphs as we go: the main idea being that we update the graph mappings and their measures as we update our local Semantic Web cache. This is necessary because in general the size of the datasets D_1 and D_2 will be prohibitive; if not for loading both datasets, then for computing all the possible graph mappings between them.

Our starting point is now a single URI r in a dataset D_1. We try to find the corresponding s in a dataset D_2, as well as mappings of resources in the neighbourhood of r. We illustrate this using the same example as the one in § 3.1: we want to map the URI `http://dbtune.org/jamendo/artist/5` to the corresponding Musicbrainz URI. As part of the same process we wish to map corresponding albums and tracks for this artist.

In the following, we will use Named Graphs [5], in order to track the provenance of a particular graph, and we let G_x denote the graph retrieved when dereferencing x.

The first thing we do is to retrieve G_r, and extract a suitable label l for r (using *dc*:*title* or $foaf$:*name* properties, etc). Now, we need to access some potential candidates for s. To do that, we use the same approach as described in § 2.2. We issue a query (through a SPARQL end-point or custom web service) to D_2 which involves l and constraints over what we are looking for. This gives us a list of resources.

For each s_k in this list, we access G_{s_k}. Now, for all possible graph mappings $M_{G_r:G_{s_k},i}$, we compute a measure as defined in § 3.1. If we can make a clear decision now (ie. there is just one measure above our decision threshold), we terminate and choose the corresponding graph mapping. If not, we look for object properties p such that $(r, p, o) \in G_r$ and $(s_k, p, o') \in G_{s_k}$, and we obtain G_o and G'_o. [4] Then, we update our possible graph mappings and the associated measures. We iterate this process until we can make a decision (we have one unique mapping with measure above the threshold), or until we can't go any further (no unexplored object properties). Practically, we also limit the maximum number of iterations the algorithm may perform.

In our example, we first dereference `http://dbtune.org/jamendo/artist/5`. We get access to the following facts: our URI is identifying a musical band, it is called "Both", and it *made*[5] two things: `http://dbtune.org/jamendo/record/174` and `http://dbtune.org/jamendo/record/33`. We now look for an artist named "Both" in the Musicbrainz dataset, through the Musicbrainz web service[6]. This gives us back two URIs: `http://zitgist.com/music/artist/5f9f2dfb-76f0-4872-ad7d-f9d84a908cb5` and `http://zitgist.com/music/artist/0781a3f3-645c-45d1-a84f-76b4e4decf6d`. We dereference them: the first one identifies an artist named "Both" which made two things, and the second one also an artist named "Both" which made one thing.

We now consider two possible graph mappings (corresponding to the two potential matches of our artist resource), with two measures, both equal to 1. We continue looking for further clues, as there is not yet any way to disambiguate between the two. We take the object property occurring in our three graphs, $foaf$:*made*, and dereference all the objects of this property that we currently know about. Our starting resource made two records, named "Simple Exercice" and "En attendant d'aller sur Mars". The first matching artist in the Musicbrainz dataset made two records, named "Simple exercice" and "En attendant d'aller sur Mars...". The second matching artist made one record, named "The Inevitable Phyllis". We now update the possible graph mappings, and reach the results in table 2. Now, we have one mapping identifiably better than the others, with graph similarity measure 0.9. We choose it, and hence derive the following statements:

```
<http://dbtune.org/jamendo/artist/5> owl:sameAs
          <http://zitgist.com/music/artist/
            0781a3f3-645c-45d1-a84f-76b4e4decf6d>.
<http://dbtune.org/jamendo/record/174> owl:sameAs
          <http://zitgist.com/music/record/
            3042765f-67ba-49ef-ab28-45805fabef4a>.
<http://dbtune.org/jamendo/record/33> owl:sameAs
          <http://zitgist.com/music/record/
            fade0242-e1f0-457b-99de-d9fe0c8cbd57>.
```

[4] We could additionally consider triples of the form (o, p, r) and (o', p, s_k).

[5] Captured through the $foaf$:*made* predicate

[6] See `http://wiki.musicbrainz.org/XMLWebService`

Having chosen a mapping we could go further, to also derive such statements for the tracks in these two albums. Another possible extension of this algorithm is to perform literal lookups in D_2 at each step (therefore providing new possible graph mappings each time). This helps ensure we still find the correct mapping in the case that our initial literal lookup does not include the correct resource among its results. For example, the correct target artist might be listed as having a different name in D_2 as in D_1, such that they do not feature in the results of our initial literal lookup. However, we might have some clues about who this artist is from the names of the albums they produced, and so performing additional literal lookups (on the album titles) may allow us to find the correct artist and hence the correct mapping. Such a variant of this algorithm is implemented in the GNAT software described in § 4.2.

3.3 Algorithm definition

The algorithm described above can be expressed in the following pseudo-code. We assume the existence of a function $string\_similarity(x, y)$ and define the following additional functions:

function $similarity(x, y)$:
  Extract a suitable label l_x for x in G_x
  Extract a suitable label l_y for y in G_y
  Return $string\_similarity(l_x, l_y)$

function $lookup(x)$:
  Extract a suitable label l_x for x in G_x
  Perform a search for l_x on D_2
  Return the set of resources retrieved from the search

function $measure(M)$:
  Foreach $(r_i, r_j) \in M$
    $sim_{i,j} = similarity(r_i, r_j)$
  Return $\sum_{i,j} sim_{i,j}$

function $combinations(O_1, O_2)$:
  Return all possible combinations of
  elements of O_1 and elements of O_2
  e.g. $combinations(\{1, 2\}, \{3, 4\}) =$
    $\{\{(1, 3), (2, 4)\}, \{(1, 4), (2, 3)\}\}$

Our starting point is a URI r in D_1 and a decision threshold *threshold*. Our mapping pseudo-code is then defined as:

Foreach $s_k \in lookup(r)$
  $M_k = \{(r, s_k)\}$
  $measure_k = measure(M_k)$
  $sim_k = measure_k / |M_k|$
If $sim_k > threshold$ for exactly one k, **return** M_k
Else, $Mappings$ is the list of all M_k, and **return** $propagate(Mappings)$

function $propagate(Mappings)$:
  Foreach $M_k \in Mappings$:
    $measure_k = measure(M_k)$
    Foreach p *s.t.* $(\exists (r, r') \in M_k$, $\exists (r, p, o) \in G_r$, $\exists (r', p, o') \in G_{r'}$ and $\forall Map \in Mappings, (o, o') \notin Map)$:
      Foreach $(r, r') \in M_k$:
        $O_{k,r,p}$ is the list of all o such that $(r, p, o) \in G_r$
        $O'_{k,r,p}$ is the list of all o such that $(r', p, o) \in G_{r'}$
      Foreach $Objmap_{k,i} \in \bigcup_r combinations(O_{k,r,p}, O'_{k,r,p})$:
        $sim_{k,i} = (measure_k + measure(Objmap_{k,i}))/(|M_k| + |Objmap_{k,i}|)$
  If no $sim_{k,i}$, **fail**
  If $sim_{k,i} > threshold$ for exactly one $\{k, i\}$ pair, **return** $append(M_k, Objmap_{k,i})$
  Else, $NewMappings$ is the list of all $append(M_k, Objmap_{k,i})$, and **return** $propagate(NewMappings)$ (if the maximum number of recursions is not reached, otherwise **fail**)

Now, we apply this pseudocode to our earlier "Both" example ($r = 1 =$ `http://dbtune.org/jamendo/artist/5`), with a *threshold* of 0.8. This makes us go through the following steps:

$lookup(r) = \{4, 7\}$
$M_1 = \{(1, 4)\}$, $sim_1 = 1$
$M_2 = \{(1, 7)\}$, $sim_2 = 1$
$Mappings = \{\{(1, 4)\}, \{(1, 7)\}\}$

$propagate(\{\{(1, 4)\}, \{(1, 7)\}\})$
  $k = 1$, $p = foaf{:}made$
    $O_{1,1,foaf:made} = \{2, 3\}$, $O'_{1,4,foaf:made} = \{5, 6\}$
    $Objmap_{1,1} = \{(2, 5), (3, 6)\}$, $Objmap_{1,2} = \{(2, 6), (3, 5)\}$
    $sim_{1,1} = 0.9$, $sim_{1,2} = 0.4$
  $k = 2$, $p = foaf{:}made$
    $O_{2,1,foaf:made} = \{2, 3\}$, $O'_{2,7,foaf:made} = \{8\}$
    $Objmap_{2,1} = \{(2, 8)\}$, $Objmap_{2,2} = \{(3, 8)\}$
    $sim_{2,1} = 0.55$, $sim_{2,2} = 0.55$
  Now, $sim_{1,1}$ is the only $sim_{k,i}$ above the threshold, we therefore choose $\{(1, 4), (2, 5), (3, 6)\}$ as our mapping, which corresponds to the RDF code in § 3.2.

Of course, several heuristics could be added to this pseudo-code, in order to improve the scalability of the algorithm. In practice, we associate weights to properties, in order to start from the most informative one ($foaf{:}made$, for example).

4. EXPERIMENTS

In this section, we detail two experiments using this algorithm, and their respective evaluations. The first one deals with the automatic interlinking of two online music datasets. The second one deals with the linking of a personal music collection towards corresponding web identifiers.

4.1 Linking two overlapping web datasets

In this section, we focus on a concrete interlinking which has been achieved using this algorithm, between two overlapping web datasets: Jamendo and Musicbrainz. We implemented this algorithm[7] in SWI-Prolog [11], with only one lookup on the Musicbrainz end-point, at the artist level. Our algorithm derived 10944 similarity statements for artist, record, and track resources so far, which allows us to get detailed editorial information from Musicbrainz, and the actual audio content, as well as tags, from the Jamendo dataset.

We focus our evaluation on artist resources. As we perform only one lookup, at the artist level, no tracks or records can be matched if the artist is not. In order to evaluate the quality of the interlinking, we take a random sample from the Jamendo dataset: we first collect every single artist URI[8], and we randomly select 60 from among them. Then,

[7]The source code of all the software mentioned is available as part of the `motools` project: `http://sourceforge.net/projects/motools`
[8]The results of such a SPARQL query are available at

Table 3: Evaluation of the Jamendo/Musicbrainz interlinking

	Link derived	Link not derived
Correct	5	53
Incorrect	0	2

we run our mapping algorithm and manually check whether the mappings are correct. Each tested resource therefore falls into one of the following categories:

- An *owl*:*sameAs* link is derived : correct (same artist in the Jamendo and in the Musicbrainz datasets) ;
- An *owl*:*sameAs* link is derived : incorrect (different artists) ;
- No link is derived : correct (there is a corresponding artist in the Musicbrainz dataset) ;
- No link is derived : incorrect (no corresponding artists in the Musicbrainz dataset).

The results[9], in terms of how many resources fall within each of the above defined categories, are shown in table 3. In our test dataset, the disambiguation was needed in 16 cases. For example, one of the artist resources was named "Hair", which matches four resources within Musicbrainz, none of them being the same band. The first case that failed is due to an implementation mistake, failing to normalise the graph similarity measures correctly when the target graph is bigger than the seed one (in this case, the artist had two releases on Musicbrainz, and just one on Jamendo). The second case that failed is due to the fact that the Musicbrainz RDF is outdated (the artist does not exist in the RDF dump, but does exist in the Musicbrainz database).

4.2 Linking personal music collections

Personal music collections can also be a part of the web of data. The Music Ontology [9] makes the same distinction as FRBR between *manifestations* (all physical objects that bear the same characteristics, eg. a particular album) and *items* (a concrete entity, eg. my copy of the album on CD). A manifestation and a corresponding item are linked through a predicate `mo:available_as`. Therefore, given a set of audio files in a personal music collection, it is possible to keep track of the set of statements linking this collection to identifiers elsewhere in the Semantic Web which denote the corresponding manifestations. These statements provide a set of entry points to the Semantic Web, allowing access to information such as the birth date of the artists responsible for items in the collection, geographical locations of the recordings, etc.

GNAT is an implementation of automatic linking from a personal audio collection to the Musicbrainz dataset—it uses audio fingerprinting and available metadata to find corresponding dereferencable identifiers, and then outputs RDF

`http://dbtune.org:2105/sparql/?query=select%20distinct%20%3Fa%20where%20%7B%3Fa%20a%20mo%3AMusicArtist%7D`

[9]The detailed results, resource by resource, are available at `http://moustaki.org/resources/results.txt`

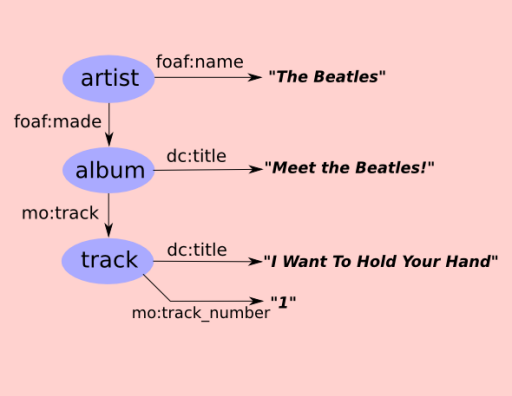

Figure 2: Constructing a graph from local music metadata

statements making the links between local audio files and the remote manifestation identifiers. The fingerprinting functionality can be useful when the metadata available is particularly poor, but since it is highly dependent on the fingerprinting service chosen, we concern ourselves here solely with the metadata-based approach.

All modern audio encodings allow for the inclusion of metadata "tags" alongside audio data in a file, such that each audio file can include the kind of editorial information contained in the data sets described above. We can therefore consider a reasonably well tagged personal music collection to be just another music data set, apply the algorithm described in § 3, and hence link each local audio file to a corresponding resource on the Semantic Web. GNAT uses the variant discussed at the end of § 3.2 which maps artists, albums and tracks, performing literal lookups at each stage. For each local audio file, a simple seed graph is constructed based on the artist, album, title and track number specified in the file's ID3 tag (see fig. 2 for an example). It proceeds as set out in § 3.3 until a single best mapping is found. After processing a directory of files in this way, GNAT outputs an RDF file providing `mo:available_as` links from URIs in the Zitgist publication of MusicBrainz data to the local audio files. For example :

```
<http://zitgist.com/music/track/
 1adfecb7-875f-4203-b3b1-8e2e643f94a2> mo:available_as
    <file:///mnt/music/Artists/Nirvana/Bleach/track5.mp3>
```

Using this algorithm rather than one of the more naïve approaches should allow GNAT to be robust to various inaccuracies in the local files' metadata. We evaluated GNAT's behaviour in the face of such problems by taking a correctly-tagged MP3 file of the Beatles track "I want to hold your hand" and artificially introducing various mistakes. The MusicBrainz dataset lists no less than 25 releases for this track by The Beatles, and dozens of artists with songs of the same name. The correct set of metadata is shown in table 4 and results are shown in table 5.

We can see that GNAT performs well in the face of inaccurate metadata. The release chosen when the album field is missing or set to a random string is arguably correct—it is

Table 5: Evaluation of GNAT's linkage on particular cases

Change made	Resulting mapping
Artist field missing	Correct
Artist set to random string	Correct
Artist set to "Beetles"	Correct
Artist set to "Al Green"	Mapped to Al Green's cover version
Album field missing	Mapped to correct track on the release
Album set to random string	"The Capitol Albums, Volume 1 (disc 1:
Album set to "Meat the Beatles"	Meet the Beatles!)"
Track set to random string	Correct
Track set to "I Wanna Hold Your Hand"	Correct

Table 4: Base (correct) metadata for GNAT evaluation

Artist	"The Beatles"
Album	"Meet The Beatles!"
Title	"I Want To Hold Your Hand"
Track Num	1

the same CD, released as part of a box set. One would hope that the release "Meet the Beatles!" would be chosen in the case of the album being misspelled.

With a practical implementation some trade-offs must be made. In the case of setting the artist to "Al Green", we have two conflicting pieces of information (artist and album) and our implementation here chooses the mapping for which the artist matches. A more sophisticated version of GNAT might consider other tracks in the current directory to establish that the track most likely comes from the Beatles release rather than Al Green's release.

Such links from a user's own files to information on the Semantic Web could be a significant step towards making data on the Web available and relevant to people, but a user agent must act on such links before they are directly useful to a human. In the next section we describe a companion tool to GNAT, designed to exploit the links GNAT produces.

4.3 Use-cases

For the links between a user's files and Semantic Web resources to be useful, an application must have some information about the resources. The GNARQL program in the `motools` project is beginning to explore some of the possibilities in this direction. The program loads in all the `owl:sameAs` links produced by GNAT, dereferences the corresponding URIs, and then aggregates additional information about those resources.

The basic mechanism for aggregating additional information is to *crawl* outwards from the given resource, dereferencing linked resources and adding their descriptions to the local RDF store. In the simplest case, this crawling can be unguided, simply following all links regardless of the properties used, and performing a breadth-first traversal of the Semantic Web from all known resources.

More sophisticated crawling strategies may lead to better aggregation for a user's purposes. For example, GNARQL can prioritise links which use properties from the Music Ontology (or other specified namespaces). This helps ensure that relevant information is prioritised over less obviously-useful information.

Information relating to resources of interest may also be retrieved based on specific rules. For example, if we know that the SBSimilarity service provides "similar track" information about tracks by appending their Musicbrainz ID to a given prefix[10] , we can specify a rule in GNARQL for generating `rdfs:seeAlso` links from known tracks to the corresponding documents in the SBSimilarity namespace. These rule-derived links will then be followed as part of the crawling strategy, and their information added to the local RDF store.

Naturally, the information held in GNARQL's store need not come solely from GNAT. In fact, any RDF data in the designated directory tree will be loaded. In our research group, this means that Chord Ontology[11] transcriptions are held alongside the MusicBrainz data retrieved from Zitgist and social tag data retrieved from various websites.

To enable applications to take advantage of this aggregated data, GNARQL provides a SPARQL endpoint. This frees end-user applications from the need to themselves maintain a database relating to the user's music collection, and allows multiple applications to benefit from a single store of aggregated information.

Based on datasets available today, some example queries such user interfaces might pass on to GNARQL include "Find tracks which are performances of works by Russian composers around the turn of the twentieth century", "Find me cover versions of rock songs in non-rock genres" or potentially (by using information linked from a user's FOAF file) "Find me gigs by the artists I play frequently which fit with my vacation schedule".

To experiment with browsing the data aggregated by GNARQL, we have developed a prototype user interface, based heavily on the `/facet` program described in [6]. This provides a web browser-based interface for exploring the aggregated information, and performing simple facet-based

[10] eg. `http://isophonics.net/music/signal/280b7fae-724e-4a6d-8e91-6fe3f0a2bdad` provides extra information about `http://zitgist.com/music/signal/280b7fae-724e-4a6d-8e91-6fe3f0a2bdad`

[11] See `http://purl.org/ontology/chord/`

queries. The map functionality allows the results of such queries to be plotted geographically, as shown in fig. 3.

5. FUTURE WORK

5.1 Work on the interlinking algorithm

The algorithm proposed here has several limitations. Firstly, it doesn't specify any heuristics to use if the ontologies differ. In this case,we would need a different methodology, perhaps inspired by the approach described in [8]. Secondly, the algorithm is designed for the case where a meaningful similarity measure between pairs of individual resources is available (here, using string similarity of labels attached to them with certain predicates). In a linking scenario where there is no particularly good similarity measure on individual resources and the graph structure is therefore the most salient factor for correct linking, another algorithm may be more appropriate. In this case, Melnik's "similarity flooding" [7] approach could be used, which prioritises graph structure rather than node similarity, relying on a post-processing stage to filter out unsuitable mappings.

Based on these observations, further work developing the interlinking algorithm could provide a framework for interlinking a wider variety of datasets.

5.2 Work on implementations

Currently, the GNAT tool implements two distinct approaches to finding manifestation URIs for local audio files. One uses just the available metadata, and this approach is described in § 4.2, above. Since audio metadata in personal collections is frequently incomplete, inaccurate, or missing entirely, this may not be sufficient. The other approach therefore exploits audio fingerprinting [4] to try to identify the track, and then if there is remaining ambiguity the local metadata is used to choose a single URI.

Ideally, we could use the fingerprint of an audio file as just another piece of information about the track, incorporating it into our graph mapping approach. In practice, the main fingerprinting service available with a large supporting database, MusicIP's MusicDNS service[12], is relatively opaque. Fingerprinting a track either returns a PUID, which can be used to perform a search on MusicBrainz, or returns no results. It therefore provides only a boolean test for similarity, and some hidden decisions have been made by the MusicDNS service using any available metadata. As a result, there is no obvious way to uniformly combine fingerprint information and local metadata in a graph mapping approach.

A fingerprinting service which exposed the server-side database and the actual process of matching a fingerprint to a database entry could allow for some more sophisticated linkage between personal audio collections and the Semantic Web. We have some hopes that Last.fm's recently launched fingerprinting service might gather a large database and make it freely available in a flexible way.

Although the core of the GNARQL tool is in place, more work is required to explore different approaches to crawling. Also, since Semantic Web user interfaces are relatively young, more work is required to fully exploit the functionality GNARQL is beginning to exhibit.

[12]See `http://www.musicip.com/dns/`

6. CONCLUSION

In this paper, we described several different methods for interlinking Semantic Web datasets. We mentioned two naïve approaches, leading to the construction of a more elaborate algorithm, which takes into account not only the similarity of the resources themselves but also the similarity of their neighbours. Its main advantages are to provide a best-effort mapping, without any need for a learning step (for which we would have to manually interlink some resources), and to work in a linked data environment, where new resources are discovered as we get through the mapping process. We described two implementations of this algorithm. The first one was used to interlink artists, records, and tracks in two online music datasets: Jamendo and Musicbrainz, and the second one allows any user to link their personal music collection to corresponding identifiers in the Musicbrainz dataset. We evaluated these two implementations separately.

Creating links between heterogeneous datasets can dramatically enhance the usefulness of each. Using such links, a Semantic Web user agent can jump from a band within the Jamendo dataset to the corresponding resource in the Musicbrainz one, to the corresponding resource in DBpedia, to its approximate geographic location, to the famous composers born in that city, etc. However, if it is possible to define such links manually for small datasets, it is impossible for large ones. We need methodologies to discover them in an automated way. The techniques presented here are far from perfect, but represent some initial efforts in this direction.

7. DATASETS

The following datasets are mentioned throughout the paper:

Jamendo on DBTune: `http://dbtune.org/jamendo/`
BBC John Peel sessions: `http://dbtune.org/bbc/peel/`
SBSimilarity: `http://www.isophonics.net/SBSimilarity`
Musicbrainz RDF: `http://zitgist.com/music/`
DBpedia: `http://dbpedia.org/`
Geonames: `http://geonames.org/`

8. NAMESPACES

We use the following namespaces throughout our RDF examples:

```
@prefix mo: <http://purl.org/ontology/mo/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```

9. ACKNOWLEDGEMENTS

The authors acknowledge the support of both the Centre For Digital Music and the Department of Computer Science at Queen Mary University of London for the studentship for Yves Raimond. This work has been partially supported by the EPSRC-funded ICT project OMRAS-2 (EP/E017614/1).

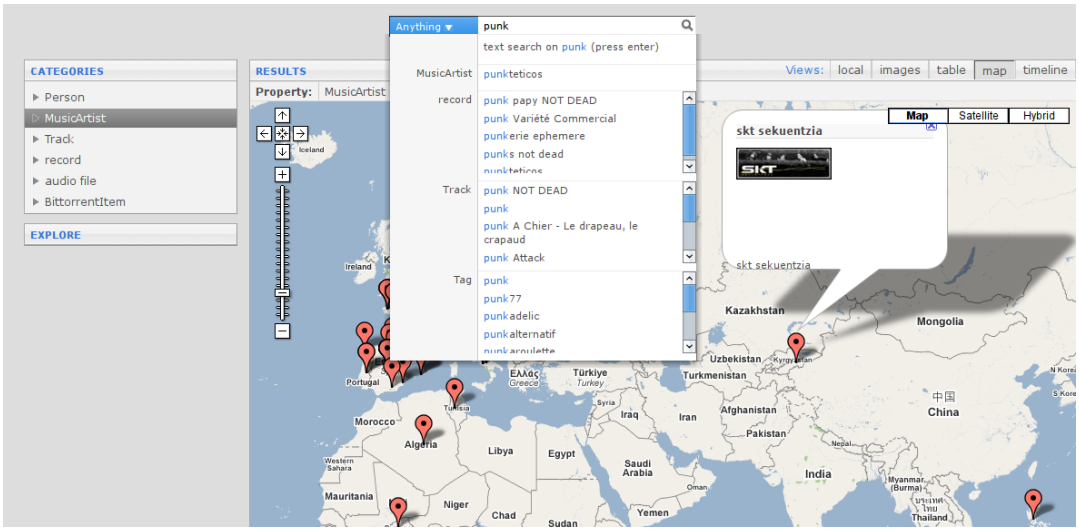

Figure 3: Navigating a personal audio collection using aggregated Semantic Web data

10. REFERENCES

[1] S. Auer, C. Bizer, J. Lehmann, G. Kobilarov, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the International Semantic Web Conference*, Busan, Korea, November 11-15 2007.

[2] Tim Berners-Lee. Linked data. World wide web design issues, July 2006. Available at `http://www.w3.org/DesignIssues/LinkedData.html`. Last accessed September 2007.

[3] Chris Bizer, Tom Heath, Danny Ayers, and Yves Raimond. Interlinking open data on the web. In *Demonstrations Track, 4th European Semantic Web Conference, Innsbruck, Austria*, 2007. Available at `http://www.eswc2007.org/pdf/demo-pdf/LinkingOpenData.pdf`. Last accessed September 2007.

[4] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of audio fingerprinting. *The Journal of VLSI Signal Processing*, 41(3):271 – 284, 2005.

[5] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs. *Journal of Web Semantics*, 2005.

[6] Michiel Hildebrand, Jacco van Ossenbruggen, and Lynda Hardman. *The Semantic Web - ISWC 2006*, volume 4273/2006 of *Lecture Notes in Computer Science*, chapter /facet: A Browser for Heterogeneous Semantic Web Repositories, pages 272–285. Springer Berlin / Heidelberg, 2006.

[7] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and itsapplication to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, pages 117–128, San Jose, CA, USA, February-March 2002.

[8] M. Neiling. Data fusion with record linkage. In I. Schmitt, C. Turker, E. Hildebrandt, and M. Hoding, editors, *Proceedings of the Workshop 'Foederierte Datenbanken'*, Aachen, 1998. Available at `http://citeseer.ist.psu.edu/189652.html`. Last accessed January 2008.

[9] Yves Raimond, Samer Abdallah, Mark Sandler, and Frederick Giasson. The music ontology. In *Proceedings of the International Conference on Music Information Retrieval*, pages 417–422, September 2007. Available at `http://ismir2007.ismir.net/proceedings/ISMIR2007_p417_raimond.pdf`. Last accessed January 2008.

[10] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago - a core of semantic knowledge. In *16th international World Wide Web conference*, 2007. Available at `http://www.mpi-inf.mpg.de/~suchanek/publications/www2007.pdf`. Last accessed january 2008.

[11] Jan Wielemaker, Zhisheng Huang, and Lourens Van Der Meij. SWI-Prolog and the web. *Theory and Practice of Logic Programming*, 2003. Available at `http://hcs.science.uva.nl/projects/SWI-Prolog/articles/TPLP-plweb.pdf`. Last accessed September 2007.

[12] W. Winkler. Advanced methods for record linkage. Technical report, Statistical Research Division, Washington, DC: U.S. Bureau of the Census., 1994. Available at `http://citeseer.ist.psu.edu/254560.html`. Last accessed January 2008.