Linked Data And You: Bringing music research software into the Semantic Web

Chris Cannam, Mark Sandler
Centre for Digital Music
Queen Mary University of London
Mile End Road, London, E1 4NS

{chris.cannam,mark.sandler}@elec.qmul.ac.uk

Michael O. Jewell, Christophe Rhodes, Mar
Department of Computing
Goldsmiths, University of Londor
New Cross, London, SE14 6NW

{m.jewell,c.rhodes,dinverno}@gold

Abstract

The promise of the Semantic Web is to democratise access to data, allowing anyone to make use of and contribute back to the global store of knowledge. Within the scope of the OMRAS2 Music Information Retrieval project, we have made use of and contributed to Semantic Web technologies for purposes ranging from the publication of music recording metadata to the online dissemination of results from audio analysis algorithms. In this paper, we assess the extent to which our tools and frameworks can assist in research and facilitate distributed work among audio and music researchers, and enumerate and motivate further steps to improve collaborative efforts in music informatics using the Semantic Web. To this end, we review some of the tools developed by the OMRAS2 project, examine the extent to which our work reflects the Semantic Web paradigm, and discuss some of the remaining work needed to fulfil the promise of online music informatics research.

1 Introduction and Motivation

As researchers and developers in the field of audio analysis and music informatics, the authors have worked on many software tools and applications which involve the interchange of data extracted from audio and music. Examples include Sonic Visualiser, an application for interactive visualisation and automated analysis of music audio; audioDB, a database that provides fast localised search of audio recordings; and the Vamp plugin system for audio analysis.

One general issue in the field of music informatics is the difficulty of sharing musical data, particularly because of copyright protections. While this is most obvious with respect to commercial audio recordings, it extends to anything with significant originality, such as typeset scores, MIDI performance transcriptions, and other symbolic representations of music, even if the original composition is itself out of copyright.

This difficulty is an impediment to the development of music informatics as a rigorous field. It is difficult to independently reproduce the results of experiments performed by other researchers, even if those researchers provide a full and precise description of their methods, because the experiment often involves a particular corpus of non-redistributable media. In addition, there is no sizeable standardised corpus within the music informatics community for testing and competition purposes, unlike in the text and image information retrieval disciplines.

OMRAS2 (Online Music Recognition and Search II) is an EPSRC-funded research project covering annotation, search, and collaborative research using online collections of recorded audio and symbolic music data. One of the goals of the OMRAS2 project is to provide a framework for enabling researchers in music informatics to collaborate and share data meaningfully, and consequently to improve the ability of music informatics researchers to test and discriminate between new algorithms and techniques.

One part of the development of this framework has been to take existing tools and applications that were developed without features for online or collaborative working, and to extend them to facilitate online use. The core idea behind our work is to maximise the ability of researchers to exchange outputs from their tools and results of their experiments in a mutually intelligible way. This has two benefits: if researchers have access to the same media but different tools, the outputs of those tools can be meaningfully compared and integrated; in contrast, if researchers have access to the same tools (for example some software licenced under Open Source terms) but different media, then the outputs from those tools can be accumulated to provide baseline performance measurements on a larger collection for a particular task.

Working from the principle that the advantages of employing standard and widely accepted formats may outweigh any inefficiencies in the data representation itself, we have found ourselves testing the suitability of the Semantic Web and Linked Data concepts for use with some of our more technically traditional music information retrieval applications and tools. Why did we choose to use Semantic Web technologies rather than, for example, focusing on Web Services methods such as SOAP? [1]

- Much work has already been done in describing musical metadata for the Semantic Web [1]. This presents an alluring vision of a future in which we can reliably associate analytical results with other information about the source material.

- A substantial quantity of data about music is already available in the Semantic Web, including data from the MusicBrainz database of artists and recordings, [2] BBC broadcast data, [3] and information from Wikipedia resources. [4] [2] Using a compatible framework for our data will make it easy to augment results with links to these existing resources.

- Semantic Web languages are widely supported by existing, free library code for data structure, storage, syntax, serialisation, and querying. This means we can use existing implementations for all of these rather than having to provide and test our own code.

- The human-readable and extensible nature of RDF makes it attractive for use in published work, and particularly for early publication of exploratory work.

- The Semantic Web deals with documents and data rather than services, which makes it attractive as a means of one-off or ad-hoc publication – especially of work done by researchers who may not be in a position to provide ongoing support of a live service.

Because of this last aspect – the document-centric, rather than service-oriented, nature of the Semantic Web – an approach based on Semantic Web technologies also provides more flexibility in deployment than service-oriented architectures. It is straightforward to publish data on the Semantic Web; a standard Web server (such as Apache) serving up static files suffices. However, if it does transpire that providing a service to query or generate data is a good idea, it can often be added after the fact as a SPARQL [5] endpoint – a mechanism by which an RDF data store may be queried remotely using syntax somewhat like that for a traditional SQL database.

1.1 About this paper

In the rest of this paper we will first introduce in more detail the technical background of the Semantic Web (section 2); we then describe some of the software tools for audio and music analysis we have developed and published during the course of the OMRAS2 project, and examine how far Semantic Web technologies and methods have been, or can be, applied to make these tools useful in new interactive and collaborative contexts, drawing conclusions and highlighting unsolved problems in section 6.

The tools we will look at are:

- The Vamp audio analysis plugin API (section 3.1);

- Sonic Annotator, a "universal audio feature extractor" using Vamp plugins (section 3.2);

- Sonic Visualiser, an interactive user application for audio analysis and annotation which is also a modular set of software libraries for building new interactive applications (section 4);

- AudioDB, a contextual search database for audio collections (section 5).

2 The Semantic Web

2.1 Ontologies

An *ontology* in information science is a vocabulary of terms used to model a domain of knowledge [3]. This vocabulary typically consists of terms for classes (or sets), properties, and relationships. The acceptance of a common vocabulary provides a semantics, making it possible to apply abstractions based on the common properties of sets of data.

The terms used in an ontology usually also exist in English or other natural languages: this provides convenience and mnemonic power, but does not itself convey meaning; the meaning of a term resides in the fact that statements using it have that term in common. The specification of a particular ontology does usually include a natural language summary of each term, but this is a hint or an attempt at normative definition rather than a descriptive definition.

2.2 RDF and URIs

RDF (Resource Description Framework) is a system for modelling data relationships using subject–predicate–object statements.

To make a statement about a real object or concept, it is necessary to have or invent a Uniform Resource Identifier or URI which will represent that object or concept. Predicates are also represented by URIs. A statement therefore takes the form *subject-uri predicate-uri object-uri*, or *subject-uri predicate-uri literal-text*.

The text of a URI can be almost anything: its purpose is not to be read, but to be a "proper noun" which we can agree upon to stand in for the real object. For historical and

[1] `http://www.w3.org/TR/2007/REC-soap12-part0-20070427/`
[2] `http://dbtune.org/musicbrainz/`
[3] `http://www.bbc.co.uk/music/developers`
[4] `http://dbpedia.org/`
[5] `http://www.w3.org/TR/rdf-sparql-query/`

```
@prefix myplugins: <http://example.org/rdf/plugins/mine#> .
@prefix vamp:      <http://purl.org/ontology/vamp/> .
@prefix cc:        <http://web.resource.org/cc/> .
@prefix dc:        <http://purl.org/dc/elements/1.1/> .

myplugins:note_estimator a vamp:Plugin ;
  vamp:identifier "notes" ;
  dc:title "Note Estimator" ;
  cc:license <http://creativecommons.org/licenses/BSD/> .

:mylibrary a vamp:PluginLibrary ;
  vamp:identifier "myplugins" ;
  vamp:available_plugin myplugins:note_identifier .
```

Listing 1. RDF/Turtle fragment describing a Vamp plugin.

practical reasons, URIs usually resemble the HTTP URLs used for the traditional Web. URIs such as

`http://my.example.org/resource/me`

may be used more conveniently after separating it into a common prefix and variable suffix, such as `http://my.example.org/resource/` and `me`. By declaring `example` as an abbreviation for this prefix, we can write the above URI as `example:me`, and reuse the `example` abbreviation for other URIs sharing that prefix.

RDF does not define a single syntax. Listing 1 provides an example using the Turtle syntax,[6] which we will use throughout. Here the URI `myplugins:note_estimator` identifies an object whose type is "Vamp plugin" (section 3.1). Its name is "Note Estimator", and it is published under a liberal open-source license identified using a URI provided by the Creative Commons project. The plugin has a particular identifier string, and there exists a library, also with a particular identifier, that contains it: these two statements relate the plugin URI (which we just invented) to the actual plugin in the "real" world.

We can then supply more information about this plugin by providing further statements using the same plugin URI as subject, either in the same document or elsewhere.

This example works because we are prepared to accept the URI `vamp:identifier` as representing the relationship "has the Vamp plugin identifier", the URI `vamp:Plugin` as representing "the type of a Vamp plugin", and so on. The reason we can readily accept the meanings of these URIs is that they have been defined elsewhere, as terms in an ontology. Listing 2 shows a fragment of the ontology which defines the Vamp plugin type.

2.3 Linked Data

Although the technologies described here are generally referred to as driving the "Semantic Web", the alternative term "Linked Data" is also used to emphasise the fact that much of their power arises from the links between documents about data, rather than the documents themselves.

The term Linked Data is appropriate for much of the database and data-centric work we are interested in, though we will continue to use Semantic Web to refer to the technologies in general.

[6] `http://www.w3.org/TeamSubmission/turtle/`

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix vamp: <http://purl.org/ontology/vamp/>.

vamp:Plugin a owl:Class ;
    rdfs:label "Vamp Plugin" ;
    rdfs:comment """
A Vamp plugin is an implementation of an audio feature
extraction algorithm using the Vamp API.
""" .
```

Listing 2. The Vamp Plugin type defined.

2.4 Ontologies for Describing Audio

A number of ontologies for describing audio and musical features and metadata have been created, many of them within the OMRAS2 project.

The *Music Ontology*[7] [1] provides terms for describing musical metadata such as artist or performance. The *Similarity ontology*[8] [4] permits the expression of the similarity between things as a class with its own properties, such as to describe the quality of that similarity, in a manner flexible enough to be able to encompass concepts as diverse as artistic influence and timbral similarity.

At a signal and feature description level, the *Audio Features*,[9] *Event*,[10] and *Timeline*[11] ontologies provide terms to represent features of audio signals and of data extracted from them, such as note or key change events or spectral features. The *Chord ontology*[12] provides vocabulary for describing chords and chord sequences. The *Vamp Plugins ontology*[13] describes properties of the Vamp audio feature extraction plugin system defined for OMRAS2 (section 3.1), and also specifies a *Transform ontology* which provides terms for describing how to configure a plugin and how a particular result (presumably expressed using the Audio Features ontology) was obtained. Finally, the *Opaque Feature File ontology* (section 3.2.2) addresses the problem of separating bulky data from descriptive metadata in RDF.

[7] `http://musicontology.com/`
[8] `http://purl.org/ontology/similarity/`
[9] `http://purl.org/ontology/af/`
[10] `http://purl.org/NET/c4dm/event.owl`
[11] `http://purl.org/NET/c4dm/timeline.owl`
[12] `http://purl.org/ontology/chord/`
[13] `http://omras2.org/VampOntology`

3 Automated Analysis

In this section we will discuss the Vamp plugin format, developed for publishing audio analysis methods in a modular system, and Sonic Annotator, a tool for applying these plugins to perform batch analysis of audio. We will describe some means by which we can provide enhanced data with these tools using Semantic Web technologies, and consider some of the limitations of this approach.

3.1 Vamp Plugins

The Vamp audio analysis plugin format [14] provides a general way to make efficient binary implementations of audio analysis and feature extraction methods available to applications. A Vamp plugin is a dynamic library of platform-native binary code which can be loaded by a host application; once a plugin is loaded, the host can feed it audio data and receive analysis results in return. The meaning of those results depends entirely on the plugin.

The result features calculated by a plugin are not just streams of values; the plugin also defines some structure for each feature, such as whether it has a start time and duration distinct from its values, how many values it contains, what its units are, and whether it has a label. This structure, although fairly simple, is enough to permit a number of useful audio features to be represented, ranging from "high level" features such as beats, notes, or song structure segments, to "low level" features such as the spectral centroid, amplitude envelope, or a constant-Q spectrogram.

The Vamp plugin system consists of a C language interface and portable C++ software development kit under a liberal open source licence, and a significant number of Vamp plugins and host applications are available from the authors and several other publishers. [15]

3.1.1 Structure and Semantics

Although features returned by Vamp plugins are structured by the plugin, they do not come with any explicit semantics attached to them. A plugin that estimates note pitch and timing information cannot explicitly identify its result features as representing the concept "note", it can only express the properties of a note by giving the feature a time, duration, and a frequency value expressed in hertz.

To make the connection between the structure and the semantic concept it represents, we can supply a separate metadata document about this plugin in RDF, giving an event type for the plugin's output. An example is shown in listing 3.

Here the `note_estimator` plugin defined earlier is described as having an output that returns events of type

```
@prefix myplugins: <http://example.org/rdf/plugins/mine#> .
@prefix vamp:      <http://purl.org/ontology/vamp/> .
@prefix af:        <http://purl.org/ontology/af/> .

myplugins:note_estimator
    vamp:output myplugins:estimated_notes .

myplugins:estimated_notes
    vamp:computes_event_type af:Note .
```

Listing 3. Defining the output feature type for a plugin.

`http://purl.org/ontology/af/Note`. Provided that we accept this URI as representing the concept "note", this suffices to identify the structured features returned in the plugin's output as notes. Even if we do not accept the semantics of this representation, the use of a common type URI still serves the practical purpose of showing that these features are interchangeable with other events of the same type.

3.2 Sonic Annotator

Sonic Annotator [16] (figure 2) is a flexible command-line utility for feature estimation and extraction from audio files using Vamp plugins, developed within the OMRAS2 project. It is generally intended to be used within the scope of larger systems where its function is to apply a particular configuration of a plugin repeatably and efficiently across multiple audio files.

In addition to the source audio, Sonic Annotator needs as input a description of the specific Vamp plugin (or plugins) to run and of the parameters to use when configuring the plugin and processing the data. This bundle of information is referred to as a "transform" description and is expressed using the Transform ontology (section 2.4).

Sonic Annotator also needs a format in which to write its feature output: something that can express all of the feature structures that may be returned by a Vamp plugin. The Audio Features ontology (section 2.4) provides terms for this.

Finally, Sonic Annotator can make use of information about the plugin itself, such metadata about the purpose of a plugin which may help it to make the right decision about how to describe the output. This information can be provided using the Vamp Plugin ontology. For example, the description in listing 3 above indicates that a plugin's output returns features that should be written using the `af:Note` object class.

3.2.1 Audio features in RDF

Listing 4 is an example of output from Sonic Annotator showing a single note, together with the contextual mate-

[14] `http://vamp-plugins.org/`
[15] `http://vamp-plugins.org/download.html`
[16] `http://omras2.org/SonicAnnotator`

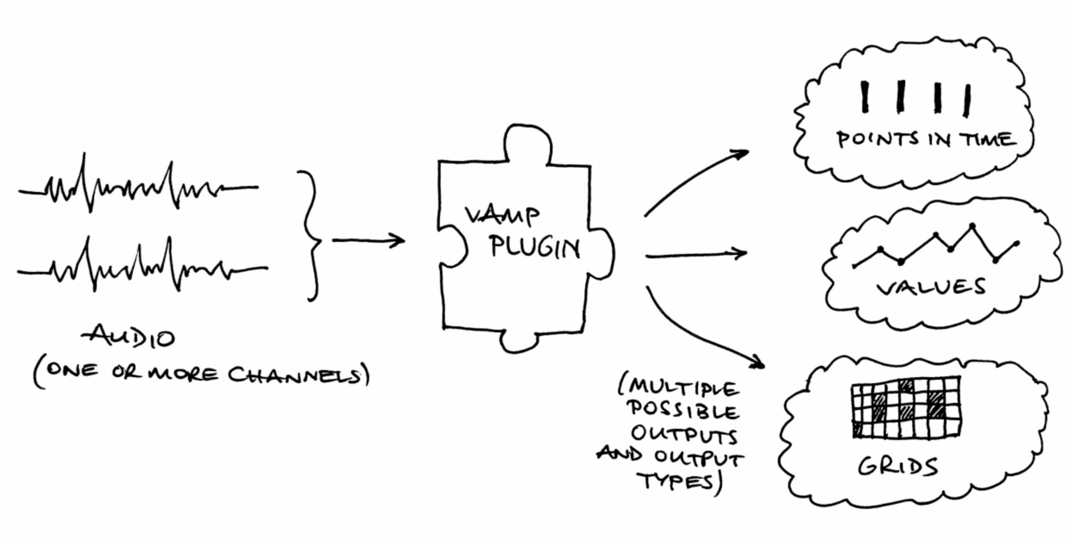


Figure 1. Overview of a Vamp plugin.

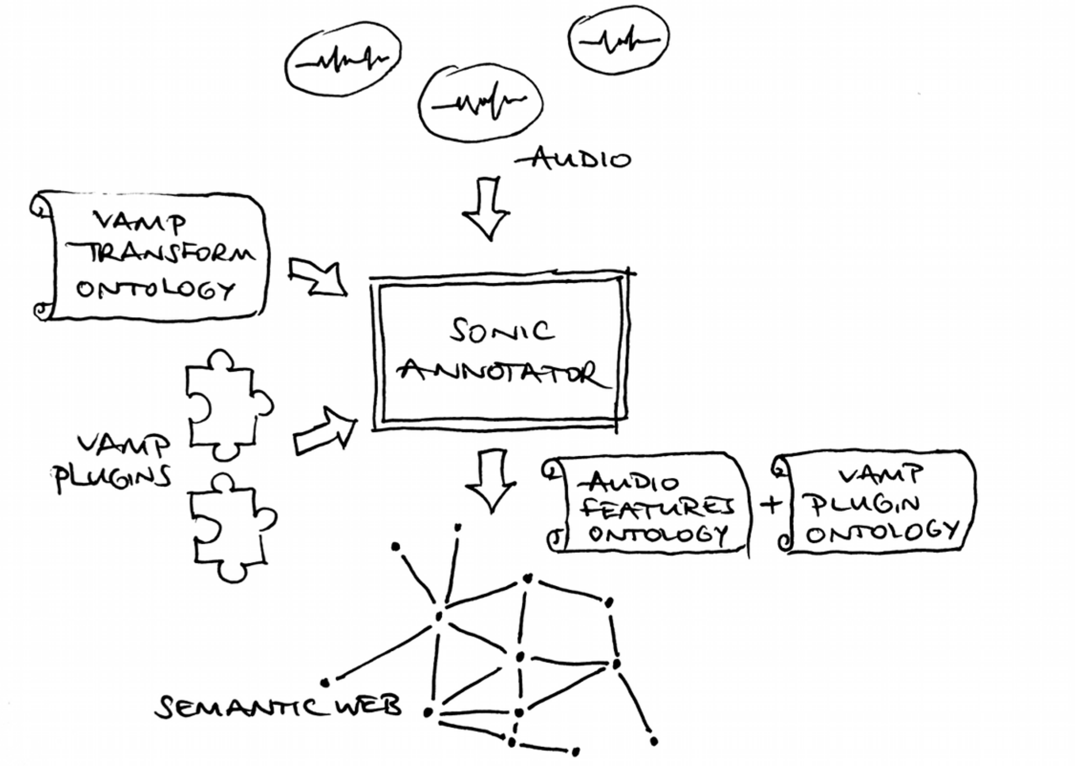


Figure 2. Sonic Annotator in context.

rial that reports how the note was calculated (which would remain the same for any number of notes).

This representation in RDF has some advantages. It can be read by humans and parsed by common library code. It is very rich: the description of each note is linked to information about the audio signal it was derived from and the plugin and parameters used to generate it. The audio file object itself may be linked to further musical metadata such as title and artist information, via further terms from the Music Ontology. (Sonic Annotator facilitates this with additional options to specify track and artist URI to be included in the output.)

Perhaps the most powerful advantage is that this rich metadata is expressed in the same syntax and structure as the data itself. This makes it very much harder to "lose" the context of a set of result data. All of the metadata will be preserved by any serialisation or storage that the data undergoes. This would not be the case if the data and metadata were separated and stored in different documents or different formats.

The representation also uses standardised encodings for the note timing, based on the Timeline ontology; this and the use of standard XML Schema (XSD [17]) data types – the type tags suffixed after `^^` – also appear beneficial for reliable interchange.

On the other hand, some disadvantages of this representation are also obvious. It is very verbose. Although the example of listing 4 is exaggerated because the whole context is reported even though there is only one note, the note object alone takes many times the space it would need in a simpler scheme. Also, the relationship between note data, generating plugin and parameters, and audio signal is complex and cannot be readily extracted from the data without a complete parse and query process.

There are further issues with RDF and SPARQL in representing and querying numerical data. XSD provides many data types for numerical values: our choices of `int` and `float` reflect the internal data types of the program that wrote the data, but we could have chosen the unbounded `integer` and `decimal`. This matters because literals are considered identical in a subsequent query only if their types match. [18] Our sample rate `"44100"^^xsd:float` will fail to match any query that searches for `"44100"^^xsd:int`, `"44100"^^xsd:decimal`, or `"44100"`. If two people or programs emit sample rates with different types, querying becomes harder (in many cases calling for a filter with an explicit type cast rather than a simple literal match – but the real problem is to know in advance which queries may require special treatment).

This problem is in principle solvable, because the type of the expected literal for a property term can be specified in its ontology. Unfortunately many ontologies, including the

```
@prefix tl: <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix mo: <http://purl.org/ontology/mo/> .
@prefix af: <http://purl.org/ontology/af/> .
@prefix event: <http://purl.org/NET/c4dm/event.owl#> .
@prefix vamp: <http://purl.org/ontology/vamp/> .

:note_1
    a af:Note ;
    vamp:computed_by :notes_transform ;
    event:time [
        a tl:Interval ;
        tl:onTimeLine :signal_timeline .
        tl:beginsAt "PT0.75"^^xsd:duration ;
        tl:duration "PT0.25"^^xsd:duration ;
    ] .

:signal_timeline
    a tl:Timeline .

:notes_transform
    a vamp:Transform ;
    vamp:output myplugins:estimated_notes ;
    vamp:plugin myplugins:note_estimator ;
    vamp:sample_rate "44100"^^xsd:float ;
    vamp:step_size "512"^^xsd:int ;
    vamp:block_size "1024"^^xsd:int .

:audio_signal
    a mo:Signal ;
    mo:time [
        a tl:Interval
        tl:onTimeLine :signal_timeline ;
    ] .

<file:///home/chris/Music/example.wav>
    a mo:AudioFile ;
    mo:encodes :audio_signal .
```

Listing 4. Output from Sonic Annotator of a single note.

Vamp plugin one at the time of writing, fail to do this, and even if they are fixed, existing data may remain incompatible. This is not a problem with the framework itself so much as with the incompleteness and fluidity of those ontologies in use and the fact that such details of the ontology are not widely interpreted by data generation tools, leaving the user to ensure that types are enforced manually.

(It is also possible to attach a language tag to textual literals, with equally awkward consequences: `"note"@en` is not the same literal as `"note"`, and SPARQL provides no way to match a literal but ignore its language. In this case the problem cannot be avoided in the ontology.)

There is no very effective way to represent numerical data in quantity directly in RDF; a textual representation of a large sequence of numbers is overwhelming for humans to absorb and inefficient for computers to parse, transmit and store. Listing 5 shows the start of an output feature from a spectrogram plugin. Despite its length, this example fails to convey any of the useful "self-describing" information found in the earlier note example. The data in

[17] `http://www.w3.org/TR/xmlschema11-2/`

[18] `http://www.w3.org/TR/rdf-concepts/\#section-Literal-Equality`

```
:spectrum a <http://purl.org/ontology/af/Signal> ;
    vamp:computed_by :spectrum_transform ;
    af:dimensions "513 0" ;
    af:value "4.07493e-11 4.12334e-11 4.26514e-11 4.49015e-11
4.78199e-11 5.11895e-11 5.47518e-11 5.82221e-11 6.13073e-11
```

Listing 5. Fragment of the output from Sonic Annotator of a spectrogram.

the `af:value` literal has no useful RDF type and is effectively opaque: no generic timeline-mapping or visualisation application, for example, would be able to use it without further information about its format. Other information that is normally useful when handling numerical data is also missing, such as the floating point precision. The only potential advantage of this encoding is that it keeps the data and information about its provenance together in a single document.

An improvement in principle might be to use an RDF collection [19] to express a sequence of typed numerical values. In practice this would be prohibitively costly even by the standards of the earlier examples, requiring two statements for every value. This represents the "dark side" of the often advantageous situation of expressing data and metadata in the same format.

3.2.2 Opaque Feature Files

Although there is no effective way to represent large quantities of numerical data directly in RDF, for many applications the textual representation in Listing 5 is adequate, with the major penalties being disk space requirements and the loss of human-readability. The Opaque Feature File project [20] represents one attempt to improve representation of dense data. This work aims to provide a common mechanism for describing in RDF the location and context of data files that are not in RDF, typically the results of some extraction or transformation process on audio data; this would allow the dense numerical data to be represented separately from the rest, while still remaining fully linked, thus restoring human-readability to the rest of the data and allowing storage of the numerical data in a more compact form. Development of the Opaque Feature File ontology is incomplete at the time of writing, but this or a similar system will be a valuable foundation for work using a mixture of RDF description with more compact data formats.

[19] `http://www.w3.org/TR/REC-rdf-syntax/\#section-Syntax-parsetype-Collection`
[20] `http://purl.org/ontology/off/`

4 Data and Analysis Visualisation

Sonic Visualiser [21] (figure 3) [5] is an application for visualisation, annotation, and automated analysis of audio recordings. It can display one or more audio files in waveform or spectral views, perform automated analysis using Vamp plugins (section 3.1), and import, export, and edit annotations such as point timings (for events like beats), notes, measurement curves, and so on; while Sonic Annotator (discussed in section 3.2) is a tool for batch analysis of audio collections, Sonic Visualiser is an interactive tool, allowing the addition and display of annotations from human subjects alongside or on top of the audio and automated analysis.

Development began prior to the start of the OMRAS2 project as a means to assist researchers by providing a flexible visualisation tool and a platform for testing and dissemination of implementations of audio feature extraction methods. Sonic Visualiser is written in C++ using the Qt toolkit, with RDF handled using Redland [22] libraries.

4.1 RDF usage in Sonic Visualiser

Sonic Visualiser uses the Vamp plugin format (section 3.1) for automated analysis. During OMRAS2 we added the ability to query Vamp plugin metadata (section 4.2 below) and to import and export annotation data using the Audio Features ontology, providing compatibility in both plugin and data formats with Sonic Annotator. (Indeed, Sonic Visualiser is actually a set of modular libraries for building visualisation and analysis applications as well as an application in its own right; Sonic Annotator is a simpler application of the same libraries.)

This provides quite a lot of power. Sonic Visualiser can import RDF descriptions of audio features, and also load both audio and annotation data from non-local HTTP URLs as well as from local storage. Because RDF about audio can link to the original audio file location and also provide audio recording metadata, the Sonic Visualiser user can export an entire session to an RDF document complete with automatic and manual annotations, audio location, and metadata. Publishing this document enables anyone to recreate the whole session, including audio and annotations, simply by giving its URL to the application.

4.2 "Find a Transform"

A straightforward facility we have added to Sonic Visualiser is the "Find a Transform" window (figure 4). This enables the user to search for Vamp plugins; for example, to type a term like "segment" and see a list of all plugin outputs that have something to do with segmentation. The term "transform" is used rather than "plugin" because the search in fact

[21] `http://www.sonicvisualiser.org/`
[22] `http://www.librdf.org/`

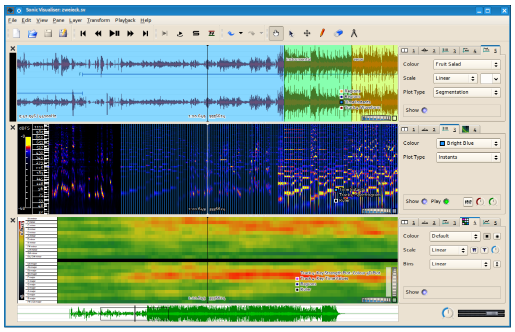
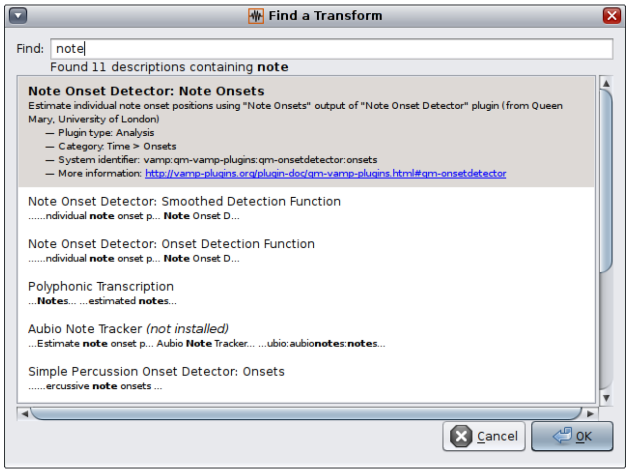


Figure 3. Sonic Visualiser.



Find a Transform

Find: note

Found 11 descriptions containing **note**

Note Onset Detector: Note Onsets
Estimate individual note onset positions using "Note Onsets" output of "Note Onset Detector" plugin (from Queen Mary, University of London)
— Plugin type: Analysis
— Category: Time > Onsets
— System identifier: vamp:qm-vamp-plugins:qm-onsetdetector:onsets
— More information: http://vamp-plugins.org/plugin-doc/qm-vamp-plugins.html#qm-onsetdetector

Note Onset Detector: Smoothed Detection Function
......ndividual **note** onset p... **Note** Onset D...

Note Onset Detector: Onset Detection Function
......ndividual **note** onset p... **Note** Onset D...

Polyphonic Transcription
...**Note**s... ...estimated **note**s...

Aubio Note Tracker *(not installed)*
...Estimate **note** onset p... Aubio **Note** Tracker... ...ubio:aubio**notes**:**note**s...

Simple Percussion Onset Detector: Onsets
......ercussive **note** onsets ...

Figure 4. Sonic Visualiser's "Find a Transform" window.

identifies individual outputs of a plugin, rather than the plugins themselves.

"Find a Transform" is driven by published descriptions, using the Vamp plugins ontology, of "known" Vamp plugins which may or may not be installed locally. These descriptions are downloaded and collated by Sonic Visualiser and searched by keyword. The documents used for this purpose are the same as those used to describe the type of output for each plugin, referred to in section 3.1.1 above. They do not have to be written by the same author as the plugin, nor bundled with the plugin; indeed, it is helpful if Sonic Visualiser can obtain descriptions even of plugins that are not installed on the user's system. The question is, how does Sonic Visualiser find out about them?

This simple problem (see figure 5) is a good example of a general difficulty in locating relevant data and documents. In order to know that your plugin is available, the host needs to find and download the RDF document that describes it. To do this, it needs to know that the document is available and where it lives. This requires either a central registry or a reliable search engine.

Alternatively, rather than publish documents about plugins we may choose to make the same information available via a service such as a SPARQL query endpoint. This poses similar problems. If each author provides their own endpoint, we have the same discovery problem with the additional difficulty that SPARQL clients cannot currently query multiple endpoints at once. If we propose a central database, we need to make it easy and secure to update.

In this case, we addressed the problem with a simple central registry of Vamp plugin RDF document locations. This is a text file served from the Vamp web server;[23] each document whose URL is given there can describe any number of plugins. However, this solution will not scale for situations involving larger and more dynamic collaborative data sets.

5 Contextual Search

Vamp plugins (discussed in section 3.2) and other audio analysis tools can generate feature vectors purporting to represent musical aspects (e.g. harmony, note onsets, timbre) from audio files; those feature vectors can be results in and of themselves, and displayed using Sonic Visualiser (section 4), but can also be used for content-based search. AudioDB is a feature-vector store and query engine for approximate matching in metric spaces, developed from observations about effective and efficient methods for performing similarity search on large collections of musical items [6, 7, 8] and sequential multimedia. It is intended to scale to millions of multimedia items, and to allow searching using sub-track fragments so that the entire database can be

[23] The registry is found at `http://vamp-plugins.org/rdf/plugins/index.txt`

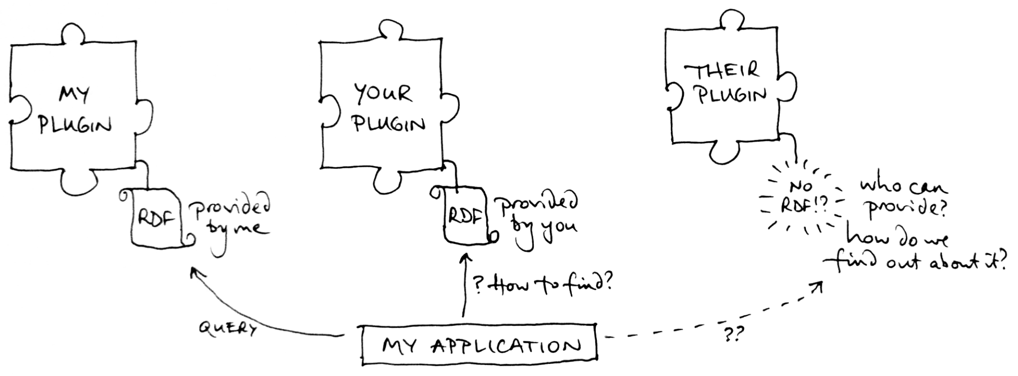


Figure 5. How do we find out about new plugins?

searched for matches for a short segment, allowing for more general searches than simple track-to-track matching.

For smaller collections, a linear search over the database may be acceptable on modern hardware; depending on the size of the collection and granularity of the features, searching for similar segments to a 5-second snippet across 1000 tracks may take no more than a few seconds. Although this search is eminently parallelisable, this will not be enough to perform useful, interesting searches over Internet-sized, or even modern consumer-device-sized, collections of musical audio. AudioDB therefore also implements a probabilistic indexing strategy, Locality Sensitive Hashing. If the length of the search is defined in advance, then a single linear scan of the database can produce an index data structure where retrieval of similarity results scales sublinearly in the number of items in the database, even in very high-dimensional spaces.

Access to this functionality is provided in a number of ways. The core software is written in C++ and provides a portable C library interface, with bundled bindings to Python, Pure Data, and Common Lisp; existing implementations of network interfaces to audioDB include a SOAP client/server model, and an HTTP server returning JSON-formatted information.

5.1 AudioDB and the Semantic Web

The online, data sharing aspect of the OMRAS2 project motivated us to provide reflective capabilities in order to share data over the Semantic Web. Firstly we are able to import dense feature data produced by Sonic Annotator, compressing by a large factor in the process, to provide a search component as part of an integrated workflow for researchers (see figure 6). AudioDB is not an RDF data store and the import process discards information not related to the feature vectors, so the exported data from Sonic Annotator must be preserved if desired.

Secondly, inspired by [9] we have implemented an audioDB interface with the facade of an RDF store, reflecting the implicit similarity judgments contained in an audioDB data structure. The feature vector data, along with a distance metric, effectively encodes similarity judgments between audio sequences, with which we can build similarity judgments between tracks. We can therefore think of the feature vectors stored within an audioDB data structure as encoding track-to-track similarity, which can in principle be exposed for Linked Data purposes as a set of statements. Since it would be hugely inefficient to actually store the $O(N^2)$ statements involved, we instead allow the user to query the database instance using SPARQL, computing similiarity judgments on demand.

To provide the query facility we have built a storage module which may be used with the Redland[24] RDF libraries. Storage modules allow for the development of triple stores that back a Redland model, and thus they need not be aware of the querying approach being employed. Our current implementation is read-only, but this may be easily extended to provide writable functionality in future.

In the case of audioDB, we cannot use a traditional database-backed store. There is some data which may be immediately queried, namely metadata related to the database's feature vectors, but similarity information must be generated dynamically. We use an internal memory storage model as a cache to store results and temporary objects for similarity queries, with audioDB itself accessed for feature information and to perform similarity searches.

Every sequence of feature vectors in the audioDB data structure is reflected using the Signal type of the Music Ontology (section 2.4); implementationally, the unique identifier corresponding to each track acts as the URI of the Signal. When a query is received, the storage module is passed a template statement with one or more of the subject, predicate, or object omitted. The module is responsible for determining what should be returned, which can be done all at

[24] `http://www.librdf.org/`

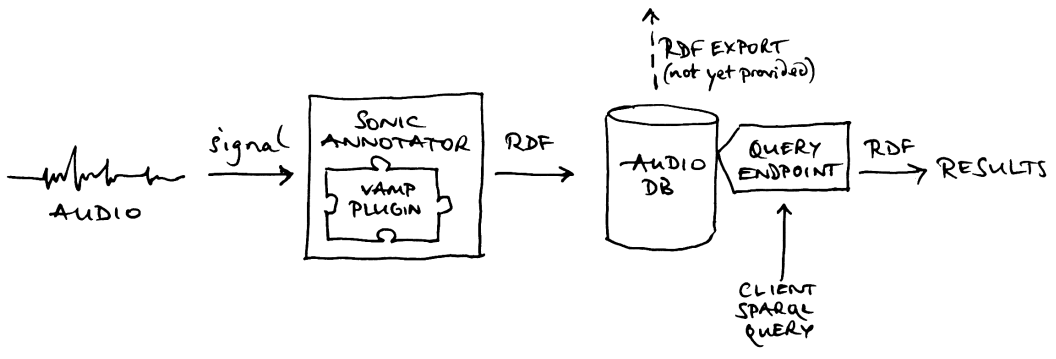


Figure 6. RDF dataflow around audioDB.

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX af: <http://purl.org/ontology/af/>
PREFIX ksa_charm:
    <http://omras2.gold.ac.uk/catalogue/ksa_charm/>

SELECT ?dimension ?vectors WHERE {
    ksa_charm:KSA_CHARM_339 a mo:Signal;
        af:dimension ?dimension;
        af:vectors ?vectors.
}
```

Listing 6. A SPARQL query to retrieve feature metadata.

```
PREFIX sim: <http://purl.org/ontology/similarity/>
PREFIX ksa_charm:
    <http://omras2.gold.ac.uk/catalogue/ksa_charm/>

SELECT ?distance WHERE {
    _:s a sim:Similarity;
        sim:element ksa_charm:KSA_CHARM_339;
        sim:element ksa_charm:KSA_CHARM_309;
        sim:distance ?distance.
}
```

Listing 7. SPARQL query to retrieve the distance between two signals.

once or lazily (on-demand).

Two separate forms of retrieval query are provided. The first is retrieval of metadata about the features from the database. This has two stages:

- Given a Signal subject and a suitable predicate, the object is filled in with the correct value from the database. The available predicates, drawn from the Audio Features ontology, are `dimension` and `vector`, representing the number of dimensions and the number of vectors in the provided feature (see listing 6).

- Given the RDF `type` predicate and the Signal class URI as object, all of the stored Signal URIs are returned as an iterable list. This allows for the retrieval of track IDs stored in the database, which may then be incorporated into other queries.

The second, more complex, query process is that of distance-based retrieval. We use the Similarity ontology[25] to specify the signals which should be compared. The Similarity class defined within this ontology has two `element` predicates, which refer to the two resources to be compared, and a `distance` predicate to either specify or retrieve the distance between these elements. In a divergence from true SPARQL, the ordering of these predicates is critical at present; both elements must be available before a distance predicate is supplied. The following process is applied:

- Given the RDF `type` predicate and the Similarity class URI, a blank node is created and cached in the internal store.

- Given the above Similarity instance reference as the subject, the `element` predicate, and a Signal URI, this information is also cached in the internal store.

- Given the Similarity instance subject and `element` predicate but no object, a statement is created for each Signal URI, with the URI replacing the object.

- Given the Similarity instance subject and the `distance` predicate, the distance is calculated using the audioDB API and the provided element URIs. An exhaustive search is employed here, and a Euclidean distance measure returned.

5.2 Issues

While this approach does provide the means to query audioDB data structures via a standard SPARQL endpoint, it

[25] `http://purl.org/ontology/similarity/`

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX af: <http://purl.org/ontology/af/>
PREFIX sim: <http://purl.org/ontology/similarity/>
PREFIX ksa_charm:
    <http://omras2.gold.ac.uk/catalogue/ksa_charm/>

SELECT ?signal ?distance WHERE {
    ?signal a mo:Signal.
    _:s a sim:Similarity;
        sim:element ksa_charm:KSA_CHARM_339;
        sim:element ?signal;
        sim:distance ?distance.
}
ORDER BY (?distance) LIMIT 5
```

Listing 8. SPARQL query to retrieve the 5 signals closest to the input.

is not yet a viable solution for large databases. The distance querying process currently compares tracks on an individual basis, but in many cases (such as listing 8) it is possible to perform the query with a single call to audioDB. Adapting the storage module to support this form of optimisation is difficult, however, as statement templates are supplied individually, and the results are expected immediately. This is particularly inefficient in the above query, as every track must be compared to every other track in separate calls to the backend database.

Secondly, the query must be written in a specific order to ensure that the storage model is able to perform the search. As such, Similarity individuals must be declared prior to any elements, and element predicates must be declared prior to the distance predicate. The Similarity predicates should be allowable in any order, but as the distance predicate relies on knowing the two signals to compare, this is currently impossible.

Finally, as mentioned above, the feature import process disregards metadata about the feature extractor and source audio. This information must be obtained by querying against an additional metadata store. This is straightforward when using two queries with two separate SPARQL endpoints, but techniques to execute queries against multiple endpoints are not yet standardised.

6 Conclusions and Future Directions

In this paper, we have described some of the software tools and applications developed during OMRAS2 and the initial steps we have taken to make them interoperable using Semantic Web technologies. Our motivation in doing this work was to make the interchange of data, experimental results, and tool outputs relatively easy.

We believe that enabling this interchange using open and extensible technology will give researchers confidence that they can work together without having to interpret divergent data formats or move away from ways of working with which they are already comfortable. The knowledge that data, and descriptions of data, can persist and be understood independently of any particular tools should be of great interest to researchers and anyone with an interest in the outcomes of research.

One way of assessing the progress and potential of this work is to consider how much data of interest to music informatics researchers is already available: for example, there are about 14 billion statements in our music-related knowledge store at `http://dbtune.org/`. This number should of course be taken with a pinch of salt: many such statements convey generic *scènes à faire*, transcodings of other data sources, or connections between one datum of interest and another, but it is nevertheless an indication of the general scale enabled by work applying the technologies described here.

A point to note about data published through these mechanisms is that so far for the most part it is musical metadata, rather than data related to the content of musical artifacts. This metadata information is certainly of interest to music informatics researchers, as it allows exploration of relationships and connections between artists, performers, works, and listeners, but it leaves largely unexplored the publishing of content-based metadata, for reasons that we have touched on in this paper and summarise in section 6.1 below. One notable exception is the publication of summary content-based information from the *SoundBite* application,[26] containing single features for 152,410 tracks produced by 6,938 distinct artists [10], whose data is used in a recommender system and open to other uses.

In the process of permitting the achievement of these qualified successes, we have encountered some problems and limitations in our approach. In the sections below, we summarise these limitations, consider how they can be resolved, and outline some future work.

6.1 Difficulties and Limitations

- Encoding data directly in RDF can be wasteful of space and processing time (section 3.2.1). This is problematic because much work in this field depends on exchanging very large data sets.
- There is no efficient means of structuring numerical data such as vectors or matrices (section 3.2.1).
- Simple numerical data are sometimes encoded using inconsistent data types (section 3.2.1), making queries hard to write and, more seriously, unreliable when used with real-world data from multiple sources.
- More sophisticated data types tend to have wildly different encodings across different sources: for exam-

[26] `http://www.omras2.org/SoundBite/`

ple, there are many different ways to encode a "birth date" literal type, most of which can be found in existing linked data repositories.

- Where it is desirable to separate data and metadata (section 3.2.2), there is not yet any standard way to link between the two for storage or delivery purposes.

- The unordered nature of RDF terms does not easily lend itself to optimisation for queries such as distance computation in audioDB (section 5.2). Further implementation effort is required to ensure that queries are performed in a manner that prioritises low-cost searches.

- Ontologies are typically incomplete and "under construction", and updating them can be hard because of the risk of orphaning already-published data and the lack of effective methods for supporting multiple versions of an ontology.

- Querying across multiple data sets in order to make use of existing linked data is difficult with current tools (section 4.2). The standard SPARQL query language does not support federated queries, and there are no standard means of discovering, pooling, and cacheing multiple documents about a subject.

6.2 What Next?

Our work on enhancing existing software tools has yielded promising results, with some significant limitations. We believe that many of the current limitations are surmountable:

- Work on the Opaque Feature Files project or similar (section 3.2.2) should be sustained, in order to provide a reliable means for combining RDF data with data not well suited to representation in RDF.

- AudioDB should be extended to support data export using RDF, perhaps in conjunction with Opaque Feature Files.

- Published ontologies need to be improved to clarify the intended data types for literal values (section 3.2.1), and tools that write statements should be made to import and follow the type information specified in ontologies in order to reduce the probability of failures resulting from type mismatches.

- More work should be carried out into investigating reliable methods for querying across literals with multiple data types.

- Developers of tools that perform queries using SPARQL should be encouraged to support federated query (section 4.2). The Jena framework proposes one possible extension for this. [27]

Despite the issues raised in section 6.1, we are optimistic about the future for this work and keen to develop it further. The fact that these issues and limitations are an issue at all is a reflection of how much is easily and transparently implementable. Also, many of the difficulties are independent of the fact that we have used Semantic Web technologies: for example, data file formats are changed or extended just as ontologies are.

Further work includes integration of Sonic Annotator into a web service for automated feature extraction (SAWA); a Linked Data resource providing information about composers and works from the classical canon, and applications for structured browse and search among recordings of these works using components from Sonic Visualiser and audioDB; and deployment of our tools in the context of the Networked Environment for Musical Analysis [28] project.

7 Acknowledgements

The work in this paper was supported by the EPSRC through the OMRAS2 project EP/E017614/1, EP/E02274X/1.

Sonic Visualiser and the Vamp plugin format were initially developed by Chris Cannam as part of the European Commission SIMAC project IST-FP6-507142. AudioDB was originally conceived and developed by Michael Casey.

8 References

[1] Y. Raimond, S. Abdallah, M. Sandler, and F. Giasson. The Music Ontology. In *Proc. International Symposium on Music Information Retrieval*, pages 417–422, Vienna, Austria, September 2007.

[2] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee. Media meets semantic web - how the BBC uses DBpedia and linked data to make connections. In *Proceedings of the European Semantic Web Conference In-Use track*, 2009.

[3] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[4] K. Jacobson, Y. Raimond, and M. Sandler. An Ecosystem for Transparent Music Similarity in an Open World. In *Proc. ISMIR*, pages 33–38, Kobe, Japan, October 2009.

[27] `http://jena.sourceforge.net/ARQ/service.html`
[28] `http://nema.lis.uiuc.edu/`

[5] C. Cannam, C. Landone, M. B. Sandler, and J. P. Bello. The sonic visualiser: A visualisation platform for semantic descriptors from musical signals. In *Proc. International Symposium on Music Information Retrieval*, pages 324–327, 2006.

[6] M. Casey and M. Slaney. The Importance of Sequences in Music Similarity. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, volume V, pages 5–8, Toulouse, France, May 2006.

[7] M. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney. Content-Based Music Information Retrieval: Current Directions and Future Challenges. *Proceedings of the IEEE*, 96(4):668–696, 2008.

[8] M. Casey, C. Rhodes, and M. Slaney. Analysis of Minimum Distances in High-Dimensional Musical Spaces. *IEEE Transactions on Audio, Speech and Signal Processing*, 16(5):1015–1028, 2008.

[9] Y. Raimond, C. Sutton, and M. Sandler. Interlinking Music-Related Data on the Web. *IEEE Multimedia*, 16(2):52–63, 2009.

[10] D. Tidhar, G. Fazekas, S. Kolozali, and M. Sandler. Publishing Music Similarity Features on the Semantic Web. In *Proc. ISMIR*, pages 447–452, Kobe, Japan, October 2009.