Tawny-Overtone: Mixing Music and Metadata

Phillip Lord and Michael J. Bell

June 26, 2014

Abstract

In this document, we describe our work on the Tawny-Overtone project. This short project investigated the notion of born-semantic music, where semantic annotation can be added at any point by the musicians involved in the music production process. We have achieved this by combining two systems, one capable of music synthesis, and one capable of producing semantic metadata into a single system. This allows us to attach metadata to any part of the music synthesis process, and to percolate this metadata with the music specification. We describe the technical detail behind this project.

1 Introduction

Many fields of publishing are changing rapidly. One of these changes is, the incorporation of semantics following the advent of Linked Open data and related semantic technologies. The idea of attached or embedding rich semantic descriptions into media is an attractive one, enabling later querying and search.

There have been several attempts to describe music, using semantic or related technology. The music ontology describes what might be broad characterised as the technical and social knowledge about a piece of music[1]. Less public tools such as the Music Genome Project from Pandora describe music in terms of mood and feel.

Previously, we have investigated semantics during the academic publication process. We now have a significant body of linked open data available; the idea of semantic publication is, therefore, an attractive one, where documentation is embedded in the linked data environment, providing a rich narrative through it. The narratives that describe the production and content largely remains in the form of the "lumped" PDF; despite being in existence since the early 1990s scientific publication has been only lightly touched by the web, yet alone the semantic web. We have argued a prerequisite for changing this situation is to incorporate semantics at the point of production; in this way, we can enable the computer to enable the author to write, check and compose better text (Lord et al., 2012). As a by-product, these semantics are available to pass through the publication process and can, in turn, make life richer for the reader. In short, the authors have the knowledge necessary to add rich metadata; if they do not gain from adding this, why would they bother?

In the case of semantic authoring, we developed our "kblog" tools which integrated semantics into the authoring process by adding a light-weight syntax

[1]`http://www.musicontology.com`

to an existing blogging engine, which is then translated into a rich representation of mathematics, a bibliography and, finally, a widget displaying a citation for the articles. The authors benefit from these tools; for example, the bibliography generator takes a URI, and generates a reference list, which both saves the author typing and effectively checks that the URI is live and correct[2].

For the Tawny-Overtone, we wish to address the same question for music: can we enable the musician to both compose or perform music at the same time as they author metadata about that music. The most obvious objection to this is that music is very different from authoring. While there are notations for music, they are not used for all forms of music and only used during composition; during performance there is little or no writing at all.

Here, we square this circle by focusing on two novel environments, one called Overtone which enables the programmatic creation of music as part of a composition or in a live performance environment, and the other called Tawny-OWL which allows development of metadata in a similar way[3]. Critically, these use the same programmatic environment and, therefore, a mixable syntax.

In this document, we discuss how this technical integration has been performed, and potential future development for Tawny-Overtone.

2 Technical Background

2.1 Overtone

Overtone is a collaborative, programmatic environment for music generation. It uses the SuperCollider system as a sound-generation backend, with an extensive library (overtone) at the front. While SuperCollider produces the actual sounds, Overtone provides most of the abstractions on top of this; it is possible to define sounds or to load samples, to define instruments, then rhythms, melodies and chords.

Overtone itself is implemented in Clojure – a modern Lisp dialect implementing on the Java Virtual Machine (JVM), although it is now also cross-compilable to other environments, including JavaScript. Clojure is a general-purpose programming language; most of the abstraction features of Overtone, for example, are simply built up using Clojure functions and data structures.

Clojure also has access to all the facilities that would be expected of a modern language, either implemented directly in Clojure or through its strong Java (strictly JVM) interoperability. Overtone uses this in a variety of ways: it has a sophisticated download and caching system for its sampled instruments (using samples from freesound.org). Its general networking and concurrency facilities mean that several performers working together can control a single backend synthesizer in a synchronized manner.

2.2 Tawny

The development of Tawny-OWL was partially inspired by Overtone. Overtone allows the fully programmatic creation of music, backed by a general-purpose

[2]Part of this tool chain, `http://greycite.knowledgeblog.org`, also works with LaTeXwas used for this paper

[3]Although, not yet as part of a live performance

programming language; Tawny-OWL provides many of the same facilities for the composition of ontologies and ontological metadata.

Tawny-OWL enables the fully programmatic construction of ontologies by wrapping the OWL API(Horridge and Bechhofer, 2011) which is also used as the basis for the Protègè Ontology Editor(Various). Tawny-OWL abstracts some details of this API, including it's change object system and factory patterns, so simplifying its use; at the same time, it brings an evaluative shell, which enables rapid, iterative development which are borrowed directly from Clojure. The Lisp heritage of Clojure also makes it a more natural fit for ontology development; an ontology is a set of statements, as a Lisp program is a set of functions. As a simple example, Listing 1 shows an example from the Tawny-OWL version of the widely used pizza ontology (Stevens, 2010).

```
(as-disjoint
  (defclass Pizza
    :label "Pizza")
  (defclass PizzaTopping)
  (defclass PizzaBase)
```

Listing 1: Pizzas in Tawny

That both Tawny-OWL and Overtone share the same syntax and interpretation environment provides a unique opportunity to combine the two. This should enable the author to build up both ontological knowledge and a music representation at the same time.

3 Ontological Resources

In the first instance, we decided to use the Music Ontology as the basis for description within Tawny-Overtone. This was not a particularly principled decision, more one of convenience. It lacks many concepts that we might validly wish to describe about music such as rhythm and key, as well as concepts about sounds such as oscillators, filters or sample-rate. It is, however, well-known and well-documented.

4 Reading Ontologies

In order to use the music ontology within Tawny-OWL, it must first be made available. This can be achieved in one of two ways. Firstly, we can *import* the ontology directly using OWL API functionality to read the relevant OWL files. While this makes all the axioms and entities in the ontology available, they have to be referred to by IRI. The second approach is to *read* the ontology; this approach first imports the ontology and then *interns* a Lisp symbol for every entity; this provides a more naturalistic method for referencing terms, as shown in Listing 2. This comes at the slight cost of hiding the type of the entity which is apparent in the former; the music ontology has both a class called `Record` and a property called `record` differentiated only by their capitalization.

```
;; Referencing by name
;; => #<OWLClassImpl <http://purl.org/ontology/mo/Record>>
(owl-class (iri "http://purl.org/ontology/mo/Record"))
```

```
;; Referencing by symbol
;; => #<OWLClassImpl <http://purl.org/ontology/mo/Record>>
Record
```

Listing 2: Referencing in Tawny

Our initial attempts to import or read the Music Ontology were hampered by two difficulties. Firstly a bug in the OWL API was preventing correct handling of the Music Ontology URI; full resolution requires following a 302 "Moved Temporarily" redirect, and then a 303 "See Also". The OWL API was following only the former. This issue was subsequently fixed (see commit `03d5b97e`). Secondly, either reading or importing an ontology also imports the entire import graph; at the time (and still at the time of writing), the MuSim ontology (`http://purl.org/ontology/similarity`) resolves to an informative page rather than an ontology, something which is in conflict with the OWL specification (Motik et al., 2012).

In addition to these immediate difficulties, loading these ontologies from their URIs was a significant performance issue; therefore, we introduced support to the `read` functionality of Tawny-OWL to use local resources and a convenience function to download the import graph. The issues with the MuSim ontology meant this had to be completed by hand.

As a second step, to `read` an ontology requires transforming some aspect of that ontology into the Clojure symbol that will be used to refer to it. For the Music Ontology, this was achieved straight-forwardly by using the fragment of the IRI, which results in a human-readable name.

5 Ontologizing Music

There are a number of ways in which we could integrate a notation combining music and OWL differing in their level of integration. We consider three possibilities, describe their implementation and the value that they could bring.

First, we consider the value of music and metadata in a single file; this represents a very loose integration between the music notation and the metadata, but also carries immediate benefits. By packing both pieces of knowledge together, the two become linked and can be passed around together. Although this form of semantics is very light-weight, we have shown that it is beneficial in other areas; for example, our own kblog-metadata tools embedded Open Graph Protocol metadata into web pages, which allows others to accurately refer to these pages.

With Tawny-Overtone, this is trivial to achieve and follows directly from having a single syntax. We simply add ontological statements to the file which also contains our music specification (see Listing 3).

```
(tawny.owl/defindividual ctford :type Composer)
```

Listing 3: Declaration of an Individual

However, while it may still be useful, this form of representation is quite limiting. The metadata is attached to only the entire file which may contain many elements of a piece of music – for instance, sounds, rhythms or melody, which may require different types of metadata.

A second problem is that our attachment is purely syntactic; it will not be carried with the computational entities that we have created. For example, consider the following harpsichord sound (see Listing 4).

```
(definst harpsichord [freq 440]
  (let [duration 1]
    (*
     (line:kr 1 1 duration FREE)
     (pluck (* (white-noise) (env-gen (perc 0.001 5) :action FREE))
            1 1 (/ 1 freq) (* duration 2) 0.25))))
```

Listing 4: An Harpsichord Sound

The programmatic nature of overtone means that we could potentially use this definition from other Clojure namespaces by importing it. In these circumstances, it would be effectively stripped of its metadata.

Clojure provides a solution to this, as a general programming language feature called, rather unfortunately in this case, “metadata”. To distinguish between the two, we will use the terms “Clojure metadata” and “OWL metadata”. Clojure metadata is supported by most entities within the Clojure language, including symbols and most data structures. This metadata is a Clojure map of key-values pairs which can be any object at all.

Clojure uses this feature for many purposes, including tags (a Java class which enables avoiding reflection in compiled code), privacy, stating dynamic scope or documentation. We show a contrived example below (see Listing 5)

```
(def
  ^{:tag Integer
    :private true
    :dynamic true
    :doc "This is a variable called f"
   f)
```

Listing 5: Metadata in Clojure

We can use this feature with Tawny-Overtone to link specific subsets of the music to metadata. For example, we can associate a composer and arranger with a melody (see Listing 6).

```
(tawny.owl/defindividual ctford :type Composer)
(tawny.owl/defindividual MichaelBell :type Arranger)

(def #^{:doc "This is metadata"
        :owl (owl-and
               (owl-some composer ctford)
               (owl-some arrangement_of MichaelBell))}
  melody
  (let [pitches
        [:C4 :C4 :C4 :D4 :E4]
        durations
         [1 1 2/3 1/3 1]
         times (reductions + 0 durations)]
     (map vector times pitches)))
```

Listing 6: A melody with Metadata

The advantage of this approach is that an aggregate document which used this melody as part of a remix would still carry the ontological metadata with in, which could then be used to present, a description of all the artists who had contributed and in what capacity they have contributed to a piece of work. For example, the melody in Listing 6 extends upon the original composition by Chris Ford (?), who is still attributed in the ontological metadata.

6 Discussion

During the Tawny-Overtone project, we have completed an initial integration between an innovative music synthesis platform and an innovative knowledge representation system.

The integration had some technical challenges associated with it; these have been overcome, and have resulted in improvements both to our own software and to the OWL API, which should lessen these challenges in the future.

We have investigated various techniques for integration between the music and OWL annotation. However, we believe it is possible to go further. Strictly, the code shown (Listing 6) associates the metadata with the Var object that is identified by the `melody` symbol; in Common Lisp terminology this is equivalent setting a property on symbol; or in Java annotating a method. We could also have associated it with the actual melody data structure, which is a list in this case. We believe that this offers the tightest level of integration between music and OWL metadata. This metadata may allow us to predict whether elements of music are compatible either in key, rhythm, timbre or higher-level concepts such as mood or appropriateness.

Part of the motivation for this work, was to recast our notion that semantics must make life better for authors (Lord et al., 2012), by making life better for musicians. The work described in this paper lays the semantic ground-work, but does not immediately address this issue. To do so, the metadata that we have embedded needs to be made visible within an existing environment; with Overtone, this currently means within a live-programming, music performance environment which presents an interesting sets of challenges.

References

M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web Journal*, 2, 2011.

P. Lord, S. Cockell, and R. Stevens. Three steps to heaven. *SePublica*, 2012. URL `http://www.russet.org.uk/blog/2012/04/three-steps-to-heaven/`.

B. Motik, P. F. Patel-Schneider, and B. Parsia. Owl 2 web ontology language structural specification and functional-style syntax (second edition). `http://www.w3.org/TR/owl2-syntax/`, 2012. URL `http://www.w3.org/TR/owl2-syntax/`.

R. Stevens. Why the pizza ontology tutorial? `http://robertdavidstevens.wordpress.com/2010/01/22/why-the-pizza-ontology-tutorial/`, 2010. URL `http://robertdavidstevens.wordpress.com/2010/01/22/why-the-pizza-ontology-tutorial/`.

Various. The protégé; ontology editor and knowledge acquisition system. `http://protege.stanford.edu/` [Online. last-accessed: 2012-10-07 18:08:04]. URL `http://protege.stanford.edu/`.