The Mobile Audio Ontology: Experiencing Dynamic Music Objects on Mobile Devices

Florian Thalmann, Alfonso Perez Carrillo, György Fazekas, Geraint A. Wiggins, Mark Sandler
Centre for Digital Music
Queen Mary University of London
Email: f.thalmann@qmul.ac.uk

***Abstract*—This paper is about the *Mobile Audio Ontology*, a semantic audio framework for the design of novel music consumption experiences on mobile devices. The framework is based on the concept of the *Dynamic Music Object* which is an amalgamation of audio files, structural and analytical information extracted from the audio, and information about how it should be *rendered in realtime*. The Mobile Audio Ontology allows producers and distributors to specify a great variety of ways of playing back music in controlled *indeterministic* as well as *adaptive* and *interactive* ways. Users can map mobile sensor data, user interface controls, or autonomous control units hidden from the listener to any musical parameter exposed in the definition of a Dynamic Music Object. These mappings can also be made dependent on semantic and analytical information extracted from the audio.**

I. Introduction

Within less than a decade, the capabilities of mobile devices grew so quickly that a majority of the tasks previously accomplished on personal computers can now easily be managed on these devices. Even more, mobile devices have developed into compact multi-sensory computers, bringing novel ways of interaction, such as multi-touch gestures, accelerometer control, or geolocation tracking into everyones hands and pockets [1]. Yet, the music listening experience is only slowly adjusting to this new environment and its capabilities. Although music players add functionality derived from social platforms such as recommendation schemes and shareable playlists, the listening process itself is often no different from how it was using a Walkman in the early eighties. Most music applications, even more experimental ones, usually do not take advantage of the controls available of mobile devices and still base their interfaces on an emulation of skeuomorphs from the early analog world, such as sliders and knobs [2].

In this paper, we introduce the *Mobile Audio Ontology (MAO)*, a Semantic Web framework that investigates new ways in which music can be experienced on mobile devices. The MAO defines so-called *Dynamic Music Objects (Dymos)* which in this work we see as bundles of music files and a flexible structural definition of the musical material that also gathers semantic information extracted from the files and exposes a variety of musical parameters. A playback configuration called *rendering*, which can also be specified using the MAO, then defines a multiplicity of mappings between the analytical features, the controls available on a mobile device, and specific musical parameters of the Dymo. A Dynamic Music Object will typically sound different each time it is listened to, according to constrained musical variations which can depend on several factors including spontaneous decisions made by the player, listener preferences, the contextual situation the listener may be in, such as time of the day or activity, and finally some simple user interface and sensor controls made available to the listener.

The *MAO* allows producers, distributors, or consumers themselves to customise a listener's experience for any musical format such as single tracks, albums, playlists, mixes, or more experimental formats. A simple interface, a prototype for which is called *Dymo Designer*, allows them to import the music, define the structural definition, view the semantic information extracted from the files, and finally define the mappings of the playback configuration graphically for a great variety of use cases, all of which can then automatically be exported to a file that is publishable on the Semantic Web. The listeners, in turn, can use a platform-independent mobile player, such as the prototype called *Semantic Music Player*, which understands the ontological format, reads the audio, queries the structural and analytical data, and takes decisions in realtime.

We begin by providing a brief overview of earlier musical ontologies that we build on and the tools used to extract semantic information from the music files. Then, we describe our notion of Dynamic Music Objects and their relation to so-called Digital Music Objects. In the main part of the paper, we then introduce the *MAO* more formally, explain its connections to other ontologies and give several examples illustrating the variety of use cases that can be defined using the ontology. Finally, we briefly discuss the two prototypical applications, the producer-oriented *Dymo Designer* and the consumer-oriented *Semantic Music Player*.

II. Musical Ontologies and Extracting Semantic Information from Music

Since the existence of the internet all its contributors have been accumulating information and data at an exponential pace, referencing each other's sites, and publicising their own point of view about any conceivable subject matter. Even though the internet is an almost endless and dynamic source of information, the fact that it is mainly based on text makes it difficult to be interpreted by machines rather than humans. This is what Semantic Web technologies try to

address by standardising the ways in which semantic relationships between instances of data on the web can be expressed. Formats such as the *Resource Description Framework (RDF)*[1] or the *Web Ontology Language (OWL)*[2] enable the definition of web ontologies, which describe real-world entities and their properties as dynamic graph structures made of simple logical triples based on first-order logic, which can be read and interpreted by computers. The data published using such ontologies can be gathered, queried, and inferred upon using the SPARQL language.[3] More recently, more light-weight formats that optimise both performance and minimises space requirements have emerged including JSON-LD[4], a JSON-based linked data compatible serialisation format for RDF data.

As the popularity of these technologies is growing, increasingly many musical platforms use them to describe a great variety of musical information. These can be classified into two main kinds. *Extra-musical* metadata, as for instance gathered in the platform MusicBrainz,[5] describes composers, musicians, production environments, dates, etc, and is typically annotated manually. On the other hand, *analytical metadata* or simply *features* are typically extracted from audio or symbolic representations using the algorithmic methods of *Music Information Retrieval (MIR)* and describe musical characteristics such as harmony, melody, tempo, segmentations, as well as more low-level descriptive or acoustic quantities, such as the spectral centroid or the amplitude envelope.

There is a great variety of musical ontologies that describe either of the two types of musical information and thereby focus on different aspects of music. The *Music Ontology* is the most basic of these ontologies and defines a vocabulary for and relationships between musical actors and groups (musicians, composers, etc), musical items (CDs, files, scores, instruments, etc), manifestations and events (concerts, venues, etc), as well as activities and processes and their results (orchestration, transcription, etc) [3]. Several smaller ontologies were defined as extensions of the Music Ontology, e.g. the *Studio Ontology*, the *Event Ontology*, the *Chord Ontology*, or the *Tonality Ontology*, the latter two of which describe higher-level analytical features [4]. Finally, the *Audio Feature Ontology* resides at the other end of the information spectrum and describes musical features and existing extraction methods in a more technical and reproducible way.

Data using feature-based ontologies rely on the extraction of analytical features from music files, which can for instance be done using *Sonic Annotator*[6], a command line tool that outputs feature data in various formats. Sonic Annotator analyses audio files based on any number of given *Vamp Plugins*, each of them specialised on a small number of related features [5]. It's most relevant feature to this work is that using Vamp Plugins, Sonic Annotator can output data as RDF files, thereby linking to appropriate musical ontologies telling us about the data. Other serialisation formats include simple CSV or JSON based formats, with an RDF compatible JSON-LD format under development.

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/owl2-overview/
[3] http://www.w3.org/TR/sparql11-query/
[4] http://json-ld.org
[5] http://musicbrainz.org
[6] www.vamp-plugins.org/sonic-annotator/

III. Dynamic Music Objects

The Mobile Audio Ontology builds on the notion of *Dynamic Music Objects (Dymos)*, which we define as flexible and modifiable musical objects that can be played back in various ways. Dymos extend the more general concept of a Digital Music Object (DMO), a musical adaptation of the *Research Object*. The latter is an amalgamation of a research publication, its results, and the methods used to arrive at the results, such as collected data and computer code [6]. One of the main motivations behind Research Objects is to facilitate reproducible research, enabled by the inclusion of metadata and executable workflows of scientific experiments.

De Roure defines *Executable Music Documents* first as a bundle of music research, analogous to Research Objects. However, he then extends the definition to composition, production, and consumption: "[the DMO] enables ease of reuse and remixing of music right through the chain from composition to consumption, with the consumer equally empowered to produce and compose." [7]

It is the latter of De Roure's understanding that we build on here. We think of *Dynamic Music Objects* as Digital Music Objects aimed at consumers and intermediary music professionals. In contrast to the broader Digital Music Objects, they are designed to be flexible and modifiable within degrees of freedom and constraints that are given by the producers and composers. They are meant to be played back directly based on this information, rather than remixed or reused in a more general sense. Within these constraints, Dymos typically sound different every time they are listened to. Their musical variations are based on semantic information about the music and can be brought about either by autonomous decision-making units or by a certain form of listener interaction. More specifically, we define Dymos as:

- a bundle of *music files*
- a *structural definition* relating the music files, enriched with *semantic analytical data* extracted from the audio, and enabling a number of modifiable *musical parameters*
- a playback configuration called *rendering*, which maps controls and features to parameters

The MAO facilitates and standardises the definition of the latter two, structural definitions and renderings, while attempting to keep definitions as general as possible in order to allow for a great variety of use cases. In the following, we describe the current state of the ontology.

IV. The Mobile Audio Ontology

The *Mobile Audio Ontology* is an OWL ontology that partially builds on previous musical ontologies (Section II) and focuses on describing ways in which music can be rendered and controlled with contemporary mobile devices. Its top-level

concepts are the structural definitions of Dymos and their renderings, as described in the previous section.

Dymos are currently defined as recursive multi-hierarchical structures of objects, each of which can have sub-objects or *parts*, and can be linked to other objects in other hierarchies via similarity relations. These structures are based on CHARM, an abstract music representation system based on abstract data types, allowing multiple hierarchies of musical objects or events related by arbitrary logical formulae and abstracted from concrete applications [8]. CHARM is currently being reimplemented using OWL, JSON-LD, and JavaScript [9].

Renderings are composed of any number of *mappings* that map the values of single or groups of *controls* and *features* to any of the musical *parameters* available in the Dymo structure. After a quick overview of how mappings are commonly defined in computer music, we will devote a section to each of these concepts and their types.

A. Conceptual Background

Mappings are a common way to think about controllers and interaction in computer music helping to simplify the interface while ensuring maximum musical flexibility. Hunt and Kirk claim that the musical parameters of a multi-parametric system are best controlled using a complex combination of convergent (many-to-one) and divergent (one-to-many) mappings, rather than with a confusing multitude of one-to-one mappings [10, p. 235]. They further suggest the use of weighting and biasing (dependencies between controls) when combining several mappings. In multiple experiments, such interfaces proved to be less challenging and more engaging to the user, and they sparked a longer-lasting interest [10, p. 255]. Other authors suggest multi-level mappings, where controls map to intermediate layers of higher-level parameters, which are then mapped to the musical parameters, in a fashion reminiscent of neural networks [11, p. 639].

Any such mappings can be described using mapping functions of various shapes. The method Hunt and Kirk suggest leads to weighted sum functions of linearly mapped control values, while multi-level mappings are already quite complex polynomial functions. With the MAO, we generalise these mappings by allowing any kind of mapping functions such as linear, logarithmic, triangle, square, trigonometric, or polynomial functions, which can then be combined using sum and product functions etc.[7]

As a further generalisation, the function parameters can include any kind of control, feature, or Dymo parameter rather than merely interface and sensor controls, as suggested in the sources cited above. In this way, we can build mapping situations that take analytical information about the music into account. We encourage applications using the MAO to use complex many-to-many mappings, and especially one-to-many mappings using few controls, which are perhaps the most suitable for mobile devices in that they simplify the interface and can add a sense of mystery that may encourage listeners to experiment in discovering the various ways in which the music will play back.

[7]The current implementation even allows mappings to be defined as arbitrary serialised JavaScript functions which can contain any algorithm based on parameters within the function scope.

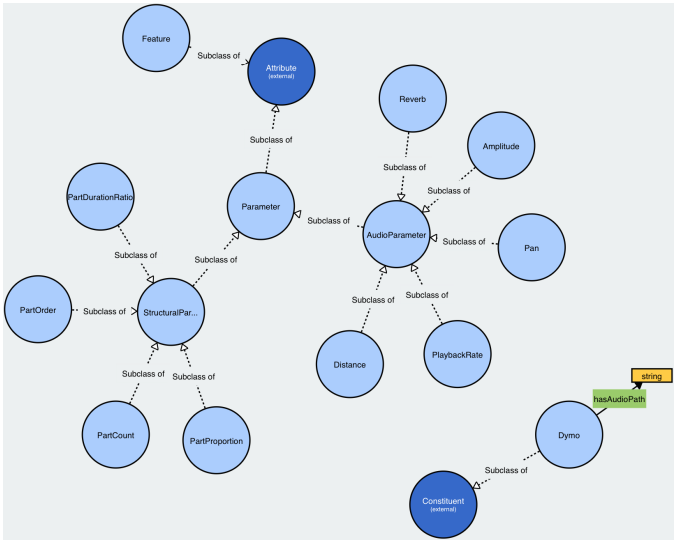

Fig. 1. Dynamic Music Objects as defined by the MAO. The relationships between Dymos, Parameters, and Features are defined by the CHARM Ontology [9].

B. Dynamic Music Objects and their Parameters

The *MAO* represents *Dymos* as multi-hierarchical structures of musical objects (`:Dymo`),[8] which are sub-classes of CHARM constituents (see above). Each object may contain a set of sub-objects or parts (`:hasPart`) which are again Dymos. As extensions of constituents, Dymos can also have a type (`:hasType`) which determines how their parts relate to each other and to the parent, e.g. form a sequence, a simultaneity, or more musically distinct types, such as a melody or a progression.[9] A Dymo can also be directly related to Dymos in other hierarchies by being connected via relations such as `:similarTo`. Figure 1 shows the definitions the MAO adds to the CHARM ontology, visualised using WebVOWL.[10]

Each Dymo, or each node in the Dymo graph, can contain any number of immutable analytical features (`:hasFeature`) and mutable musical parameters (`:hasParameter`),[11] as well as musical source files such as audio files (`:hasAudioFile`). The musical *parameters* defined for each object can be any selection from the standard parameters defined by the ontology, which are sub-classes of `:Parameter`, such as amplitude, panning, or temporal positions, as described in the following.

[8]In the following we informally discuss OWL classes simply by referring to their names (words that start with upper-case letters) and the properties that have a domain restriction on that class (words that start with lower-case letters).

[9]Types enable different ways of manipulating dymos.

[10]http://vowl.visualdataweb.org/webvowl/

[11]These two properties are both sub-properties of the CHARM property `:hasAttribute`, which point to general musical attributes.

If the same parameter occurs at various adjacent positions in the *Dymo* hierarchy, we can define functional relationships between them using arbitrary mappings.[12] Whenever a higher-level parameter is changed, lower-level ones are adjusted accordingly. With the predicate `:hasInitialValue`, we can also define initial parameter positions, relative to which modifications will then take place. Analytical features, sub-classes of `:Feature` are based on the feature ontology and can take any values, which will however not be possible to be changed.

Using these definitions, we can characterise any of the most common structures found in audio processing. For instance, we can define a simple multichannel mix with the following triples:[13]

```
:mix a :Dymo ;
   :hasParameter :Amplitude ;
   ...
   :hasPart :track1, :track2, :track3 .
:track1 a :Dymo ;
   :hasParameter :Amplitude ;
   ...
```

This definition enables us to access the overall amplitude of the object (the master level), as well as the amplitude of each sub-object, in this case the single tracks.

Figure 2 shows a slightly more complex example of a Dymo structure representing a simple multi-track mix that offers parameters on various hierarchical levels. In this example, we can control the amplitudes of the main *mix* object, which changes the overall amplitude, but also for the *riddim* object, which merely affects organ and drums. This is a simple example of grouping, a technique frequently used in audio mixing. We also ensure access to a few other audio parameters such as reverb, delay, or panning, which we briefly discuss in the following sections. The example also shows how music files can be associated with Dymos. Here, only objects with no parts refer to an audio file. If one of them is played back, we will merely hear the respective file, whereas a playback of the `mix` object will result in a simultaneous playback of all three files.

1) Audio and Playback Parameters: These parameters currently include `:Amplitude`, `:PlaybackRate` (which affects pitch, as with tapes and records), `:TimestretchRatio` (which leaves pitch unchanged), `:Transposition` (pitch shifting with no change of duration), `:Reverb`, `:Filter`, and `:Delay`. The binary parameters `:Play` and `:Loop` simply start and stop Dymos, and activate and deactivate Loop mode.

2) Spatial Parameters: Current spatial parameters include `:Pan`, `:Distance`, and `:Height` (relative to the spatial coordinate system). There are also some global spatial parameters, which are tied to the listener rather than specific Dymos, such as `:ListenerPosition` and `:ListenerDirection` (see Section V-A for an example).

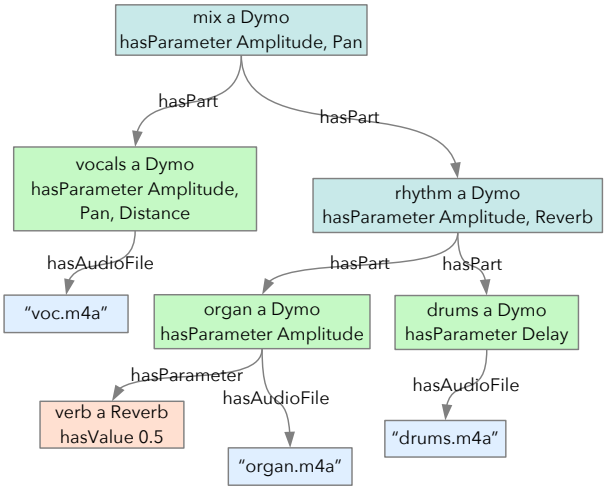

Fig. 2. A sample Dynamic Music Object.

3) Structural and Temporal Parameters: These parameters can be assigned to objects at any level that contain parts, however these are meant to be played back (sequentially, simultaneously, or tiled). `:DurationRatio` affects the total time the material of an object will be played for. If the material is longer than the given duration, it will be cut off, if it is shorter, it will be played past its ending point if possible, or looped otherwise. `:Onset` determines the time an object will be played at.

`:PartCount` concerns the discrete number of parts of an object played. For instance, if we expose this parameter in bar (measure) objects containing a number of parts corresponding to beats, we become able to change the meter of the piece by either increasing or decreasing the beat count. In case of an increase, some of the segments will be repeated. Similarly, we can get access to chord complexity by changing the number of chord members played back.

`:PartOrder` affects the order in which subelements are played back. `:Tempo` simply determines the playback speed at the current level of the Dymo.

`:PartProportion` determines the proportional length for which Dymo parts are played back. For instance, again on the beat level, we can shuffle the rhythm (increase the swing amount), by increasing the proportion of odd-numbered parts.

Figure 3 illustrates these parameters schematically for a multilevel temporal segmentation.

4) Higher-Level Parameters: In addition to the predefined parameters, users can define their own higher-level parameters for any dymo and link them to lower-level ones by defining an appropriate set of mappings (from the custom parameter and any features to the lower-level parameters).

5) Features: As mentioned above, Dymos can also include features (`:Feature`), which can contain any analytical data about the objects, such as the spectral centroid, average amplitude, meter, tempo, or harmonic information. These data

[12]This is a generalization of the relative transformation of satellites as for instance described in [12]. To model that case, we can define a mapping function that adjusts lower-level parameters by the same amount in which the higher-level parameter was changed.

[13]The examples in this paper and are expressed in Turtle, a textual syntax for OWL ontologies (www.w3.org/TR/turtle/). The `a` keyword refers to the `rdf:type` property, which defines instances of OWL classes, written in capitalized words. In some figures we use a simplified version of this notation.

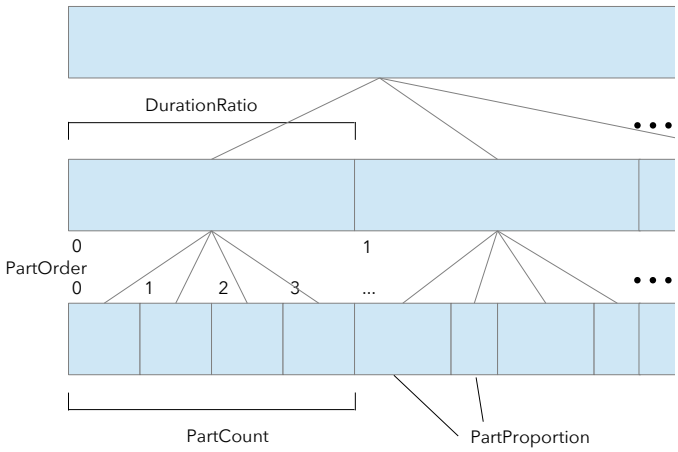

Fig. 3. The structural parameters illustrated with a multilevel segmentation.

can then be used either in mappings to mutable parameters, as described in the next section, or while querying larger, more complex Dymos in order to get subsets of their sub-objects, which can again be targeted by mappings. We will discuss some example features in greater detail later on.

A special type of features, segmentations, are best represented by the Dymo structure itself. Examples are onset, beat, bar, or section features. Such features divide the music into segments, which are best represented as parts of a higher-level Dymo referring to the music file. This way, we can easily represent multilevel segmentations, e.g. with beats being parts of bars, bars parts of sections, and sections parts of a main object.

C. Renderings and Mappings

Apart from defining Dynamic Music Objects and their parameters and features, the MAO also describes how they can be played back on mobile devices in indeterminate, adaptive, and interactive ways. Figure 4 shows a simplified view of the MAO definitions of renderings and mappings. The top-level constructs are so-called *renderings* (`:Rendering`) that refer to a Dymo (`:hasDymo`) as well as to a number of mappings (`:hasMapping`) which describe how any of the Dymo's available parameters can be controlled.

Mappings (`:Mapping`) map to parameters in any arbitrary subset of Dymos (`:toDymo`). They can have any number of *domain dimensions* (`:MappingDimension`) and are based on a mapping function (`:hasFunction`) which can be one of several types (linear, logarithmic, exponential, triangle, but also serialised JavaScript function, see Section IV-A). The function value is then sent to all associated parameters (`:toParameter`) of all Dymos that are mapped to via `:toDymo`.

By mapping from a Dymo's immutable features we can give specific initial values to each parameter, which will stay the same in case there is no control mapping to the same parameter. This way, we can for instance smoothen or exaggerate the local tempo of a piece, by inversely mapping a tempo feature to the `:TimestretchRatio` parameter. Feature

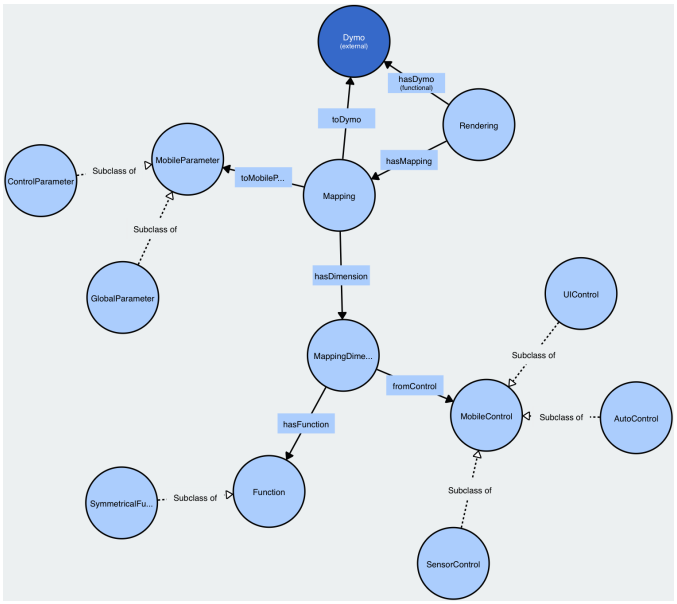

Fig. 4. A simplified view of renderings and mappings defined by the MAO.

mappings can also be used for creative or analytical purposes. We can for instance highlight certain analytical aspects of the music, e.g. exaggerate expressive variety by mapping an amplitude feature to the `Amplitude` parameter.[14]

Mappings that include controls, on the other hand, map the data stream of any controls to Dymo parameters. As mentioned in Section IV-A, some of these controls are interactive and allow users to influence the music or interact with it, whereas others are autonomous or hidden and let the player take decisions on its own. In the following, we briefly survey the various control categories and their members that are currently available. All controls are subclasses of the OWL class `:MobileControl`.

1) Mobile Sensor Controls: As mentioned earlier, mobile devices typically contain a great number of sensors, all of which can be accessed via APIs. In the*MAO*, they are grouped by the class `:SensorControl`. Any multi-dimensional sensor controls are currently split into single dimensions so that in mappings they can be recombined in arbitrary ways. Accelerometer controls are thus split into `:AccelerometerX`, `:AccelerometerY`, and `:AccelerometerZ` and geolocation controls into `:GeolocationLatitude` and `:GeolocationLongitude`. Further sensor controls are one-dimensional, including `:CompassHeading` or `:GeolocationDistance`. There are also higher-level sensor controls, derived from other controls, such as `:TiltX` and `:TiltY`.

2) UI Controls: In addition to these sensor controls *MAO* also supports more traditional UI controls (`:UIControl`). Currently, the ontology supports traditional skeuomorphs such as `:Slider`, `:Button`, and `:Toggle`. In the future, we will also

[14]Another such possibility is described in [13], where we sonify decomposed audio recordings by spatialising it based on analytical features.

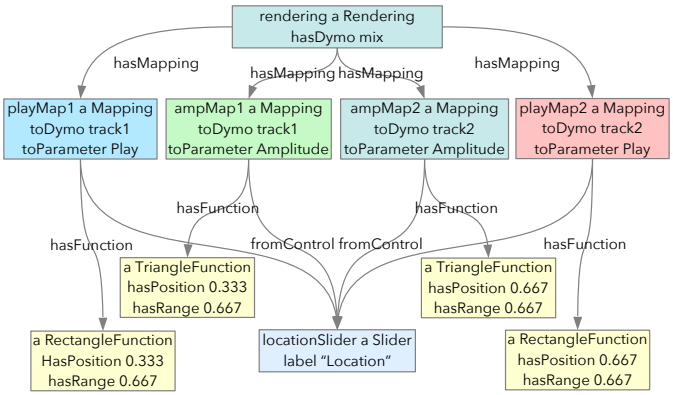

Fig. 5. A sample rendering of a Dynamic Music Object with two tracks.

add controls such as `:TouchX` and `:TouchY` controls which will be configurable for multitouch with `:hasFingerIndex`.

3) Autonomous Controls: Currently there are two types of autonomous controls (`:AutoControl`), both of them based on statistical decisions. `:RandomControl` is simply meant to output streams of random numbers based on given statistical distributions, which can be associated with the control via `:hasDistribution`. `:BrownianControl` is similar in that it outputs values based on various kinds of random walks. On the other hand, `:GraphControl` navigates the Dymo structure via a given type of relation (`:hasRelation`). An implementation of a graph control chooses a path in the given graph, in the most simple case, for instance, by randomly choosing among the possible outgoing edges for every node. It will then set the associated parameter of each Dymo it reaches along the way. For instance, the graph control can be used to navigate the similarity relations (`:similarTo`) of Dymo structures, as will be shown in an example later on.

4) Contextual Controls: In the future, we will also add contextual controls `:ContextualControl` to the ontology, which are based on contextual information gathered from the web, such as user preferences, trends, or weather information.

Figure 5 shows a simple example of a rendering with control mappings where one slider is mapped to various parameters of a Dymo with two parts, each of them representing an alternate track. The two mappings with the rectangle functions decide when each of the tracks is started or stopped while two mappings with the triangle functions control the amplitudes which creates a cross-fade between the two tracks. Figure 6 shows a graph of the resulting mapping, where the vertical axis represents both the *Play* and the *Amplitude* parameters. Multi-dimensional mappings can quickly lead to intricate results and they can be defined in an easy and intuitive way using the Dymo Designer, which will be introduced later on. In the next section, we will give a few more intricate examples of use cases in order to illustrate the versatility of the ontology.

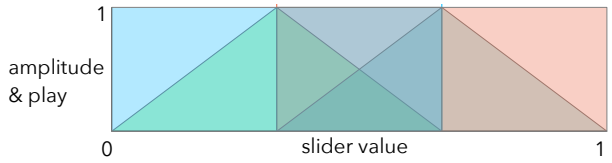

Fig. 6. Graph of the mappings in Figure 5.

V. A Few Sample Use Cases

A. Spatial Navigation

The simplest type of Dymo is just a single audio track represented by a single Dymo as follows:

```
:spatial a :Dymo ;
    :hasAudioFile "spatial.m4a" ;
    :hasParameter :Distance, :Reverb ;
    ...
```

With the two `:hasParameter` lines we provide access to the Dymo's distance and reverb parameters. We can then define a rendering that maps compass and geolocation sensor data to these two parameters as well as to the global `:ListenerOrientation` parameter as follows:

```
:orientationMapping a :Mapping ;
   :hasDimension [ a :MappingDimension ;
      :fromControl [ a :CompassHeading ;
         :isRelative True ] ;
      :hasFunction [ a :LinearFunction ] ] ;
   :toParameter :ListenerOrientation .
:distanceMapping a :Mapping ;
   :hasDimension [ a :MappingDimension ;
      :fromControl [ a :GeolocationDistance ;
         :isRelative True ] ;
      :hasFunction [ a :LinearFunction ] ] ;
   :toDymo :spatial ;
   :toParameter :Distance, :Reverb .
:spatialUseCase a :Rendering ;
   :hasDymo :spatial ;
   :hasMapping :orientationMapping,
      :distanceMapping .
```

Here, both the `:CompassHeading` and the `:GeolocationDistance` are defined relative to the movement of the listener here. The `:distanceMapping` is an example of a one-to-many mapping (Section IV-A) where we increase both the distance of the audio source and the reverb associated with it while the listener moves away from the initial position with their mobile device. Simultaneously, depending on the direction the device points to, the listener orientation is updated. Altogether, this gives an impression of the musical source staying where the listener was in the first moment.

In a similar way, we could implement a use case where several musical voices, such as the instruments of a band, are distributed in space and the listener could navigate around between them or away from them.

B. Multitrack Mixing

Another basic use case can be created using a Dymo just like the one defined in the first example in Section IV-B. If we define a set of tracks as parts of a main Dymo and provide access to some low-level mixing parameters such as amplitude

and panning, these can then be mapped to sliders or knobs as follows:

```
:t1AmpMap a :Mapping ;
   :hasDimension [ a :MappingDimension ;
      :fromControl [ a :Slider ;
         rdfs:label "Bass␣Amplitude" ] ;
      :withFunction [ a :LinearFunction ] ] ;
   :toDymo :track1 ;
   :toParameter :Amplitude .
   ...
:mixUC a :Rendering ;
   :hasDymo :mix .
   :hasMapping :t1AmpMap .
   ...
```

With similar definitions, the processes of multitrack players such as [14] can be described including the definition of various presets that can be triggered via appropriate controls.

C. Location Mapping

In order to illustrate *multidimensional mapping*, we can create a rendering based on two-dimensional geolocation mappings, which assign two different sub-Dymos to two different geolocation positions. First, we would have to define a Dymo with two tracks, similar to the one used in Section V-A, which however includes alternative rather than parallel audio files, similar to the Dymo illustrated in Figures 5 and 6. The rendering could then look like this:

```
:track1Location a :ProductMapping ;
   :hasDimension [ a :MappingDimension ;
      :fromControl :GeolocationLatitude ;
      :hasFunction [ a :TriangleFunction ;
         :atPosition 0 ;
         :hasRange 0.6 ] ] ;
   :hasDimension [ a :MappingDimension ;
      :fromControl :GeolocationLongitude ;
      :hasFunction [ a :TriangleFunction ;
         :atPosition 0 ;
         :hasRange 0.6 ] ] ;
   :toDymo :track1 ;
   :toParameter :Amplitude .
:track2Location a :ProductMapping ;
   ...
:locationMixing a :Rendering ;
   :hasDymo :mix .
   :hasMapping :track1Location, track2Location .
   ...
```

The choice of a product mapping (`:ProductMapping`) with two triangular functions results in a cone structure, at the maximum of which the amplitude of the corresponding track is the highest. More intricate definitions could be used to describe location-based audio experiences reminiscent of the ones described in [15] consisting of many alternative and intersecting Dymos and sub-Dymos played in sub-regions etc, as illustrated in Figure 7. In the example of this figure, we use the same technique as in Section IV-C where we used rectangle function mappings to the `:Play` parameter in order to synchronise parallel tracks.[15] We could then define similar mappings to other musical parameters in order to increase the amount of musical variation.

[15]For a more interesting result, we can replace these functions with arbitrary discrete and gradual polygon-containment functions. The Dymo Designer allows users to draw such polygons and calculates the corresponding serialised JavaScript functions.

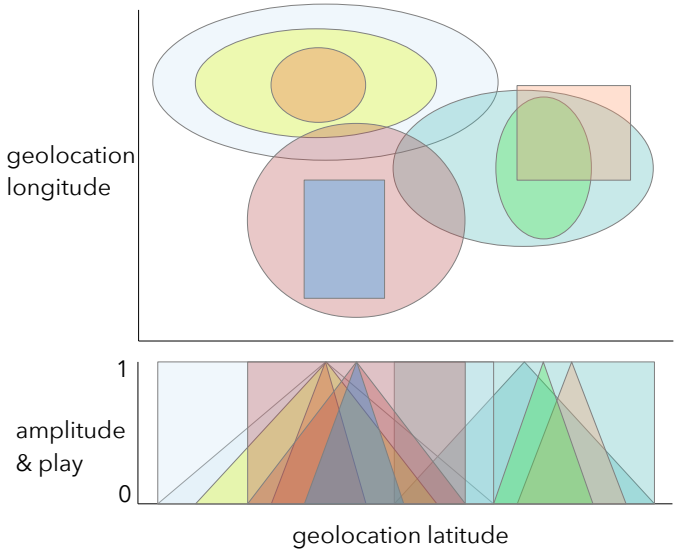

Fig. 7. A top-down view and a cross-section of mappings from the two-dimensional geolocation controls to the play and amplitude parameters of a hierarchical Dymo.

D. Graph-Based Variation of Temporal Structure

If we wish to vary a Dymo's temporal structure using the parameters described in Section IV-B3 we need to define a multilevel Dymo (here `:beats`) in the fashion of the schematic graph in Figure 3. We can then use the following rendering to navigate its similarity structure:

```
:beatMapping a :Mapping ;
   :hasDimension [ a :MappingDimension ;
      :fromControl [ a :GraphControl ;
         :hasRelation :Similarity ] ;
      :hasFunction [ a :LinearFunction ] ] ;
   :toDymo :beats ;
   :toParameter :PartOrder .
:beatGraphNavigation a :Rendering ;
   :hasDymo :beats ;
   :hasMapping :beatMapping .
```

VI. First Applications Based on the Ontology

In the context of this project we are working on two prototypical tools that use the ontology at two different ends, one that facilitates the definition of Dymos and their renderings and a second one that plays them back.

A. The Dymo Designer

Defining use cases in Turtle, as we did all along in this paper, can be rather tedious or confusing for non-ontologists. If we plan to make the ontology available to music professionals, e.g. producers or distributors, it will be helpful to provide a tool that facilitates the definition of Dymos. The *Dymo Designer* is a browser-based application that visualises Dymos in highly flexible ways and provides a simple and intuitive interface for automatically adding features to the structure and defining mappings supported by the MAO. All these things can be done in a visual and interactive way, mapping functions can be drawn onto the screen, either one-dimensional ones as

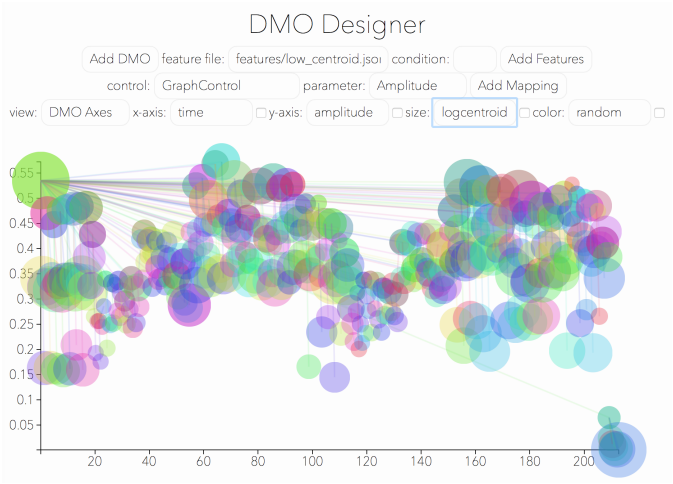

Fig. 8. The Dymo Designer showing an adventurous visualization of a multilevel segmentation object enriched with amplitude and spectral centroid features.

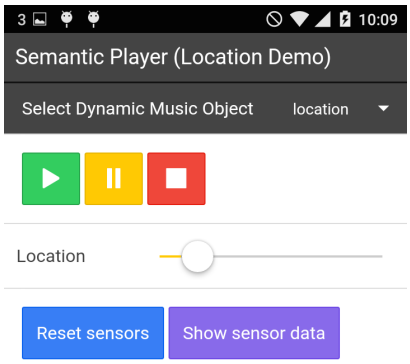

Fig. 9. The GUI dynamically generated for the rendering in Figure 5, shown on an Android device.

simple graphs, or multi-dimensional ones as areas in multi-dimensional space. Finally, the musical result can also be previewed using mock controls. Figure 8 shows the current version of the Dymo Designer representing an object with several levels of temporal segmentation.

B. The Semantic Music Player

The *MAO* evolved to the current state in parallel with a prototypical player simply called the *Semantic Music Player (SMP)* which is able to play back anything that can be described using the ontology. The *SMP* is a cross-platform mobile app that also works in browsers on any computer. It is based on *Ionic*[16] and various W3C standards, such as the *Web Audio API*[17]. It reads Dymos from *MAO* turtle files (currently using *rdfstore.js*[18]) or JSON-LD files, which are typically encapsulated in file folders containing all referenced audio files and analytical data. For each such Dymo, the player dynamically loads the necessary drivers for the sensors and constructs a visual interface containing any specified UI controls (see Section IV-C2). Figure 9 shows the interface generated for the rendering shown in Figures 5 and 6, consisting of just one slider labelled *Location* and the standard interface buttons.

VII. Conclusion and Further Work

We have introduced a first version of the Mobile Audio Ontology and illustrated how it works by giving some example use cases. The Semantic Music Player and the Dymo Designer are two early prototypes of applications that show the potential and the versatility of the framework. However, there are still many potential extensions of the framework and as the applications evolve, new use cases will be encountered and incorporated into the ontology. Some of these extensions were already suggested in this paper. For instance, one could introduce new types of contextual controls based on data taken from the web, as well as more advanced controls based on artificial intelligence that navigate the structural definitions of Dynamic Music Objects in more informed and intricate ways.

[16]http://ionicframework.com
[17]https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html
[18]https://www.npmjs.com/package/rdfstore

References

[1] G. Essl and M. Rohs, "Interactivity for mobile music-making," *Organised Sound*, vol. 14, no. 2, pp. 197–207, 2009.

[2] T. Kell and M. M. Wanderley, "A high-level review of mappings in musical ios applications," in *Proceedings ICMC, SMC*, Athens, Greece, 2014.

[3] Y. Raimond, S. A. Abdallah, M. B. Sandler, and F. Giasson, "The music ontology," in *ISMIR*, 2007, pp. 417–22.

[4] G. Fazekas, Y. Raimond, K. Jacobson, and M. Sandler, "An overview of semantic web activities in the omras2 project," *Journal of New Music Research*, vol. 39, no. 4, pp. 295–311, 2010.

[5] C. Cannam, M. Sandler, M. O. Jewell, C. Rhodes, and M. d'Inverno, "Linked data and you: Bringing music research software into the semantic web," *Journal of New Music Research*, vol. 39, no. 4, pp. 313–25, 2010.

[6] D. De Roure, "Towards computational research objects," in *Proceedings of the 1st International Workshop on Digital Preservation of Research Methods and Artefacts co-located at Joint Conference on Digital Libraries*, Indianapolis, 2013.

[7] ——, "Executable music documents," in *Proceedings of the 1st International Workshop on Digital Libraries for Musicology*, 2014, pp. 91–3.

[8] M. Harris, A. Smaill, and G. Wiggins, "Representing music symbolically," in *Proceedings of the IX Colloquio di Informatica Musicale*, Venice, 1991.

[9] N. Harley, "An ontology for abstract, hierarchical music representation," in *Late-Breaking Demo at the 16th International Society for Music Information Retrieval Conference (ISMIR 2015)*, Malaga, Spain, 2015.

[10] A. Hunt and R. Kirk, "Mapping strategies for musical performance," in *Trends in Gestural Control of Music*, M. Wanderley and M. Battier, Eds. Paris: Ircam - Centre Pompidou, 2000.

[11] M. M. Wanderley and P. Depalle, "Gestural control of sound synthesis," *Proceedings of the IEEE*, vol. 92, no. 4, pp. 632–644, 2004.

[12] F. Thalmann and G. Mazzola, "Visualization and transformation in general musical and music-theoretical spaces," in *Proceedings of the Music Encoding Conference 2013*. Mainz: MEI, 2013.

[13] F. Thalmann, S. Ewert, M. Sandler, and G. A. Wiggins, "Rendering decomposed recordings spatially – integrating score-informed source separation and semantic playback technologies," in *Late-Breaking Demo at the 16th International Society for Music Information Retrieval Conference (ISMIR 2015)*, Malaga, Spain, 2015.

[14] G. Herrero, P. Kudumakis, L. J. Tardón, I. Barbancho, and M. Sandler, "An html5 interactive (mpeg-a im af) music player," in *Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research (CMMR), Marseille, France*, 2013, pp. 15–18.

[15] A. Hazzard, S. Benford, and G. Burnett, "Sculpting a mobile musical soundtrack," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, Seoul, 2015.