Semantic Metadata for Music Production Projects

Thomas Wilmering, György Fazekas and Mark B. Sandler

Centre for Digital Music (C4DM)
Queen Mary University of London
thomas.wilmering@eecs.qmul.ac.uk

Abstract. In this paper we describe how the Audio Effects Ontology, an extension to the Studio Ontology, can be used for the ontological representation of detailed metadata about the use of audio effects in music production projects. The ontologies are using Semantic Web technologies that enable knowledge representation and sharing, either on the Semantic Web or local RDF databases maintained by music production studios. The generated metadata facilitates reproducibility and detailed analysis of music production practices. We discuss how audio effect implementations and transformations are conceptualised in the ontologies, give examples of real-world use cases and present results of a qualitative evaluation.

1 Introduction

The development of tools and services for the realisation of the Semantic Web has been a very active field of research in recent years, with a strong focus on linking existing data. In the field of music information management, Semantic Web technologies may facilitate searching and browsing, and help to reveal relationships with data from other domains. At the same time, many algorithms have been developed to extract low and high-level features, which enable the user to analyse music and audio in detail. The use of semantics in the process of music production however is still a relatively new field of research. With computer systems and music processing applications becoming increasingly powerful and complex in their underlying structure, semantics can help musicians and producers in decision processes, and provide more natural interactions with the systems. Moreover, employing these technologies in order to create detailed metadata describing music productions can aid music producers to recreate workflows and musicologists to analyse trends in music production. Ontology-driven knowledge management in music production has also been discussed in [1, 2].

Within this field, the main focus of our study is the representation of information about audio effects, which play an integral part in modern music production. They modify an input signal and may be applied in order to enhance the perceived quality of a sound or to make more drastic changes to it in the composition process. The Audio Effects Ontology presented in this paper is designed as an extension to the Music Ontology [3] and Studio Ontology [1], as

well as our previous work detailed in [4, 5]. Following the discussion of the Studio and Audio Effects Ontology we show data produced with the help of these ontologies in a real world production scenario. Finally, discuss the results of a qualitative evaluation based on lexical coverage, before outlining directions for future work.

2 The Studio Ontology

The Studio Ontology is an OWL ontology for capturing the nuances of record production by providing an explicit, application and situation independent conceptualisation of the studio environment [1]. It is presented as a modular framework of ontologies, which in turn are built on the Music Ontology framework [3] and its components. It uses its core elements that allow for the representation of time-based events (Event and Timeline ontologies), and the workflow of music production in an editorial context subsumed under broader terms defined by the Functional Requirements for Bibliographic Records (FRBR) [6].

The Music Ontology allows for describing the music production workflow from composition to delivery, however, it lacks some concepts to do so in sufficient detail. The Studio Ontology provides some of the necessary extensions that form the foundation for a comprehensive representation of audio effects, and their application in music production. Here, we outline only those of its features and components which make it suitable as basis for our work.

- **Foundational components**: The Studio Ontology allows for characterising and describing the application of technological artefacts (devices) in music production. The Device Ontology provides a fundamental device and device decomposition model, and entities for representing device states, such as variable device parameters at different levels of granularity.
- **Complex device descriptions**: The ontology provides a model for describing complex devices such as signal processing tools and their interconnections. It includes a four-layered abstract model of these devices resembling the FRBR model.
- **Core components**: The ontology provides a model for describing recording studios on the editorial level (e.g. personnel and available equipment). It also provides for describing signal processing workflows in the studio using a parallel event and signal flow utilising music production tools.
- **Domain specific extensions**: The Studio Ontology supports the provision of domain specific extensions. It also provides some extensions for describing audio recording (e.g. microphones), mixing, editing and a core model for describing audio effects.

The application of audio effects to signals can be described using the concept *studio:Transform* defined by the Studio Ontology. This concept represents an event that takes a signal as a factor, and produces a transformed signal. This concept may be subsumed in more specific effect ontologies. The Studio Ontology sets

aside the problem of defining specific audio effects, their classifications, parameters and their application specific descriptions. The Audio Effects Ontology fills this gap.

3 The Audio Effects Ontology

The Audio Effect Ontology consists of harmonised modules that integrate seamlessly, where the main ontology defines the vocabulary. It defines an audio effect as an abstract concept corresponding to a physical phenomenon. For instance, the phenomenon associated with reverberation is the reflection of sound waves from surrounding surfaces. Other digital effects, as for example digital distortion by the reduction of the bit depth, do not have a naturally occurring counterpart. However, its acoustical effect is still conceptualised as a phenomenon independent from the DSP technique used to achieve it. Thus, the layer of acoustical phenomena as abstract concepts in our ontology, is comparable to the level in the FRBR layer representing products of intellectual work. A class specifying the technique employed to implement an effect can been seen as analogous to *expression* in the FRBR model. Since we focus here on digital implementations the technique consists of an algorithm in our case. For instance, a reverberation effect may be implemented by convolution with an impulse response or based on Schroeder-Moorer circuitry [7]. The manifestation of an effect takes the form of a specific effect implementation, such as a given plugin. A specific plugin instantiated in a DAW corresponds to *item*.

In the design of the Audio Effects Ontology we took into consideration existing ontologies, particularly the Device Ontology, part of the Studio Ontology [1]. Indeed, several concepts of the Audio Effects Ontology are subclasses or subproperties of existing classes and subproperties of the Device Ontology. The relationships of the conceptual models of FRBR, the device ontology and the audio effects ontology are visualised in Figure 1. All audio effects are subclasses of the class `afx:Fx`, which in turn is a subclass of `device:Phenomenon`.

The main ontology defines several concepts for the detailed description of an audio effect implementation. Among these descriptors are the implementation name, version number and the plugin standard (e.g. VST or AudioUnit). Using concepts of the FOAF Ontology [8] the developer can be specified as well. The Audio Effects Ontology consists of the modules:

- **Main ontology** - provides for the description of technical specifications for a given effect implementation, and the application of audio effects. Furthermore, it can be used to describe parameters and parameter settings of audio effects and their application in a music production project.
- **Parameter Ontology** - the Parameter Ontology includes a taxonomy which classifies parameter types. This enables agents to retrieve audio effects and audio effect implementations based on the types of parameters associated with them. Moreover, it facilitates the identification of similarities between audio effects based on parameter types.

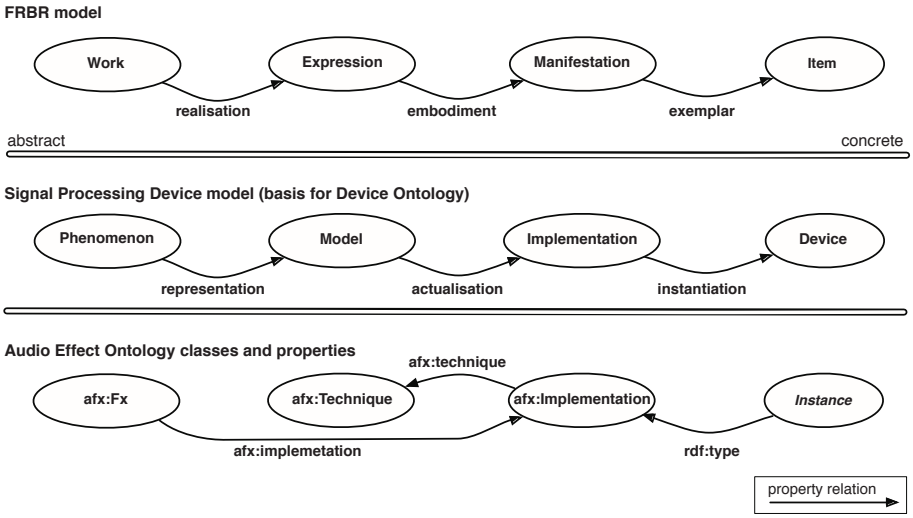

Fig. 1. Entities and relationships of the Device Ontology with reference to the FRBR model [9] and respective classes and properties of the Audio Effects Ontology

– **Effect classification** - different classifications of audio effects are defined in individual ontology modules.

  - **Perceptual classification** - contains a subclass-structure defining an effects classification based on auditory perceptual attributes. The hierarchy is based on the classification proposed in [10, 11] and the results of our listening test for the evaluation of the taxonomy [12].
  - **Technical classification** - an ontology module including a taxonomy based on technical aspects of the effects (e.g. delay, modulation, dynamics processing).

3.1 Effect Implementations

In order to describe signal processing devices we define a separate `afx:Implementation` class. This class represents the connection to the Studio Ontology since it is an equivalent class to the `studio:EffectPlugin` class. We define a property `implementation_of` to link an implementation to its effect type, which is the inverse property of `afx:implementation` shown in Figure 1.

$$\forall I(\text{afx:Implementation}(I) \rightarrow \text{studio:EffectPlugin}(I)) \quad (1)$$

$$\forall (F, I)(\text{afx:implementation\_of}(I, F) \rightarrow \text{afx:Fx}(F) \quad (2)$$
$$\wedge\ \text{afx:Implementation}(I))$$

An additional class, `afx:Format` conceptualises the type of implementation. These include various plugin formats, such as VST, AudioUnit and LV2[1], as well as a stand-alone implementation. The Audio Effects Ontology can be used for two main purposes: the description of digital audio effects implementations in order to create a database of effect software which gives detailed information about their specification, and the description of the application of effects in a music production project. We chose this design in order to facilitate expressing that a given implementation may be available in different formats. This information helps identifying implementations that are compatible with a given host software. The property `afx:available_as` can be used to express the formats in which a given effect is available.

Table 1 shows some of the classes and properties for the description of an audio effect implementation using the Audio Effects Ontology. In order to specify provenance information such as the product name and the names of the developers we use the properties `dc:title` and `dc:creator` respectively from the Dublin Core Ontology. Including information about developers and companies associated with an effect implementation facilitates not only the retrieval of effects developed by a given company or developer, but also enables management of information regarding intellectual property. Other concepts are defined in the Audio Effects Ontology, such as the supported operating system, the effect type, and the algorithm that forms the basis for a given implementation. While it would be possible to define the different plugin formats and operating systems in the form of individuals, we build a hierarchy of classes, such as *VST*, *AudioUnit*, *LV2*. This allows for further specification, e.g. version numbers of plugin APIs and operating systems, without the need for including one individual for each version (for instance VST 1.0, VST 2.0).

Table 1. Some of the concepts and properties for the description of an audio effect implementation.

Concept	Property	Range	Some subclasses/individuals
Product name	dc:title	*literal* (xsd:string)	-
FX format	available_as	Format	Vst, Au, Lv2, Rtas
Operating system	os	Os	Windows, MacOS, Linux
FX type	implementation_of	Fx	Chorus, Bandpass
FX technique	technique	Technique	SchroederMoorer, PhaseVocoder
FX parameters	parameter	Parameter	NumParameter, IndexedParameter
FX preset	preset	Preset	-
Audio inputs	audio_inputs	*literal* (xsd:int)	-

[1] Linux Audio Developer's Simple Plugin API version 2, http://lv2plug.in/

An important factor in the description of an effect implementation is the precise representation of the parameters that control the plugin settings. We distinguish two types of parameters: numerical parameters and indexed parameters. The former are parameters that have a numerical range. These are represented by the class `afx:NumParameter`. Each parameter has an identification number, which is given by a literal of type `xsd:int`. In addition to the parameter name we can specify the parameter range and default value for each parameter. Numerical parameters can have different units, for instance, a gain parameter may be given in decibel, a delay time parameter in milliseconds, and a modulation frequency parameter in Hertz. An example for an indexed parameter is a menu selecting a specific filter type, where the parameter values are selected from an indexed list of words such as *low pass*, *high pass* and *band pass*. We discuss these parameter type further later in this section.

A goal in the development of the Audio Effects Ontology is to facilitate the comparison and retrieval of similar audio effects, and the representation of specified effect settings. Let us assume we have two implementations of an echo effect, one using the unit seconds, the other the unit milliseconds for the delay time parameter. If we would represent the parameter values as simple float values without any further description it would be impossible to compare parameter ranges and default values of different implementations, since an agent could not retrieve sufficient information. Therefore, we use concepts of the Quantities, Units, Dimensions and Data Types (QUDT) ontology[2]. It provides the class `qudt:QuantityValue` for the representation of quantities. The unit for the quantities can be specified by linking the value to a unit using the property `qudt:unit`.

$$\forall Q, U(\text{qudt:unit}(Q, U) \rightarrow \text{qudt:QuantityValue}(P) \wedge \text{qudt:Unit}(U)) \qquad (3)$$

The Audio Effects Ontology provides subproperties of `qudt:numericValue`, which allows setting instances of `qudt:QuantityValue` to a desired scalar. The subproperties distinguish the three values: minimum, maximum and default. Although the QUDT ontology already contains a large number of unit types, we reuse the ontology and create subclasses for missing units that are audio or music specific. For example, we define `afx:Cent` as an individual of the class `qudt:Unit` to represent the unit *cent*, which describes a 100th of a semitone in equal temperament.

We define another property that directly links a parameter to a unit. This simplifies querying for the unit used for a particular parameter. Without this additional property the unit could only be retrieved by querying for the unit of the minimum, maximum or default value, i.e. in this case these values would have to be defined to retrieve the information. However, since the parameter unit can be assumed to be the same unit as that of the minimum, maximum and

[2] http://www.qudt.org/

default value, the property *afx:unit* is defined with a property chain axiom.

$$\begin{aligned}&\forall I, P, Q, U(\text{afx:Implementation}(I) \wedge \text{afx:Parameter}(P) \\ &\wedge \text{afx:QuantityValue}(Q) \wedge \text{qudt:Unit}(U) \wedge \text{afx:has\_parameter}(I, P)) \\ &\wedge \text{qudt:unit}(Q, U) \wedge (\text{afx:minimum\_value}(P, Q) \\ &\vee \text{afx:maximum\_value}(P, Q)) \vee \text{afx:default\_value}(P, Q)) \\ &\rightarrow \text{afx:unit}(P, U))\end{aligned} \tag{4}$$

By storing information about the unit associated with a parameter it is possible to convert values which enables the precise comparison of effect parameters of different implementations. As mentioned above, not all parameters are set with values within a given range, but instead may appear as a list of indexed text strings that for instance are selected from a pull down menu. In this case we cannot use the above mechanism to describe the parameter efficiently in RDF. Instead, we introduce a class `afx:IndexedParameter` and associate a group of the associated choices. The number of values for a given parameter that contains a list of values is precisely defined in an implementation. Therefore we make use of RDF *collections*. An RDF collection is defined as a list structure and makes it possible to describe a group of resources that contains specified members only. An example describing a parameter is expressed using a collection as shown in Listing 1.

Listing 1. Description of an *indexed parameter* using an RDF collection in N3 syntax.

```
:exParameter afx:indexed_parameter_values (
  [ a afx:IndexedParameterValue ;
    rdf:value "0"^^xsd:int ;
    rdfs:label "Option 1" ]
  [ a afx:IndexedParameterValue ;
    rdf:value "1"^^xsd:int ;
    rdfs:label "Option 2" ]
) .
```

Digital effect implementations make it possible to store configuration of parameter settings, which is commonly referred to as a *preset*. Loading a preset will instantly set all parameters to the configuration previously saved. A collection of presets stored in a single file is referred to as a *preset bank*. Preset banks may consist of presets that are designed for a specific musical purpose or character, or created by the same author. Many digital effect implementations have built-in presets that are stored in a *factory bank*. Additionally, it is possible to export user-generated presets and preset banks as files. The Audio Effects Ontology provides for the description of such preset banks and presets that are

associated with effect implementations. In addition to the fact that this leads to a more complete description overall, it also facilitates the transfer of settings from one effect implementation to another of a similar type by retrieving and processing RDF data. Currently, there are several file formats for storing effect plugin presets.

While plugin standards like VST and AU provide their own preset format, effect implementations may also define proprietary formats for their presets. We argue, that in addition to the sharing of information about plugins, the import and export of RDF data within audio processing software regarding parameter settings would be highly beneficial for ensuring interoperability. In the Audio Effects Ontology presets are conceptualised using the class `afx:Preset` and linked to implementations by the property `afx:preset`. Presets that are part of a preset bank are associated with the bank with the property `afx:bank_preset`. We define these entities as follows:

$$\forall P(\text{afx:Preset}(P) \rightarrow \exists I(\text{afx:Implementation}(I) \wedge \text{afx:preset}(I, P)) \quad (5)$$

$$\forall I, P(\text{afx:preset}(I, P) \rightarrow \text{afx:Implementation}(I) \wedge \text{afx:Preset}(P) \quad (6)$$

$$\forall B, P(\text{afx:bank\_preset}(B, P) \rightarrow \text{afx:PresetBank}(B) \wedge \text{afx:Preset}(P) \quad (7)$$

The class `afx:PresetBank` has subclasses for factory banks and user generated banks.

$$\forall F(\text{afx:FactoryPresetBank}(F) \rightarrow \text{dafx:PresetBank}(F)) \quad (8)$$

$$\forall U(\text{afx:UserPresetBank}(U) \rightarrow \text{dafx:PresetBank}(U)) \quad (9)$$

The relationship between an implementation and a preset bank for an implementation is expressed by using the property `afx:preset_bank`. If an implementation is associated with a given preset bank holding a given preset, it follows that the preset itself is also directly associated with the implementation:

$$\forall I, B, P(\text{afx:Implementation}(I) \wedge \text{afx:PresetBank}(B) \wedge \text{afx:Preset}(P) \quad (10)$$
$$\wedge \text{afx:preset\_bank}(I, B) \wedge \text{afx:bank\_preset}(B, P) \rightarrow \text{afx:preset}(I, P))$$

To express this in our ontology we make use of a feature introduced in OWL 2, i.e. the property `owl:propertyChainAxiom`. It allows us to define an axiom which states that the property chain `afx:preset_bank` followed by `afx:bank_preset` is spanned by the property `afx:preset`. In OWL 2 this is expressed as shown in Listing 2.

3.2 Audio Transformations in Music Production Projects

In the previous section we described how the Audio Effects Ontology can be used to describe digital audio effect implementations for the creation of database that can be published on the Semantic Web or used in a local database for the retrieval of information about audio effect implementations that exist in a given studio. In the following, we detail another purpose of the proposed ontology,

Listing 2. Description of classes and properties for presets and preset banks in N3 syntax.

```
afx:Preset a owl:Class .
afx:PresetBank a owl:Class .

afx:preset_bank a owl:ObjectProperty ;
       rdfs:domain afx:Implementation ;
       rdfs:range afx:PresetBank .

afx:bank_preset a owl:ObjectProperty ;
       rdfs:domain afx:PresetBank ;
       rdfs:range afx:Preset .

afx:preset a owl:ObjectProperty ;
       rdfs:domain afx:Implementation ;
       rdfs:range afx:Preset;
       owl:propertyChainAxiom ( afx:preset_bank afx:bank_preset ) .
```

which is the description of audio transformations in a music production project. The main motivation for this functionality is to be able to create detailed metadata about a given production for archives and reproducibility of workflows. To enable this functionality, the Audio Effects Ontology defines concepts for describing the application of effects to a signal. The concepts integrate seamlessly into the Studio Ontology that provides the class `studio:Transform` for the conceptualisation of audio transformations. Thus, a signal flow can be described as shown in Figure 2. A *transform* can be linked to an effect implementation using the `studio:effect` property of the studio ontology.

Using this mechanism we can express the following when storing information about an applied transform:

- A detailed description about the effect implementation involved in the transformation
- Exact information of the device state at the time of the transformation
- Linking the effect to high level semantic descriptors concerning audio effects

In order to describe a transform we introduce additional properties, while some aspects can be related to relevant concepts using properties we described previously in the context of expressing attributes of an effect implementation. For instance, while we use the property `afx:preset` to express that a certain preset exists for a given effect implementation the Audio Effects Ontology includes the property `afx:preset_used`. This enables to state that a given preset has been used to execute the transformation of an audio signal at a given time. Moreover, we are able to associate events on the audio signal timeline as defined by the Music Ontology to a particular transform and its parameters.

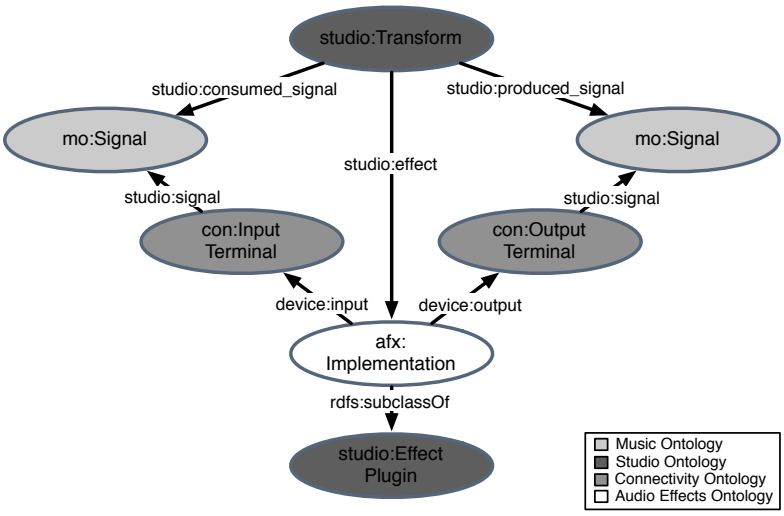

Fig. 2. Transformation of an audio signal described using the Studio Ontology and Audio Effects Ontology.

This implementation class may also act as the connection to the `afx:Fx` class specifying the effect type with the property `afx:implementation` (an inverse property linking an implementation to an effect type is also given by `afx:implementation_of`).

The property `afx:implementation_version` can be used to further describe which plugin format, i.e. which implementation type, has been used in a particular case for describing a specific application of an effect. For instance, a plugin may be available as a VST and AudioUnit plugin, which may differ in their implementation. If the implementation type would be defined as a subclass of the implementation, a given audio effect implementation would be inextricably linked to one implementation type. Instead, conceptualising the implementation independently makes it easier to cover the two use cases described above. In other words, we are able to express that a certain effect implementation has been used independently of the implementation type.

In music production it is common practice to automate effect parameters, i.e. the parameters of a given effect may change over time. Modern DAWs are able to store the automation data for this purpose. In order to represent changing effect parameter values the Audio Effects Ontology provides the class `afx:State` which represents a similar concept as the `device:State` class, proposed in the context of *consolidated reification* in [9]. It conceptualises variable attributes and relationships of an audio effect. In our case these consist of changes of parameter values over time. The `afx:State` class is a subclass of `event:Event`. Therefore, we can describe a region on a signal timeline in which a certain parameter setting is true. The region is defined by an *entry* and *exit* point. These are subclasses

of `tl:Instant` of the Timeline Ontology. Using this mechanism we can describe the device states during a given transform accurately.

3.3 Describing Audio Effect Transformations

In the previous section 3.2 we described the concepts of the Audio Effects Ontology concerned with the description of audio transformations involving audio effects. Since the application of audio effects is an essential part in modern audio and music production, it is desirable to document this aspect of the production workflow accurately. A database of such detailed data facilitates the analysis of workflows, for example for the comparison of different productions, and ensures reproducibility of workflow steps.

The example shown in Listing 3 includes the term `:exFx` referring to a feedback delay implementation, which in turn may be linked to a detailed effect description as described in Section 3.1. Furthermore, in the example the transform is applied in a time interval ranging from 60s to 120s. The setting for the parameter *delay time* changes at the 90s point from 180 to 90. If an unaltered preset would have been used instead, stating the individual parameter values would be redundant since these are described in the preset description. In this case the preset would be linked by an effect state via the property `afx:preset_used`.

4 Evaluation

We evaluated the Audio Effects ontology using a data-driven approach proposed by Brewster et al. [13], which has also been used in [14] and [9]. We take into account a text corpus generated from the KVR Audio Product database[3] to establish how well our ontology fits lexically to the domain. In addition to the derivation of a measure for the lexical coverage we manually map frequently occurring words in our data set to ontology concepts.

For a data data-driven approach we use a data set from a text corpus consisting of product descriptions of 3109 individual effect plugins. The texts were obtained from entries in the KVR database by extracting the respective descriptions from each website dedicated to a given product. These feature descriptions fit well into the domain of discourse of our ontology. Since it is not aimed at software developers that are concerned with implementation techniques, they describe the digital audio effect focussing on practical use in the studio. Other product descriptions are given in prose, including more detailed descriptions of implemented features or examples of use cases, for instance:
The text corpus consists of 431582 words in total. After removing numbers, stop words[4], product names and company names, we used the Porter Stemmer algorithm [15] to reduce inflected and derived words to their stems. This process

[3] http://www.kvraudio.com/q.php
[4] http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a11-smart-stop-list/english.stop

Listing 3. Description of the device state of an example effect during an audio transformation produced by the application of a feedback delay plugin implementation in N3 syntax. The parameter IDs, types and units are: 1, delay time, milliseconds; 2, lowpass filter, Hertz.

```
:exTransform a studio:Transform ; studio:effect :exFx .
:exFx afx:state :exState1, :exState2 .

:event1 a event:Event ; event:time [ a tl:Instant ;
       tl:timeline :exampleTimeLine ;
       tl:at "60.0s"^^xsd:duration ].
:event2 a event:Event ; event:time [ a tl:Instant ;
       tl:timeline :exampleTimeLine ;
       tl:at "90.0s"^^xsd:duration ] .
:event3 a event:Event ; event:time [ a tl:Instant ;
       tl:timeline :exampleTimeLine ;
       tl:at "120.0s"^^xsd:duration ] .

:exState_1 afx:parameter
       [ a afx:Parameter ; afx:parameter_id "1"^^xsd:int ;
        qudt:value :par1Value1 ] ,
       [ a afx:Parameter ; afx:parameter_id "2"^^xsd:int ;
        qudt:value :par2Value ] ;
       afx:entry :event1 ; afx:exit :event2 .

ex_State_2 afx:parameter
       [ a afx:Parameter ; afx:parameter_id "1"^^xsd:int ;
        qudt:value :par1Value2 ] ,
       [ a afx:Parameter ; afx:parameter_id "2"^^xsd:int ;
        qudt:value :par2Value ] ;
       afx:entry :event2 ; afx:exit :event3 .

:par1Value1 qudt:numericValue "180"^^xsd:double .
:par2Value qudt:numericValue "5000"^^xsd:double .
:par1Value2 qudt:numericValue "90"^^xsd:double .
```

yielded 246544 stemmed words with 8023 unique stems. In order to broadly approximate the coverage of our ontology we attempt to map the most frequently occurring words in our text corpus with concepts defined in our ontology. We take in to account ontology terms from the Music Ontology including its extensions, namely the Studio, Audio Effects and Audio Features Ontology (the QUDT Unit Ontology is included by the Audio Effects Ontology). Table 2 shows the results for the 50 most frequently occurring terms. We can assign an ontology concept to 38 of these terms, of which 20 are concepts that are added by the inclusion of the Audio Effects Ontology. Basic terms for the description of a recording, such as *sound*, *signal*, *recording* and *channels* are covered in the

Music Ontology. Some lower level concepts are already given in the Studio Ontology modules, for instance for input and output connectors in the Connectivity Ontology. As mentioned above, the described mapping of ontology terms only gives a rough idea of the coverage of the ontology. Hence, in the next section we measure our ontology on the lexical level and compare it against existing ontologies.

Table 2. Top 50 words in the data set and the ontology mapping to terms of the Music Ontology with Audio Features, Studio and Audio Effects Ontology extensions.

Term	Count	Ontology mapping	Term	Count	Ontology mapping
control	3674	`afx:Parameter`	includ	1138	-
sound	3073	`mo:Sound`	output	1137	`con:OutputTerminal`
effect	2772	`afx:Fx`	mode	1136	-
featur	2626	-	eq	1084	`class3:Equaliser`
filter	2529	`class3:Filter`	preset	1065	`afx:Preset`
plug	2280	`afx:Plugin`	gain	1011	`par:Gain`
audio	1793	`mo:Signal`	user	983	-
compress	1754	`class3:Compressor`	design	958	-
signal	1748	`mo:Signal`	track	951	`mt:Track`
band	1743	`par:Bandwidth`	paramet	934	`afx:Parameter`
frequenc	1705	`par:Frequency`	reverb	916	`class3:Reverb`
modul	1653	-	sampl	911	-
time	1586	`mo:Time`	db	899	`unit:Decibel`
stereo	1572	`mo:channels`	record	862	`mo:Recording`
midi	1387	`mt:MIDICommand`	bit	849	`unit:Bit`
high	1369	-	vst	841	`afx:VST`
set	1369	`afx:parameter`	model	836	`afx:Technique`
mix	1341	`studio:Mixing`	support	806	-
delay	1328	`class3:Delay`	creat	776	-
process	1293	`studio:Transform`	phase	772	`par:Phase`
channel	1256	`mo:channels`	meter	764	`studio:VUMeter`
input	1239	`con:InputTerminal`	pitch	757	`af:Pitch`
low	1239	-	make	712	-
level	1225	`af:RMSAmplitude`	limit	709	`class3:Limiter`
adjust	1180	`afx:parameter`	algorithm	703	`afx:Technique`

To measure the lexical coverage of our ontology we compare it against the combined text corpus of the product descriptions. First, we retrieve all labels, comments and descriptions from our ontology to produce a combined text that describes the content of the ontology. We then create a matrix of vectors consisting of *tf-idf* [16] values. Each vector represents one document. Finally, we calculate the cosine distance between the vectors. The *tf-idf* is defined as the product of the term frequency (the number of times a word appears in a given document) and the inverse document frequency (a measure of whether a term is

common or rare across all documents):

$$\mathrm{TFIDF}(i,j) = \mathrm{TF}(i,j) * \mathrm{IDF}(i) \tag{11}$$

In addition to the product descriptions and the Audio Effects Ontology with its classification and parameter modules we include in our vector space the Music Ontology with the Audio Features Ontology (MO), the Studio Ontology (STUDIO) and the LV2 Ontology (without API-specific modules). We calculated the cosine similarities between the vector corresponding to our data set and each of the other vectors constructed from the ontologies. The resulting inter-document similarities are shown in Table 3.

Table 3. Cosine similarities between vectorised ontology labels, comments and descriptions, and the KVR audio effects product descriptions dataset.

Ontology	Similarity
LV2	0.0511
MO+STUDIO	0.1686
AFX	0.1723
MO+STUDIO+AFX	0.1916

The numbers show that the Audio Effects Ontology on its own performs better than the LV2 Ontology. This is due to the fact that it encompasses a wider range of information. For instance, the LV2 Ontology is does not deal with audio transforms. It is part of the LV2 API and is specialised for LV2 development. Its classes specifically define LV2 plugins, and not abstract concepts of effects. Furthermore, the Audio Effects Ontology includes concepts for more audio effect types and parameters. We also measure the improvement of the lexical coverage when the Audio Effects Ontology is combined with the existing Music and Studio ontologies. In our vector space the cosine similarity of these combined ontologies increases by 15.7% compared to the combination of the Music and Studio ontologies only. The Audio Effects Ontology standing on it own also measures higher than the combination of the Music and Studio ontologies. Nevertheless, these two ontologies exhibit a higher similarity than the rather narrow LV2 ontology.

The results confirm that including the Audio Effects Ontology results in the closest fit with our corpus. However, this does not give any information about the expressivity and applicability of the ontology. Therefore, we also evaluated our ontology in a task based approach, testing it in a number of use cases and proposed applications. For instance, in [4] we propose a new class of audio effects that uses high-level semantic audio features in order to obtain control data for multitrack effects. The metadata is expressed in RDF using several music and audio related Semantic Web ontologies, including the Studio and Audio Effects ontologies. The metadata is retrieved using the SPARQL query language[5]. In

[5] SPARQL is a recursive acronym for SPARQL Protocol and RDF Query Language.

[5] we show how the Audio Effects Ontology can be used to retrieve detailed metadata about audio effect settings from an RDF database describing music production projects.

5 Conclusions

In this paper we discussed the creation of metadata for music production projects using Semantic Web ontologies with focus on the application of audio effects. We have shown that by employing the Audio Effects Ontology it is possible to represent detailed metadata about the application of audio effects in music production workflows. This allows for the analysis and comparison of musical works with regards to production workflows.

In order to exploit the full potential of the ontology it is necessary to develop applications that support the ontology. Such applications may be in the form of DAW that are capable of capturing and retrieving of information concerning the use of audio effects in the music production process. Thus the ontology should be used by software developers to directly integrate its functionality into novel software applications, whereby the Semantic Web technologies such as query languages and RDF representations are hidden in favour of a user-friendly GUI.

The advantages of using Semantic Web technologies over employing other metadata standards are manifold. For instance, interoperability between agents is limited when relying on proprietary formats. RDF represents knowledge by using defined ontologies and unique identifiers, as opposed to simply data. It permits publishing on the Semantic Web, and supports reasoning and inference. However, retrieving metadata by querying RDF graphs is comparably slow – a problem that needs to be addressed for the realisation of applications using the ontology. Further advances in computer technology, such as more powerful processors, can alleviate this problem. Nevertheless, research in the modes of operation of specialised software agents is key.

References

1. Fazekas, G., Sandler, M.B.: The studio ontology framework. Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011) (2011)
2. Barkati, K., Bonardi, A., Vincent, A., Rousseaux, F.: GAMELAN: A knowledge management approach for digital audio production workflow. In Mercier-Laurent, E., ed.: Proceedings of Artificial intelligence for Knowledge Management Workshop (AI4KM) of ECAI. (2012)
3. Raimond, Y., Abdallah, S., Sandler, M., Giasson, F.: The music ontology. Proceedings of the International Conference on Music Information Retrieval (2007)
4. Wilmering, T., Fazekas, G., Sandler, M.B.: High level semantic metadata for the control of multitrack adaptive audio effects. presented at the 133rd Convention of the AES, San Francisco, USA (2012)
5. Wilmering, T., Fazekas, G., Sandler, M.B.: The audio effects ontology. Proceedings of 14th International Society for Music Information Retrieval Conference (ISMIR 2013), Curitiba, Brazil (2013)

6. Plassard, M.F., ed.: Functional Requirements For Bibliographic Records : final report / IFLA Study Group on the Functional Requirements for Bibliographic Records. Volume 19. K.G. Saur (1998)
7. Schroeder, M.R.: Natural sounding artificial reverberation. Journal of the Audio Engineering Society **10**(3) (1962) 219–223
8. Dodds, L.: An introduction to foaf. [Online] available: http://www.xml.com/pub/a/2004/02/04/foaf.html (2004)
9. Fazekas, G.: Semantic Audio Analysis - Utilities and Applications. PhD thesis, Queen Mary University of London (2012)
10. Verfaille, V., Guastavino, C., Traube, C.: An interdisciplinary approach to audio effect classification. Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06), Montreal, Canada (September 2006)
11. Verfaille, V., Holters, M., Zölzer, U.: Introduction. In Zölzer, U., ed.: DAFX - Digital Audio Effects. 2nd Edition, J. Wiley & Sons (2011)
12. Wilmering, T., Fazekas, G., Sandler, M.B.: Audio effect classification based on auditory perceptual attributes. presented at the AES 135th Convention, New York, NY, USA (2013)
13. Brewster, C., Alani, H., Dasmahapatra, S., Wilks, Y.: Data driven ontology evaluation. Proceedings of the International Conference on Language Resources and Evaluation, Lisbon, Portugal (2004)
14. Raimond, Y.: A Distributed Music Information System. PhD thesis, Queen Mary, University of London (November 2008)
15. Porter, M.F.: An algorithm for suffix stripping. Program: Electronic Library and Information Systems **14**(3) (1980)
16. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, Inc. New York, NY, USA (1983)