NOVEL METHODS IN INFORMATION MANAGEMENT FOR ADVANCED AUDIO WORKFLOWS

György Fazekas and Mark Sandler

Centre for Digital Music,
Dept. of Electronic Engineering and Computer Science
Queen Mary University of London, UK
gyorgy.fazekas@elec.qmul.ac.uk

ABSTRACT

This paper discusses architectural aspects of a software library for unified metadata management in audio processing applications. The data incorporates editorial, production, acoustical and musicological features for a variety of use cases, ranging from adaptive audio effects to alternative metadata based visualisation. Our system is designed to capture information, prescribed by modular ontology schema. This advocates the development of intelligent user interfaces and advanced media workflows in music production environments. In an effort to reach these goals, we argue for the need of modularity and interoperable semantics in representing information. We discuss the advantages of extensible Semantic Web ontologies as opposed to using specialised but disharmonious metadata formats. Concepts and techniques permitting seamless integration with existing audio production software are described in detail.

1. INTRODUCTION

Loss of information in the media production workflow chain is a major issue when it comes to collecting, managing and repurposing metadata about various aspects of the production process or the media under consideration. This is especially true in the world of editing and mastering audio.

Prominent reasons for collecting musical metadata include the effective organisation of musical assets, such as sounds from a sample library or previously recorded takes of multitrack master recordings. These examples represent cases of content-based information management and retrieval.[1] Another distinct category can be designated as metadata extraction for creative applications.[2] Often, this is overlooked by developers of metadata standards.

Several musical applications of these categories are discussed in the literature (see [1], [2], [3], [4], or [5] for examples) focusing on specific case-based implementations. However, no generic formats and practices emerge from previous research and industrial solutions. Disjoint purposes for covering overlapping musical domains produce a plethora of disharmonious standards and methods. This seriously impairs the exploitation of metadata in developing ubiquitous creative applications. A part of this problem can be identified in the use of non-normative development and publishing techniques, rather than flaws in design. We recognise that common approaches to overcome this, including standardising syntax through the use of XML[3] or creating a reference library (such as those accompanying SDIF[4] or AAF[5]) do not provide sufficient ground for modularity and interoperability. The heart of the problem lies in the assumption that musical information can be expressed using structured static data sets. In our research, we challenge this general assumption and hypothesise that musical information is better modelled as dynamic semi-structured data.

Because of corresponding needs in representing diverse, virtually unbounded information, we turn to Semantic Web technologies for data modelling and knowledge engineering techniques. In particular, we use the Resource Description Framework (RDF) as our fundamental information model, and Semantic Web ontologies as our basis domain model. Existing tools for building the Semantic Web however are not designed for the use cases and requirements of audio and multimedia signal processing. They typically fall short in efficiency, because of the vast amount of data required for encoding content-based audio descriptors, and language compatibility, because of the dominance of scripting and Java based solutions used in web tools, as opposed to efficient C++ implementations common in audio processing.

Our primary motivation is in creating software tools for easy adaptation and use of the Semantic Web information model in audio processing applications. For this purpose, we develop a software library which provides an ontology based information management solution and easy integration with music production software. Our library can be used as a shared, dynamically loaded object. It relies on efficient local database implementation, low-level RDF graph manipulation and automatic mapping between application data and metadata governed by ontology schema. Our proposed architecture therefore avoids the need for expensive query processing and the overhead of network communication.

2. METADATA FORMATS IN RELATED WORK

There is no conclusive previous work addressing the issues we described in the first section. However, we briefly review the most relevant methods and practices in representing metadata for music and audio processing. We also discuss some important problems associated with them.

Music related metadata can be grouped into four distinct categories by scope or the sub-domain they address. Our framework aims to capture and represent the following main metadata types in a unified way:

[1]This typically involves feature extraction followed by machine learning or data mining techniques.

[2]e.g. feature extraction followed by adaptive signal processing.

[3]eXtensible Markup Language

[4]Sound Description Interchange Format

[5]Advanced Authoring Format

- *Editorial Metadata:* General information about songs, artists and other participants.
- *Content-based Signal Descriptors:* Spectral and temporal characteristics of the recorded audio signal.
- *Processing Workflow Data:* Information about audio processing, effects and parameters.
- *Musicological Features:* Musically meaningful features such as notes, rhythm, key and chords.

In our brief review, we focus on the second and third categories since they present the most prominent problems we address in this work.

2.1. MPEG-7

The most widely used standard for content-based multimedia management is MPEG-7. It provides a comprehensive set of audio descriptors and enables applications such as retrieval from digital archives or filtering personal music collections. MPEG-7 *descriptors* and *descriptor schemes* are specified using an extended XML Schema Language. Although this was sufficient to express the structural and syntactical requirements of the standard, it does not provide a machine-understandable representation of the semantics associated with its schemes. Therefore, the ability to integrate or reuse descriptors or the metadata in other domains is severely limited. This led to efforts such as described by Hunter [6] or García [7] to provide mapping between MPEG-7 elements and Semantic Web ontologies. However, they neither offer an efficient solution, nor do they solve the fundamental problem of lack of semantics in the standard specification itself.

2.2. SDIF

The original aim of SDIF (Sound Description Interface Format) [5] was the efficient binary representation of frame-based spectral data in an analysis/synthesis framework. However, this was soon extended towards the inclusion of more complex types and higher level features. Although this format is available in existing audio applications, unfortunately its rigid specification is too specific to its original purpose. This compromises semantics and extensibility. Moreover, its binary representation is not well suited for searchable persistent database storage. Therefore this format can not ideally be used in a general information framework.

2.3. ACE XML

ACE XML is a collection of multi-purpose file formats developed for the jMIR package. It enables communication between its components such as the jAudio [8] feature extraction library and the ACE [9] classification engine. It provides Document Type Definitions (DTD) for storing audio features as well as training class labels and taxonomies for classification. Unfortunately the format wasn't developed further to cover a wider set of use cases or a wider range of audio features within a hierarchical relationship. It falls short compared to MPEG-7 even in representation and compactness with regards to multidimensional feature values. As a common problem with XML based formats, it permits somewhat ad-hoc definition of feature types without the possibility to establish meta-level relationships (such as equivalence) for the purposes of interoperability or the reuse of data expressed using this format.

2.4. IXD

The Integra Extensible Data (IXD) format was developed within the Integra project for the libIntegra [10] library. This software library provides interlinking and persistent storage facilities for audio processing and live composition environments (such as PD and Max/MSP) in a software independent manner. It is based around the concept of multimedia module encapsulating signal processing functionality. The IXD format appears to be well designed and well suited for representing module information for the purposes of storing module states and parameters in the local file system or in a remote database. It provides a good example of an XML-based schema which doesn't entirely overlook semantics and ontological considerations. For instance, it is possible to define hierarchical inheritance relations between modules. However, it is not possible to do so in defining module attributes, closely tied with module definitions in an object-orientated manner. Although the format moves in this direction, it does not permit the definition of semantic associations rich enough to be used in a more generic way.

2.5. AAF and the AES31-3 standard

The Advanced Authoring Format (AAF) [11] is in our concern because of its ability to describe media production workflows. It is the successor of Open Media Framework (OMF) and considered to be a superset of the AES31-3 standard. These formats deal with the relationship of signals within an audio processing application such as the clip composition of a project, and simple events such as fades, transitions, and process automation. AES31 was published in text, without the use of a definition language. Therefore, its semantics is highly intermingled with its syntax specification. AAF on the other hand provides a reference software library besides its textual specification.

This format specifies an object-orientated (OO) class model for interchanging audio-visual content as well as associated metadata. This maps well on the typical class hierarchy of application storage containers. For extensibility, the format specifies Meta-Classes which can be included in Meta-Dictionaries and sent along with an AAF file. Yet, the specification contains statically coded information such as media types. Therefore it fails to support a dynamically extensible system. Most extensions would require generating and recompiling source code. This does not permit easy adaptation to user's needs in situations such as describing new signal processing elements of a production system. The rigid data model consisting of closely joint objects and properties, and the typically used binary encoding, do not support the development of query interfaces to assist workflow management, nor does it support automated inferencing on persisted metadata. Building an interlinked database over various facilities would also be problematic. Finally, the supplied SDK confines monolithic, single language implementation. These drawbacks had led to the need for mapping AAF terms onto XML based vocabularies and data encoding [12] types. However, the format is still lacking semantic associations, therefore the afore mentioned problems are not fully solved.

2.6. Common Problems and a Solution

It is generally recognised that the use of metadata requires agreement on how it is produced, structured and interpreted. However, an important problem can be identified as the lack of normative methods in creating and publishing metadata formats. Describing a standard in a plain text document is equivalent merely to setting conventions on how data should be understood. Using an XML based schema language, such as XSD, is a significant step forward. However, it permits ad-hoc definition of metadata elements, and lacks the formalism of defining relationships between them. This hinders modularity and reusability of the published schema, and interoperability of the represented data. Although choosing a formalised syntax allows machine-readability and translation, it does not support automatic interpretation.

Ontology languages provide a solution by formalising the expression of semantics, that is, what we mean by certain concepts, and how we intend to relate them to other concepts in a domain. This permits modularity and extensibility of schema expressed using these languages, thus allows developers to focus on a field of interest without limiting the use of their specification outside of that domain.

The W3C[6] has issued a number of recommendations for representing formal ontologies. Since machine-processable representation of heterogeneous data on the Semantic Web is similar to the problem of representing diverse musical information, we argue that these technologies provide good common ground for representing data in music processing applications.

3. DATA MODEL

In this section, we review the data model we use, and the set of ontologies permitting the representation of main metadata types we listed in section 2.

3.1. RDF and the Semantic Web

Similarly to the general Web: an interlinked network of documents, the Semantic Web is a heterogeneous network of interconnected data and services. This network may only work if various disjoint data sets and services speak the same language. Thus, they have to follow some common data model or structured schema. The problem however is that the Web exposes unbounded, diverse information, making it hard, if not impossible to design this schema. Yet, the Semantic Web provides a surprisingly simple solution to this problem: the Resource Description Framework (RDF) [13].

RDF is a conceptual data model providing the flexibility and modularity required for publishing diverse semi-structured data—that is, just about anything on the Semantic Web. It is also the basis of more complex description languages, such as the OWL Web Ontology Language, which provides a way of publishing extensible data schema. The model is based upon the idea of expressing statements in the form of *subject – predicate – object*. These statements are also known as triples. A collection of triples can be seen as a graph, with nodes representing subjects and objects, and edges, representing predicates. Therefore, a large set of statements form a complex network of semantic relationships.

Elements of these statements are literals or resources named by a vocabulary of Unified Resource Identifiers (URI). This provides the model with an unambiguous way of referring to things, as well as a resource linking mechanism through the use of HTTP. RDF in itself does not specify a syntax for encoding information. While XML is a common serialisation format for RDF data, more compact and efficient representations exist such as the (both human and machine readable) N3 syntax.

[6]The World Wide Web Consortium (`http://www.w3.org/`)

3.2. Ontologies

Although RDF provides a fundamental data model, it does not have the facilities for expressing complex relationships required in domain modelling. In order to precisely communicate information in RDF statements, we have to be able to define and refer to concepts: such as a specific algorithm we use for audio processing, its concrete implementation and its parameters. We also need a vocabulary of well defined relationships existing in the application. For example, we link parameter values denoted as RDF literals with conceptual representation of the parameter itself. Ontologies are the tools for establishing these elements in a knowledge representation model. Building ontologies is therefore the process of Knowledge Engineering.

Semantic Web ontologies are created using the same conceptual framework that is used for communicating the data. However, additional vocabularies are required for expressing formal ontologies, as well as for improving machine interpretability. To this end, a hierarchy of languages is proposed by the W3C. This includes RDF Schema, for defining classes and properties of RDF resources, and OWL for making RDF semantics more explicit. Using OWL-DL[7] we can impose restrictions on the range and domain types of properties, and constraints on cardinality, the number of individuals linked by a property. In the next section, we review the ontologies we use as basis for our information model.

3.2.1. The Music Ontology

The Music Ontology (MO) is a a standard base ontology for describing music related information. It is described in [14] and formally specified in [15]. This ontology provides the main conceptual framework and a development model we use in our work. It is built on several ontologies specific to well-bounded domains. The four most important ones are the Timeline Ontology, the Event Ontology, the Functional Requirements for Bibliographic Records (FRBR) ontology, and FOAF [8]. We only review those most pertinent to our application, therefore the interested reader is advised to refer to the afore-mentioned documents. Besides reusing existing terms and ontologies, the Music Ontology provides ways of plugging new terms under existing concepts. This aspect is of primary importance in our application, since it permits compatible extensions.

3.2.2. Timeline Ontology

The Timeline Ontology [16] is conceptually derived from OWL-Time. It is used to express temporal information related to audio, including intervals and time instants, with possible references to multiple timelines with different origins. As an example, we may wish to relate a time based event occurring in an audio recording, such as a note onset, to an audio timeline relative to the start of the recording, or the universal physical timeline. The ontology also provides mapping between relative timelines, such as that of a continuous time signal and the regularly sampled version of it. (see figure 1.)

[7]A version of the Web Ontology Language based upon Description Logic which ensures computability in all circumstances.

[8]The Friend Of A Friend ontology represents relationships between people.

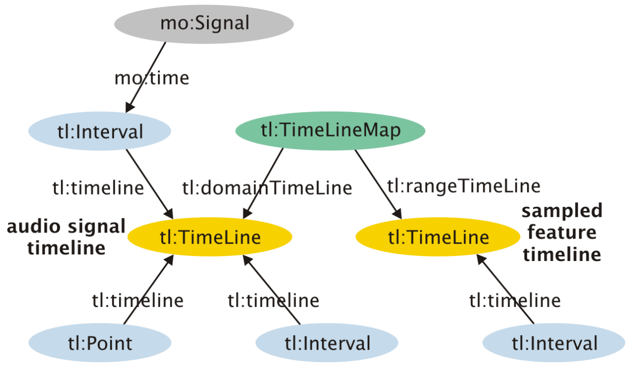


Figure 1: Using the Timeline Ontology

3.2.3. *Event Ontology*

Another important backbone of MO is the Event Ontology [17]. It is a broad conceptualisation of events: *'something, happening at a particular place and time'*. We can use this concept to describe anything with a well-defined duration, location, and a number of factors and agents. Therefore, it provides a very important link between terms in ontologies. As an example, we use this for associating effect automation parameters and acoustical features with the audio signal timeline.

3.2.4. *Audio Features Ontology*

The Music Ontology provides a wide range of musical concepts. This includes high level editorial data, production data about musical recordings, and finally structural information of music using the Event and Timeline Ontologies described above. However, it does not attempt to cover sub-domains that are too specific. These are addressed using extensions such as the Audio Feature Ontology (AF) and the Studio Ontology.

AF [18] provides an easily extended set of low-level and musicological features for modelling content-based audio descriptors. Most features are seen as time-based events, therefore they are derived from the Event concept. This concept is generic enough to be used in cases where a specific feature is not yet fully defined in the AF Ontology.

3.2.5. *Studio Ontology*

The Studio Ontology addresses music production environments. A set of domain specific ontologies are used for covering production details, for instance, the music editing workflow. This ontology includes the Multitrack Ontology, relating the internal structure of media production tools to broader concepts in the Music and Studio Ontologies, and the Edit Ontology, which provides the semantic framework for collecting information about the music production process.

The Multitrack Ontology defines a hierarchy of concepts for the representation of audio clips and tracks, and relates them to the general purpose signal concept in the Music Ontology. These constructs can be used in modelling the typical procedure of *nonlinear editing* in modern record engineering.

4. INFORMATION MANAGEMENT LIBRARY

In order to advocate advanced uses of metadata in audio processing, we develop a unified information framework and metadata collection tool. This tool can be easily integrated with existing audio production software. Our primary motivation is in the use of modular ontology schema, to overcome the limitations of currently existing disharmonious metadata formats.

So far, we have discussed the advantages of using Semantic Web ontologies, and introduced the ontologies we use. In this section, we describe the software architecture of the RDF based information back-end for managing metadata in audio and music processing applications.

4.1. Design Decisions

Although ontologies provide modularity in specifying metadata schema and flexible knowledge management, often, static software implementations limit the extensibility of a system. Among the most common problems we find the use of external (relational) database software, which is often accessed through hard-coded query templates. This limits the ability of a specific implementation to adapt to changes in metadata schema, besides, accessing the database involves expensive query processing. It is not uncommon that an object-based or RDF-based information system is used together with a relational database back-end. This incurs complicated mapping to relational schema which would otherwise be unnecessary.

Our system avoids these problems by relying on an efficient local hash database implementation, providing a native RDF store, and low-level manipulation of statements in an RDF graph. In order to abstract these low-level calls, we develop an automatic mapping mechanism between application objects and RDF statements. This system is able to persist application data, stored in an existing object hierarchy, together with semantic associations obtained from ontology definitions. The mechanism also provides atomic transaction management for groups of RDF statements associated with data from common application objects.

Since our framework is to be used as an efficient semantic metadata store, we avoid the overhead of network communication between the database and the host application. Therefore, we implement the system as a shared dynamically loaded library, compiled together with a database implementation.

Finally, our system is able to extend the application with storage containers for representing metadata terms defined in ontologies. This is achieved using a meta-object protocol and an associated type system. This is described in sections 4.2 and 4.3.

4.1.1. *Primary Requirements*

The most important requirements of the system can be summarised as follows:

- *Extension:* The system is be able to extend the application with metadata storage dynamically.
- *Mapping:* The system is able to translate and store application data represented in an existing object hierarchy.
- *Consistency:* The system maintains metadata and database consistency during interaction with a user.
- *Integration:* The system can be appended to an existing audio application with the least possible interference with original code.

4.1.2. Dependencies and Configuration

Our current aim is to build a reference implementation for existing audio applications written in C++, such as the open-source audio editor: Audacity[9]. This confines our choice of RDF tools to those with C/C++ support. As basis for our triple store implementation, we use the Redland RDF libraries. [20] This library permits in-memory or persistent storage. We configure the library to use an efficient hash database. For this purpose, we use BerkleyDB [10], a high-performance open-source embedded database solution distributed by Oracle. Our current implementation makes use of the cross-platform framework wxWidgets [11] for greater compatibility with Audacity. Ideally, it shall be using the Standard Template Library (STL). However, wxWidgets provides compatible classes which makes such transition relatively easy.

4.2. Metaobject Protocol

Considering the previously described use cases, (see section 4.1.1) finding an efficient way of accessing from or updating information to a triple store from an audio editor is a primary challenge. Using a local database which is dynamically loaded into the application, this can be achieved in two ways: using in-process API calls, or using SPARQL[12]. Because of the computational expense associated with query processing, we base our implementation on low-level API calls. However, besides the burden of manipulating the RDF store at a fairly low level, an important problem arises form conflicting data models: the class hierarchy of a typical object-orientated application, and the RDF graph. As a solution to this problem, we use a software engineering technique called metadata mapping. [13] We provide a mechanism which allows persisting application objects in an RDF database automatically. However, this requires the data in the application to be associated with ontological semantics. This is described in section 3.2. A further use case entails that we have to be able to instantiate objects, representing arbitrary metadata terms defined in an ontology. In our system, a run-time Metaobject Protocol (MOP) [21] provides the basis for the solution to both of these problems.

Metaobject protocols were originally developed for the Common Lisp Object System (CLOS) [22]. They can also be found in the context of more recent dynamic languages such as the Python interpreter[14]. Metaobject protocols allow for extensible association of data with semantics within an application. Therefore, the application is able to inspect the internal state of its objects. For our particular use cases, we do not need to implement the full protocol required for a dynamic interpreter. Yet, it is beneficial for blending functional and logic programming paradigms when managing metadata in an efficient but static programming environment.

In our library, we implement a metaobject protocol to facilitate metadata mapping between application data and RDF data. Two specific types of metaobjects are used for representing RDF classes and properties. These metaobjects are statically designed to represent information defined by ontology schema, however, they are dynamically instantiated by a generator via an inference mechanism when loading schema documents into memory. This is achieved using the following protocol: Ontology schema are parsed into a model using a suitable Redland syntax parser. Obeying RDF and OWL language rules, we build metaobjects for each class and property defined in the ontologies in question. First, we enumerate class and property declarations in the conjunctive model and instantiate a skeleton object for each. Next, we separately infer the inheritance hierarchy within the disjoint hierarchies of ontological terms and relationships. This information is appropriately used to model the same hierarchy within the set of previously created metaobjects. This is followed by assessing equivalence relationships and update the object model accordingly. Finally, the assignment of properties to classes in the model can be made. The metaobjects resulting from this process are stored in hash maps with keys corresponding to the resource URIs used for identifying them. These objects are available in the application and can be used to link data with semantics, and to create metadata containers according to their descriptions. In practice, this is achieved by constructing objects of a specialised type system we describe in section 4.3.

4.3. Type System

For the purpose of instantiating metadata containers as needed in the editing workflow, we develop a type system associated with the metaobject protocol and the basic node types appearing in the RDF model. Elements of this system can be dynamically created and used to store metadata in a generic way.

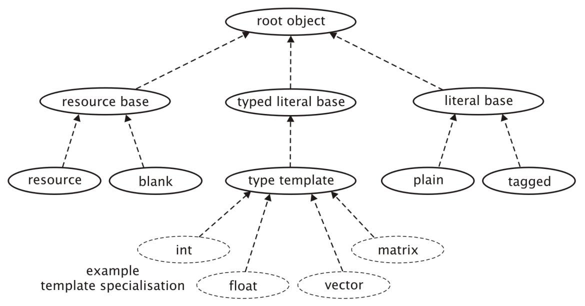


Figure 2: Type System

The system addresses the various data representation needs in our software. Generic resource types are assigned a corresponding metaobject, linking the resource to its ontological class definition. These objects contain a map in order to model the properties associated with the class. Literals are modelled after the types permitted in RDF and XML Schema. We can represent both plain string literals, and string literals coupled with a language tag this way. Numerical types however require a more complex representation. Our solution is based on C++ template specialisation. We map permitted XSD types[15] to corresponding simple or complex C++ data types wrapped into generic container templates. For instance, vectors and matrix classes can be mapped as plain strings or suitable XML literals in the RDF representation.

The classes of this subsystem can be configured in three different ways. They can act as references simply associating semantics

[9] `http://www.audacityteam.org/`

[10] `http://www.oracle.com/technology/products/berkeley-db/index.html`

[11] `www.wxwidgets.org`

[12] SPARQL Protocol and RDF Query Language, an SQL-like language for accessing an RDF knowledge base.

[13] In the context of relational databases this is called object-relational mapping.

[14] http://www.python.org/

[15] These are used in identifying data types in the RDF model.

with data stored elsewhere. This is similar to the implementation of logic references in the Castor logic programming library.[23] The objects can be added to existing data structures, wrapping existing functionality. Finally, they can be used independently, within a separate hierarchy, for fulfilling more complex metadata management needs. For example, this shall be used for storing the wide range of audio features associated with a track. In all modes of operation, a set of overloaded constructors are used for creating appropriately configured objects, depending on their use.

4.4. Architecture

The architecture of the library consists of several classes with a complex interaction model. A simplified diagram showing the main building blocks can be seen in figure 3.

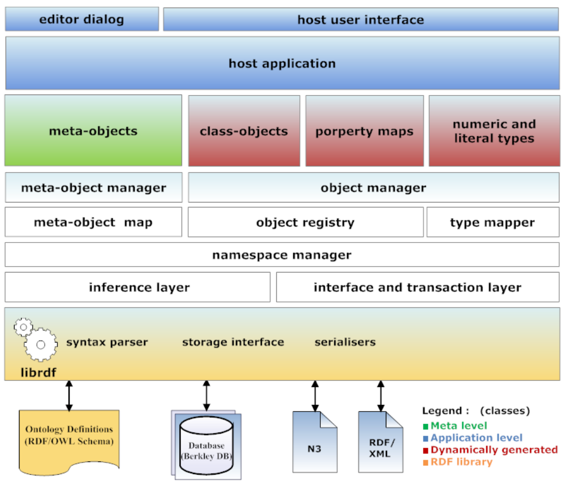


Figure 3: System Architecture

The *inference layer* is used to extract the class and property hierarchy of ontology definitions to build metaobjects. Logic programming functionalities can be added at this level in the future.

The *interface and transaction layer* is responsible for persisting the data stored in the object system, wrapping low-level graph manipulation calls and coordinating the addition of triples that must be stored atomically.

The *mapping layer* consists of a name space manager, a meta-object map, an object registry and a type mapper class. It is mainly responsible for linking meta-objects and dynamic-objects used in the system. The *TypeMapper* class maps XSD data type URI's used for identifying RDF typed literals onto function objects. These function objects (or functors) are used for creating dynamic objects, appropriate instances of dynamic class templates as described in the previous section.

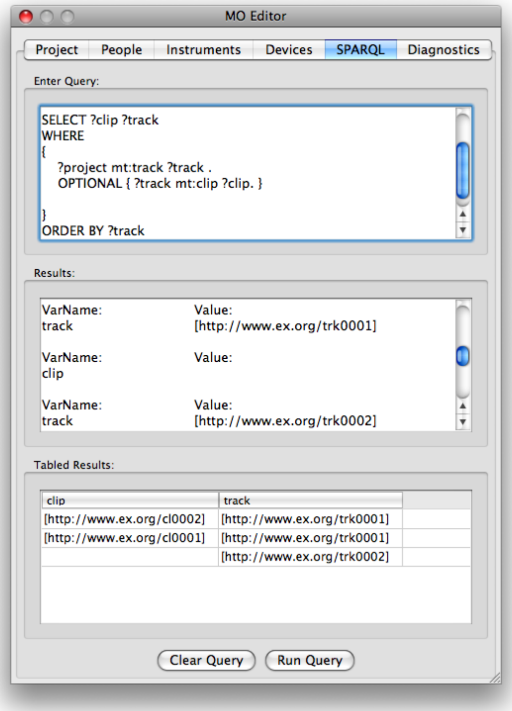
The *object management layer* consists of the meta object and dynamic object managers. Their primary function is storing and maintaining the objects created during the interaction with the model. Some additional functionality, common across dynamic objects, is implemented here using external polymorphism.

Meta-objects contain information corresponding to RDF and OWL classes and properties. For example, a *classInfo* object holds the inheritance hierarchy of a particular class. These instances are dynamically generated when parsing ontology schema.

Dynamic objects represent terms and relationships as objects in the application, string literals, simple XSD types as well as more complex numerical types. The previously described type system is implemented here.

4.5. The System in Use

Although the library is under development, it is already capable of storing a wide range of manually entered data, as well as capturing information while a user interacts with the audio editor interface. In order to test the capabilities of the system, we created a SPARQL query interface (figure 4) which allows the user to build and execute complex queries, and retrieve data stored in the RDF model.



Enter Query:

```
SELECT ?clip ?track
WHERE
{
   ?project mt:track ?track .
   OPTIONAL { ?track mt:clip ?clip. }

}
ORDER BY ?track
```

Results:

```
VarName:            Value:
track               [http://www.ex.org/trk0001]

VarName:            Value:
clip

VarName:            Value:
track               [http://www.ex.org/trk0002]
```

Tabled Results:

clip	track
[http://www.ex.org/cl0002]	[http://www.ex.org/trk0001]
[http://www.ex.org/cl0001]	[http://www.ex.org/trk0001]
	[http://www.ex.org/trk0002]

Figure 4: SPARQL query interface

SPARQL is a W3C recommendation for accessing RDF data stores. It's standardisation and increasing support promotes the adoption of RDF as a main metadata language. It allows retrieving information from RDF knowledge bases in a similar manner to querying relational databases using SQL. A query consists of a set of triple patterns that are matched against the data in the RDF store. Query results are composed of variable bindings from matching statements, typically based on a SELECT clause specified by the user.

The above example shows the use of the Multitrack Ontology for representing two audio tracks and two audio clips created in the editor. When queried appropriately, the model returns a list of clips and tracks they are associated with. This interface enables novel ways of accessing audio related metadata, for example, we will be able to answer queries such as: list all audio tracks where dynamic range compression was applied with the ratio of 1:5.

5. CONCLUSIONS AND FUTURE WORK

We presented a software library for collecting audio related metadata in an extensible, ontology-driven framework. Our approach is using techniques borrowed from the Semantic Web community, and demonstrates the application of these techniques in an audio editor application. By overcoming the limitations imposed by disharmonious metadata practices and standards, our library facilitates advanced and creative use of metadata in music production environments. Typical use case examples include advanced queries, enabling navigation by semantic associations or highlighting audio sections processed in a particular way. The system can also be used for publishing data about the music production process. This can be valuable in forthcoming music related Semantic Web applications. Future work includes interconnecting our library with a number of audio processing host environments, such as the Ladspa/LV2[25] and Vamp[24] plugin host libraries, as well as providing further ontological extensions enabling these connections.

6. ACKNOWLEDGEMENTS

The authors acknowledge the support of the School of Electronic Engineering and Computer Science, Queen Mary, University of London. This work has been carried out at the Centre for Digital Music and supported by the EPSRC-funded ICT project OMRAS-2 (EP/E017614/1). We wish to express gratitude towards the developers of *Audacity* for the open-source application used as basis in our work.

7. REFERENCES

[1] E. Gomez, G. Peterschmitt, X. Amatriain, P. Herrera "Content-Based Melodic Transformations of Audio Material for a Music Processing Application" *in Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03), London, UK, September 8-11, 2003.*

[2] V. Verfaille, U. Zölzer, D. Arfib "Adaptive Digital Audio Effects (A-DAFx): A New Class of Sound Transformations" *IEEE transactions on audio, speech, and language processing, vol. 14, no 5, pp. 1817-1831, 2006.*

[3] E. Pampalk, P. Hlavac, P. Herrera "Hierarchical Organization and Visualization of Drum Sample Libraries" *in Proc. of the 7th Int. Conference on Digital Audio Effects (DAFx-04), Naples, Italy, October 5-8, 2004*

[4] G. Fazekas, M. Sandler, "Intelligent Editing of Studio Recordings with the help of Automatic Music Structure Extraction", *presented at the AES 122th convention, Vienna, Austria, May, 2007.*

[5] M. Wright, A. Chaudhary, A. Freed, S. Khoury, D. Wessel, "Audio applications of the sound description interchange format standard," *in Proceedings of the International Computer Music Conference, Ann Arbor, Michigan, 1999.*

[6] J. Hunter, "Adding Multimedia to the Semantic Web - Building an MPEG-7 Ontology", *Proceedings of SWWS'01, (The first Semantic Web Working Symposium, Stanford University, California, USA, July 30 - August 1, 2001*

[7] R. García and O. Celma, "Semantic integration and retrieval of multimedia metadata.", [Online]. Available: `http://rhizomik.net/content/roberto/papers/rgocsemannot2005.pdf`

[8] D. McEnnis, C. McKay, I. Fujinaga, P. Depalle, "jAudio: A Feature Extraction Library", *in Proc. of the International Conference on Music Information Retrieval, London, UK, 2005.*

[9] C. McKay, R. Fiebrink, D. McEnnis, B. Li, and I. Fujinaga., "ACE: A framework for optimizing music classification." *Proceedings of the International Conference on Music Information Retrieval. 2005., 42–9.*

[10] J. Bullock and H. Frisk, "libIntegra: A System for Software-Independent Multimedia Module Description and Storage," *in Proceedings of the International Computer Music Conference, Copenhagen, Denmark, 2007.*

[11] AAF specification and supporting documentation `http://www.aafassociation.org/`

[12] D. Beenham, P. Schmidt and G. Sylvester-Bradley, "XML Based Dictionaries for MXF/AAF Applications", *Sony Broadcast and Professional Research Laboratories, UK*

[13] O. Lassila and R. Swick, "Resource description framework (RDF) model and syntax specification", 1998. `http://citeseer.ist.psu.edu/article/lassila98resource.html`

[14] Y. Raimond., S. Abdallah, and M. Sandler. "The Music Ontology", *Proceeedings of the 8th International Conference on Music Information Retrieval*, Vienna, Austria, 2007.

[15] F. Giasson and Y. Raimond. "Music Ontology Specification", `http://musicontology.com/`

[16] Y. Raimond. and S. Abdallah. "The Timeline Ontology", `http://motools.sourceforge.net/timeline/timeline.html`

[17] Y. Raimond. and S. Abdallah. "The Event Ontology", `http://motools.sourceforge.net/event/event.html`

[18] Y. Raimond. "Audio Features Ontology Specification", `http://motools.sourceforge.net/doc/audio_features.html`

[19] G. Fazekas. "Multitrack Ontology", `http://purl.org/ontology/studio/multitrack`

[20] D. Beckett, "The Redland RDF libraries", Available online: `http://librdf.org/`

[21] G. Kiczales, J. Des Rivieres, D. G. Bobrow: "The Art of the Metaobject Protocol", MIT Press, 1991

[22] N. Levine, "Fundamentals of CLOS", *International Lisp Conference ILC-03, New York, 2003.*Available online: `http://www.ravenbrook.com/doc/2003/07/15/clos-fundamentals/`

[23] Roshan Naik, "Introduction to Logic Programming in C++", Available online: `http://www.mpprogramming.com/Cpp/`

[24] C. Cannam. "The Vamp Audio Analysis Plugin API: A Programmer's Guide", `http://vamp-plugins.org/guide.pdf`

[25] T. Wilms, S. Harris, D. Robillard, "LV2 Audio Plugin Standard", `http://lv2plug.in/spec/`