USING ACE XML 2.0 TO STORE AND SHARE FEATURE, INSTANCE AND CLASS DATA FOR MUSICAL CLASSIFICATION

Cory McKay
CIRMMT
McGill University
cory.mckay@
mail.mcgill.ca

John Ashley Burgoyne
CIRMMT
McGill University
ashley@
music.mcgill.ca

Jessica Thompson
Music Technology
McGill University
jessica.thompson@
mail.mcgill.ca

Ichiro Fujinaga
CIRMMT
McGill University
ich@
music.mcgill.ca

ABSTRACT

This paper introduces ACE XML 2.0, a set of file formats that are designed to meet the special representational needs of research in automatic music classification. Such standardized formats are needed to facilitate the sharing and long-term storage of valuable research data. ACE XML 2.0 is designed to represent a broad range of musical information clearly using a flexible, extensible, self-contained and formally structured framework. An emphasis is placed on representing extracted feature values, feature descriptions, instance annotations, class ontologies and related metadata.

1. INTRODUCTION

Many music information retrieval (MIR) research projects involve three core tasks: collecting and annotating ground-truth data; extracting feature values from instances; and training classification models using machine learning. These tasks require well-designed data representations, as insufficiently expressive representations can prevent learning algorithms from accessing valuable information.

Representational formats also have an important impact on the ability of MIR researchers to share valuable data with one another, particularly since ground-truth datasets can be expensive to acquire. Legal restrictions on distributing such datasets make the ability to share extracted feature values and ground-truth annotations particularly valuable. The absence of expressive, flexible, well-defined and well-supported standardized representational formats tends to result in individual research laboratories generating their own in-house data, with consequent wasteful repeated effort and lower quality data.

Standardized file formats are also needed to facilitate compatibility of MIR toolkits such as CLAM, jMIR, Marsyas, MIRtoolbox and Sonic Visualiser. Powerful packages such as these each have their own advantages, and a common representational format is needed if research performed using different toolkits is to be combined.

Standardized file formats are also needed to facilitate the evaluation and comparison of techniques from different research groups, something that has become apparent in the yearly Music Information Retrieval Evaluation Exchange (MIREX) competition [1]. The lack of an accepted standardized representational format necessitates the development of custom formats for each sub-task, which results in compromises with respect to expressivity and longevity. For example, the MIREX audio genre classification competition is carried out each year using ground-truth where each piece is labeled with only one genre label, despite general recognition that this is an unrealistic limitation that compromises results. The availability of standardized formats such as ACE XML that can be easily used to associate multiple classes with each instance could help to address such problems.

ACE XML 2.0 is proposed as a standard for representing information associated with the application of machine learning to music, including feature values, instance labels, class ontologies and associated metadata. ACE XML 2.0 has been developed as part of the Networked Environment for Musical Analysis (NEMA) [2] project, a multinational and multidisciplinary effort to create a general music information processing infrastructure.

2. ALTERNATIVE REPRESENTATIONS

There are a number of existing approaches that can be used to represent information related to automatic music classification. One is to simply store such information as raw binary data, such as Matlab [3] MAT files. Although this can be an easy and efficient way of storing data, it has problems with respect to portability, readability and longevity. Customized software is needed to parse or write each binary file type, and such software is often proprietary and can only be expected to have a limited life span.

Text files are an alternative to binary files. Although they are usually less space efficient, they address the weaknesses of binary files with respect to longevity, portability and readability. They can also be structured in a variety of standardized ways, ranging from simple delimited formats like CSV to markup languages like XML [4].

Weka ARFF [5] is a text-based format designed for general machine learning. Although ARFF files are cur-

rently the closest thing to a standard in the MIR community, they do have some significant limitations, such as inabilities to associate windows with instances, to group the values of a feature array, to store important metadata and to associate multiple classes with a single instance.

Another approach is to store feature values in audio files themselves, such as SDIF [6]. This technique can have a limited expressivity with respect to pertinent metadata, however, and is not appropriate for dealing with mixtures of cultural, symbolic and audio data.

Music Ontology [7] is one of the few representational frameworks designed specifically with MIR in mind, and it has many admirable strengths. It can represent musical ontologies of essentially any kind using RDF [8].

Music Ontology has a much broader scope than ACE XML, but is arguably less suited specifically to machine learning and automatic music classification, despite its advantages over ACE XML in other MIR domains. The advantages of ACE XML relative to semantic web solutions in general include greater conciseness; a reduced need for markup not directly relevant to the core problem domain; a lower barrier to entry for non-ontologists, particularly with respect to simplicity and convenience; a cleaner and more explicit structuring that is advantageous from a machine learning perspective; human readability, which is useful for application debugging and development; a self-contained nature that avoids the network dependence of RDF that can cause problems with respect to data integrity, robustness and accessibility, particularly considering the typically large size of feature data; and the simplicity of relying on only a single technology that is well-known in the MIR community (i.e., XML).

3. AN OVERVIEW OF ACE XML

3.1 General Overview

The primary design priorities behind ACE XML 2.0 are the maximization of expressivity, flexibility and extensibility while at the same time maintaining as much simplicity, accessibility and structure as possible.

There are four core ACE XML file types: *Feature Value, Feature Description, Instance Label* and *Class Ontology*. These file types hold, respectively, feature values extracted from instances; abstract information about features and their extraction parameters; class labels associated with particular instances and their subsections, as well as general metadata about instances; and ontological relationships between abstract classes. These XML file types may each be used independently, or they may be packaged with one another if desired (see Section 3.5).

The ACE XML file types are explained individually in Section 4, although space constraints prohibit more detailed descriptions. The XML DTDs shown in Figures 1 to 4 do specify their functionality in greater detail, however. Sample code excerpts for each of the four core ACE XML file types are also provided in Figures 5 to 8. It is important to note that these excerpts only demonstrate a reduced subset of ACE XML's expressivity, however, as many of the ACE XML elements and attributes are optional so that they can be included only when appropriate. This makes it possible to use simple and concise files by default, while maintaining the potential for much greater expressivity when needed.

ACE XML consists of multiple file types rather than just one because of the advantages, with respect to data portability and reusability, of explicitly separating fundamentally different types of information. One might, for example, extract features once from a large number of recordings and then reuse the resulting Feature Value file for multiple purposes, such as classification by performer, composer, genre and mood.

ACE XML is implemented in XML partly because it is a standardized format for which parsers are widely available. XML is also very flexible while maintaining the ability to structure data formally and clearly. XML is also relatively easily readable by both humans and machines.

ACE XML 2.0 is a significantly updated and expanded version of the earlier ACE XML 1.1, which was originally designed specifically for use with ACE [9]. It became apparent that certain important types of information could not be expressed with ACE XML 1.1, so ACE XML 2.0 was developed in order to address these needs and to make ACE XML useful to the MIR community outside the specific scope of ACE.

3.2 jMIR Support

jMIR [10] is a powerful suite of software applications developed for use as MIR research tools. Each of the jMIR applications reads and writes ACE XML, something that provides ACE XML early adopters with a powerful set of tools that are ready for immediate use:

- *jAudio:* An audio feature extractor.
- *jSymbolic:* A MIDI feature extractor.
- *jWebMiner:* A feature extractor that extracts cultural and demographic information from the web.
- *ACE:* A meta-learning system for machine learning.
- *jMusicMetaManager:* Software for managing and cataloguing large musical datasets.
- *Codaich, Bodhidharma MIDI, SAC:* research datasets.

3.3 Incorporating ACE XML into Other Software

A key factor in the effectiveness of any effort to encourage researchers to adopt new file formats is the ease with which they can incorporate the formats into their own software. Open-source code libraries are therefore currently in the process of being implemented to support ACE XML 2.0. These libraries are implemented in Java in order to increase portability, and do not rely on any additional technologies that might require special installation. They will provide functionality for parsing, writing and merging ACE XML files; for submitting search que-

ries in JDOQL or SQL; and for performing various utility functions such as translating ACE XML data to and from Weka data. They will also include standard data structures that external code can access via a simple and well-documented API or a GUI ACE XML editor.

3.4 Linking ACE XML 2.0 to External Resources

It can be advantageous to associate instances, features or classes with various types of external information. Although ACE XML 2.0 can represent a broad range of metadata internally, the strong structuring that makes ACE XML advantageous for machine learning ultimately imposes limitations relative to the much more freely structured RDF, for example.

ACE XML addresses this issue by permitting the use of RDF-like triples via the optional *uri* XML element and

```
<!ELEMENT ace_xml_feature_value_file_2_0 (comments?,
            related_resources?, instance+)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT related_resources (feature_value_file*,
            feature_description_file*, instance_label_file*,
            class_ontology_file*, project_file*, uri*)>
<!ELEMENT feature_value_file (#PCDATA)>
<!ELEMENT feature_description_file (#PCDATA)>
<!ELEMENT instance_label_file (#PCDATA)>
<!ELEMENT class_ontology_file (#PCDATA)>
<!ELEMENT project_file (#PCDATA)>
<!ELEMENT uri (#PCDATA)>
<!ATTLIST uri predicate CDATA #IMPLIED>
<!ELEMENT instance (instance_id, uri*, extractor*, coord_units?,
                    s*, precise_coord*, f*)>
<!ELEMENT instance_id (#PCDATA)>
<!ELEMENT extractor (#PCDATA)>
<!ATTLIST extractor fname CDATA #REQUIRED>
<!ELEMENT coord_units (#PCDATA)>
<!ELEMENT s (uri*, f+)>
<!ATTLIST s b CDATA #REQUIRED e CDATA #REQUIRED>
<!ELEMENT precise_coord (uri*, f+)>
<!ATTLIST precise_coord coord CDATA #REQUIRED>
<!ELEMENT f (fid, uri*, (v+ | vd+ | vs+ | vj))>
<!ATTLIST f type (int | double | float | complex | string)
            #IMPLIED>
<!ELEMENT fid (#PCDATA)>
<!ELEMENT v (#PCDATA)>
<!ELEMENT vd (#PCDATA)>
<!ATTLIST vd d0 CDATA #REQUIRED d1 CDATA #IMPLIED
            d2 CDATA #IMPLIED d3 CDATA #IMPLIED
            d4 CDATA #IMPLIED d5 CDATA #IMPLIED
            d6 CDATA #IMPLIED d7 CDATA #IMPLIED
            d8 CDATA #IMPLIED d9 CDATA #IMPLIED>
<!ELEMENT vs (d+, v)>
<!ELEMENT d (#PCDATA)>
<!ELEMENT vj (#PCDATA)>
```

Figure 1: XML DTD for the ACE XML 2.0 Feature Value file format.

```
<!ELEMENT ace_xml_feature_description_file_2_0 (comments?,
            related_resources?, global_parameter*, feature+)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT related_resources (feature_value_file*,
            feature_description_file*, instance_label_file*,
            class_ontology_file*, project_file*, uri*)>
<!ELEMENT feature_value_file (#PCDATA)>
<!ELEMENT feature_description_file (#PCDATA)>
<!ELEMENT instance_label_file (#PCDATA)>
<!ELEMENT class_ontology_file (#PCDATA)>
<!ELEMENT project_file (#PCDATA)>
<!ELEMENT uri (#PCDATA)>
<!ATTLIST uri predicate CDATA #IMPLIED>
<!ELEMENT feature (fid, description?, related_feature*, uri*,
            scope, dimensionality?, data_type?, parameter*)>
<!ELEMENT fid (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT related_feature (fid, relation_id?, uri*,
                           explanation?)>
<!ELEMENT relation_id (#PCDATA)>
<!ELEMENT explanation (#PCDATA)>
<!ELEMENT scope (#PCDATA)>
<!ATTLIST scope overall (true|false) #REQUIRED
               sub_section (true|false) #REQUIRED
               precise_coord (true|false) #REQUIRED>
<!ELEMENT dimensionality (uri*, size*)>
<!ATTLIST dimensionality orthogonal_dimensions CDATA #REQUIRED>
<!ELEMENT size (#PCDATA)>
<!ELEMENT data_type (#PCDATA)>
<!ATTLIST data_type type (int | double | float | complex |
                          string) #REQUIRED>
<!ELEMENT global_parameter (parameter_id, uri*, description?,
                            value?)>
<!ELEMENT parameter (parameter_id, uri*, description?, value?)>
<!ELEMENT parameter_id (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

Figure 2: XML DTD for the ACE XML 2.0 Feature Description file format.

```
<!ELEMENT ace_xml_instance_label_file_2_0 (comments?,
            related_resources?, instance+)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT related_resources (feature_value_file*,
            feature_description_file*, instance_label_file*,
            class_ontology_file*, project_file*, uri*)>
<!ELEMENT feature_value_file (#PCDATA)>
<!ELEMENT feature_description_file (#PCDATA)>
<!ELEMENT instance_label_file (#PCDATA)>
<!ELEMENT class_ontology_file (#PCDATA)>
<!ELEMENT project_file (#PCDATA)>
<!ELEMENT uri (#PCDATA)>
<!ATTLIST uri predicate CDATA #IMPLIED>
<!ELEMENT instance (instance_id, misc_info*, related_instance*,
                    uri*, coord_units?, section*,
                    precise_coord*, class*)>
<!ATTLIST instance role (training | testing | predicted)
                        #IMPLIED>
<!ELEMENT instance_id (#PCDATA)>
<!ELEMENT related_instance (instance_id, relation_id?, uri*,
                            explanation?)>
<!ELEMENT relation_id (#PCDATA)>
<!ELEMENT explanation (#PCDATA)>
<!ELEMENT misc_info (info_id, uri*, info)>
<!ELEMENT info_id (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT coord_units (#PCDATA)>
<!ELEMENT section (uri*, class+)>
<!ATTLIST section begin CDATA #REQUIRED
                  end CDATA #REQUIRED>
<!ELEMENT precise_coord (uri*, class+)>
<!ATTLIST precise_coord coord CDATA #REQUIRED>
<!ELEMENT class (class_id, uri*)>
<!ATTLIST class weight CDATA "1">
<!ATTLIST class source_comment CDATA #IMPLIED>
<!ELEMENT class_id (#PCDATA)>
```

Figure 3: XML DTD for the ACE XML 2.0 Instance Label file format.

```
<!ELEMENT ace_xml_class_ontology_file_2_0 (comments?,
            related_resources?, class+)>
<!ATTLIST ace_xml_class_ontology_file_2_0 weights_relative
            (true|false) #REQUIRED>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT related_resources (feature_value_file*,
            feature_description_file*, instance_label_file*,
            class_ontology_file*, project_file*, uri*)>
<!ELEMENT feature_value_file (#PCDATA)>
<!ELEMENT feature_description_file (#PCDATA)>
<!ELEMENT instance_label_file (#PCDATA)>
<!ELEMENT class_ontology_file (#PCDATA)>
<!ELEMENT project_file (#PCDATA)>
<!ELEMENT uri (#PCDATA)>
<!ATTLIST uri predicate CDATA #IMPLIED>
<!ELEMENT class (class_id, misc_info*, uri*, related_class*,
                 sub_class*)>
<!ELEMENT class_id (#PCDATA)>
<!ELEMENT misc_info (info_id, uri*, info)>
<!ELEMENT info_id (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT related_class (class_id, relation_id?, uri*,
                         explanation?)>
<!ATTLIST related_class weight CDATA "1">
<!ELEMENT relation_id (#PCDATA)>
<!ELEMENT explanation (#PCDATA)>
<!ELEMENT sub_class (class_id, relation_id?, uri*,
                     explanation?)>
<!ATTLIST sub_class weight CDATA "1">
```

Figure 4: XML DTD for the ACE XML 2.0 Class Ontology file format.

its associated *predicate* attribute. This enables links to be specified to external resources of essentially any kind without compromising ACE XML's structured and self-contained design philosophy. In particular, it makes it easy to link ACE XML files to large RDF ontologies.

3.5 ACE XML 2.0 Project and ZIP Files

Although it is beneficial to be able to specify the information encapsulated in each of the four ACE XML formats in separate files, in practice users will want to use multiple ACE XML files together. The *ACE XML 2.0 Project* file format facilitates this by providing functionality for linking ACE XML (and other) resources together.

It is also possible to package multiple associated ACE XML files together into a single *ACE XML 2.0 ZIP* file for simplified storage and distribution. This is also advantageous because of the reduced file sizes resulting from data compression. Of course, the original ACE XML files may be extracted from this ACE XML ZIP file whenever desired. The supporting ACE XML software includes functionality for automatically generating, accessing and otherwise processing ACE XML Project and ZIP files.

4. THE CORE ACE XML 2.0 FILE FORMATS

This section provides descriptions of each of the four core ACE XML file formats: *Feature Value, Feature Description, Instance Label* and *Class Ontology*.

4.1 Feature Value Files

Feature Value files are used to express feature values that have been extracted from instances that are to be classified or used as training data. There is no assumed association with any specific kind of data, and so features may be extracted from audio recordings, symbolic recordings, textual or numeric cultural data, images of album art, etc.

Features may be extracted from instances as a whole (e.g., an entire score), from subsections of instances (e.g., audio analysis windows) or from a mixture of the two. Subsections may or may not overlap, may or may not be of equal size and may or may not cover an instance comprehensively. Each instance or subsection may also contain an arbitrary and potentially differing number of features, which makes it possible to omit features when appropriate or if they are unavailable.

Each instance in a Feature Vector file has an *instance_id* tag that may be used to associate it with class labels and metadata stored in an Instance Label file. Similarly, each feature has an *fid* tag that may be used to associate it with feature metadata stored in a Feature Description file. Other information that can be represented in a Feature Value file includes the data type (integer, double, string, etc.) of the feature, the feature extractor used to extract the feature values and links to external resources.

ACE XML 2.0 allows feature values to be expressed using any one of four methodologies, including one that is based on JavaScript Object Notation (JSON) [11]. Each such representation has its own advantages with respect to the maximum dimensionality of feature arrays, the ability to represent sparse feature arrays, human readability and space efficiency. An example of the most flexible (but not most space efficient) of these options is shown in Figure 5. This option allows feature arrays of any dimensionality and size to be represented, including sparse arrays and arrays that vary in size.

4.2 Feature Description Files

Feature Description files are used to express abstract information about features. These files do not specify actual feature values, as this information is instead specified in Feature Value files.

The information that may be represented in Feature Description files includes: details of pre-processing required before feature extraction (e.g., downsampling); feature extraction parameters; notations as to whether features are associated with instances as a whole or only with instance subsections; the dimensionality and size of each feature (i.e., a single-value feature, a feature vector or a feature array); the data type of each feature; qualitative feature descriptions; relationships between different features; and links to external resources.

There are other possible applications for Feature Description files beyond simply using them to represent information associated with Feature Value files. Examples include catalogues of features that can be extracted by particular feature extraction applications and lists of features and associated parameters that have been found to be useful for particular music classification applications.

4.3 Instance Label Files

Instance Label files are used to specify class labels and miscellaneous metadata about instances. These files are typically used to express ground-truth annotations or predicted labels, but there are certainly other uses as well.

Class labels may be assigned to instances as a whole, to subsections of instances, or to both. Subsections may be overlapping and may be of varying sizes. Weighted multi-class membership is also permitted. Additional information that may be associated with instances and their subsections includes the source of the class labels (e.g., a listener survey); whether the class label(s) for an instance are predicted labels or ground-truth; relationships of any kind between instances (e.g., one is a cover song of another); miscellaneous field-labeled qualitative metadata (e.g., the performer or composer of a piece); and links to external resources. Instance Label files may be linked with Feature Value files using matching *instance_id* tags and with Class Ontology files using matching *class* tags.

```
<instance>
   <instance_id>An Artificial Instance</instance_id>
   <f>
      <fid>A Single Value Feature</fid>
      <v>1</v>
   </f>
   <f>
      <fid>Feature Array of Value</fid>
      <vs><d>0</d><d>0</d><v>1</v></vs>
      <vs><d>0</d><d>1</d><v>2</v></vs>
      <vs><d>0</d><d>2</d><v>3</v></vs>
      <vs><d>1</d><d>0</d><v>11</v></vs>
      <vs><d>1</d><d>1</d><v>22</v></vs>
      <vs><d>1</d><d>2</d><v>33</v></vs>
   </f>
</instance>
```

Figure 5: An excerpt from a sample ACE XML 2.0 Feature Value file indicating two artificial features extracted from a single instance. The first feature has a value of *1,* and the second is the 2 by 3 feature array: *[[1,2,3],[11,22,33]]*. In practice, a Feature Value file could contain multiple such instances as well as features extracted from subsections of instances.

```
<feature>
   <fid>Beat Histogram</fid>
   <description>Tempo histogram calculated using
               Autocorrelation.</description>
   <related_feature>
      <fid>Tempo Peak</fid>
      <relation_id>derivative feature</relation_id>
   </related_feature>
   <scope overall="true" sub_section="false"
         precise_coord="false"></scope>
   <dimensionality orthogonal_dimensions="1">
      <size>161</size>
   </dimensionality>
   <data_type type="double"></data_type>
   <parameter>
      <parameter_id>normalized</parameter_id>
      <value>true</value>
   </parameter>
</feature>
```

Figure 6: An excerpt from a sample ACE XML 2.0 Feature Description file indicating information about a single feature called *Beat Histogram*. It is noted that *Beat Histogram* is related to another feature called *Tempo Peak* that can be calculated from the *Beat Histogram* feature, that *Beat Histogram* is configured to be extracted only for files as a whole, that it consists of a single vector of size 161, that feature values are stored as doubles and that the values are normalized. In practice, a Feature Description file would contain multiple such *feature* clauses, each for a different feature.

```
<instance role="predicted">
   <instance_id>C:\Symbolic\piece_42.midi</instance_id>
   <coord_units>ms</coord_units>

   <section begin="0" end="85673">
      <class>
         <class_id>Sonata Exposition</class_id>
      </class>
   </section>
   <section begin="85674" end="278894">
      <class>
         <class_id>Sonata Development</class_id>
      </class>
   </section>
   <section begin="278895" end="525419">
      <class>
         <class_id>Sonata Recapitulation</class_id>
      </class>
   </section>

   <class weight="3">
      <class_id>Haydn</class_id>
   </class>
   <class weight="1">
      <class_id>Mozart</class_id>
   </class>
</instance>
```

Figure 7: An excerpt from a sample ACE XML 2.0 Instance Label file specifying class labels for a MIDI file. As indicated by the *role* attribute, the labels are predicted classifier outputs. The subsections are classified by form and the overall instance is classified by composer. The classification system has expressed that this piece is three times as likely to be by Haydn than by Mozart. In practice, an Instance Label file would contain multiple such *instance* clauses.

```
<class>
   <class_id>Robert Johnson</class_id>
</class>

<class>
   <class_id>Muddy Waters</class_id>
   <related_class weight="10">
      <class_id>Robert Johnson</class_id>
      <relation_id>Influenced By</relation_id>
   </related_class>
   <related_class weight="1">
      <class_id>Eric Clapton</class_id>
      <relation_id>Influenced By</relation_id>
   </related_class>
</class>

<class>
   <class_id>Eric Clapton</class_id>
   <related_class weight="30">
      <class_id>Robert Johnson</class_id>
      <relation_id>Influenced By</relation_id>
   </related_class>
   <related_class weight="10">
      <class_id>Muddy Waters</class_id>
      <relation_id>Influenced By</relation_id>
   </related_class>
</class>
```

Figure 8: An excerpt from an artificial ACE XML 2.0 Class Ontology file indicating class labels consisting of names of Blues musicians. A type of relationship between classes is also specified, namely musicians influenced by other musicians. In this example, there is no relationship from Robert Johnson to the other musicians because he was not influenced by them. Both of the other musicians are influenced by Johnson, however. Clapton is more influenced by Johnson than by Muddy Waters, and Muddy Waters is strongly influenced by Johnson but only slightly influenced by Clapton, as indicated by the *weight* values.

4.4 Class Ontology Files

Class Ontology files are used to specify candidate class labels for a particular classification domain as well as weighted ontological relationships between classes. These files do not, however, specify the labels of any actual instances, as this is the domain of Instance Label files.

The ability to specify ontological class structuring has several important benefits. From a musicological perspective, it provides a simple, machine-readable way of specifying a variety of musical relationships. From a machine learning perspective, it has the dual advantages of enabling the use of powerful hierarchical classification methodologies that exploit this structuring, as well as learning schemes that utilize weighted penalization to punish "better" misclassifications less severely during training.

The information that may be expressed in Class Ontology files includes weighted taxonomical links to other classes; weighted general ontological links to other classes; structured or unstructured descriptions of such links; miscellaneous qualitative structured metadata (e.g., the birthplace of a composer if music is being classified by composer); and links to external resources.

5. CONCLUSIONS

This paper has emphasized the need for more effective representational formats for use in MIR and automatic music classification research. ACE XML 2.0 was presented as a solution to these needs. It is hoped that ACE XML will help to facilitate communication and data sharing between research groups involved in the computational study of music and correspondingly increase the efficiency and quality of research.

Much more detailed information, including sample ACE XML 2.0 files and an in-depth ACE XML 2.0 manual, are available at *jmir.sourceforge.net*.

6. FUTURE RESEARCH

Future work will focus on continuing to produce developer tools to help facilitate the integration of ACE XML functionality into other software. Once work is completed on implementing the ACE XML 2.0 support software (the ACE XML 1.1 software is already complete) in Java it will then be ported to other languages, such as Python, C++ and Matlab. There are also plans to write ACE XML 2.0 plug-ins for the popular MIR software toolkits and to implement tools for translating ACE XML to other representational formats. The upgrading of all jMIR components from ACE XML compatibility 1.1 to ACE 2.0 compatibility is a particular priority.

Another priority is the continuing extension and overall improvement of the ACE XML standard. This will include the expression of more strictly constrained rules specified using XSD or Relax NG schemas.

The publication of a common repository for data stored in ACE XML files is another key goal. This will enable such data to be posted and shared amongst researchers. This will also be a forum where best practices and extensions to the ACE XML standard can be discussed and agreed upon by the MIR community. Indeed, ideas from the MIR community for future improvements to ACE XML in general are very welcome, and upgrades to the file formats will continue, with the provision that backwards compatibility is maintained.

7. ACKNOWLEDGEMENTS

The authors would like to thank the *Andrew W. Mellon Foundation* and the *Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT)* for their generous financial support, as well as the members of the *Networked Environment for Musical Analysis (NEMA)* group for their valuable critiques and suggestions.

8. REFERENCES

[1] MIREX 2009. Retrieved 11 May 2009, from http://www.music-ir.org/mirex/2009.

[2] NEMA. Retrieved 11 May 2009, from http://nema.lis.uiuc.edu.

[3] Mathworks. Retrieved 11May 2009, from http://www.mathworks.com.

[4] Ray, E. T. 2003. *Learning XML.* Sebastopol, CA: O'Reilly Media.

[5] Witten, I. H., and E. Frank. 2005. *Data mining: Practical machine learning tools and techniques.* New York: Morgan Kaufman.

[6] Burred J. J., C. E. Cella, G. Peeters, A. Röbel, and D. Schwarz. 2008. Using the SDIF Sound Description Interchange Format for audio features. *Proceedings of the International Conference on Music Information Retrieval.* 427–32.

[7] Raimond, Y., S. Adbdallah, M. Sandler, and F. Giasson. 2007. The Music Ontology. *Proceedings of the International Conference on Music Information Retrieval.* 417–22.

[8] Powers, S. 2003. *Practical RDF.* Sebastopol, CA: O'Reilly Media.

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

[10] McKay, C., and I. Fujinaga. 2009. jMIR: Tools for automatic music classification. Accepted for publication in the *Proceedings of the International Computer Music Conference.*

[11] JSON. Retrieved 11 May 2009, from http://json.org.