The CLAM Annotator: A Cross-platform Audio Descriptors Editing Tool

Xavier Amatriain
CREATE
University of California
Santa Barbara CA 93106 USA
xavier@create.ucsb.edu

Jordi Massaguer
Universitat Pompeu Fabra
Psg. Circumvalacio, 18
Barcelona, Spain
jmassaguer@iua.upf.es

David Garcia
Universitat Pompeu Fabra
Psg. Circumvalacio, 18
Barcelona, Spain
dgarcia@iua.upf.es

Ismael Mosquera
Universitat Pompeu Fabra
Psg. Circumvalacio, 18
Barcelona, Spain
imosquera@iua.upf.es

ABSTRACT

This paper presents the CLAM Annotator tool. This application has been developed in the context of the CLAM framework and can be used to manually edit any previously computed audio descriptors. The application offers a convenient GUI that allows to edit low-level frame descriptors, global descriptors of any kind and segmentation marks. It is designed in such a way that the interface adapts itself to a user-defined schema, offering possibilities to a large range of applications.

Keywords: Audio Descriptors, XML, Annotating Tool

1 INTRODUCTION

Descriptor extraction from an audio source is one of the most important practices related to the Music Information Retrieval field. Many different research teams are focusing on finding more and better algorithms to automatically extract relevant features from the original signal.

Nevertheless, none of these algorithms can guarantee a 100% reliability. In many cases automatically extracted descriptors must be fine-tuned by hand and appropriate tools are therefore needed. These manual tools become even more important for research teams when testing new algorithms.

The goal of the CLAM Annotator is to provide such a tool, offering a flexible, reliable, extensible and efficient alternative to other existing applications such as the WaveSurfer [1]. Also, the fact that the CLAM Annotator is provided in the context of the CLAM framework [2] allows to extend it in unlimited ways, by embedding automatic extraction algorithms in it, for instance.

Another important feature of the CLAM Annotator is that it uses standard XML language to represent data. Thus it works with readable and easily understandable data files that should be also easily connected to external applications or databases.

2 ON FILES AND FORMATS

The CLAM Annotator makes use of different XML files in order to relate with the outside world. All previously generated information is input to the program through XML files and the result of the editing process is also dumped into those files.

The *Project File* contains a pointer to a *Schema File* and another one to a *Song List File*. The Song List File contains a list of *Audio Files* and *Descriptors Pool Files*.

In the following sections we will detail the content of each of those files.

2.1 The Project File

The Project file is an XML file with the ".pro" extension. It simply contains the path to the Song List file and the path to the Schema file. See the following example:

```
<Project>
 <Songs>Songs.sl</Songs>
 <Schema>Schema.sc</Schema>
</Project>
```

Listing 1: Sample Annotator Project file

2.2 The Song List File

The Song List file is also an XML file with the ".sl" extension. It contains a list of Sound file + Descriptors file tuples.

A Sound file is simply the path to an existing sound file. This sound file can be in virtually any format, including PCM encoded files such as WAVs or AIFFs or compressed formats such as MP3 or OGG. On the other hand, the Descriptors file has a pointer to the descriptors related to that particular sound file. If omitted, the program will simply add the ".pool" extension to the sound file name.

See the following example:

```
<SongFiles>
 <FileNames>
  <Name>
     <SoundFile>07.mp3</SoundFile>
  </Name>
  <Name>
     <SoundFile>08.mp3</SoundFile>
     <DescriptorsFile>08.mp3.pool</
         DescriptorsFile>
  </Name>
  ...
 </FileNames>
</SongFiles>
```

Listing 2: Sample SongList file

2.3 The Schema File

The Schema file contains the list of all the different descriptors that will later will be loaded from a Descriptors file (see 2.4). In some cases it also gives their type and range of expected values. Although this file is a regular XML file with the ".sc" extension, in many senses it mimics the purpose and syntax of an XML Schema format [3].

The Schema file is actually divided into two different sections. The first one defines the Schema for high-level descriptors while the second one defines the schema for low-level descriptors (see example in listing 3).

Although the difference between low and high-level descriptors is a matter of controversy and much has been written about it (see [4]) in this application we have taken a pragmatical approach. A high-level descriptor is considered to be any descriptor that has a whole song scope and is unique within this scope. This kind of descriptor can be of any type. On the other hand a low-level descriptor has a Frame scope and can only take floating point values.

```
<Schema>
 <HLDSchema><HLDs>
  <HLD>
   <Name>Title</Name>
   <Type>String</Type>
  </HLD>
  <HLD>
   <Name>Danceability</Name>
   <Type>Float</Type>
   <fRange><Max>10</Max><Min>0</Min></
       fRange>
  </HLD>
  <HLD>
   <Name>Key</Name>
   <Type>Enum</Type>
   <Values>A A# B C...</Values>
  </HLD>
  <HLD>
   <Name>BPM</Name>
   <Type>Int</Type>
   <iRange><Max>240</Max><Min>0</Min></
       iRange>
  </HLD>
  ...
 </HLDs></HLDSchema>
 <LLDSchema>
  <LLDNames>Pitch SpectralCentroid
      SpectralSpread...</LLDNames>
  </LLDSchema>
</Schema>
```

Listing 3: Sample Annotator Schema file

We will now see how the schema is defined both for high-level and low-level descriptors.

2.3.1 High-level descriptors

As already mentioned a high-level descriptor has a unique value for a whole song or sound source. It can be of any of the following types: floating point number ("Float"); integer number ("Int"); string ("String"); or value set restricted strings ("Enum").

A high-level descriptor is therefore defined by giving its "Name" and its "Type". In case the type is a number, an optional range of valid values may be given ("iRange" in case of integer values and "fRange" in case of floating point values). See the HLD section in listing 3.

2.3.2 Low-level Descriptors

A low-level descriptor is in any case a vector of floating point values where each value refers a particular frame. In this case we only need to define the name of the descriptors. Therefore the low-level descriptors section of the schema is simply a list of low-level descriptors names (see again listing 3).

2.4 Descriptors Pool File

This is an XML file with the ".pool" extension that contains all the values, both for the high-level and low-levels descriptors. The content must observe the restrictions given in the related Schema or else it will not be *validated*.

Every song on the project has its own descriptors file. Descriptions may be generated by any third-party application by providing a proper schema, though it is much easier to generate it from within the CLAM framework. In this case, the Descriptors file is directly the XML representation of a CLAM Descriptors Data Pool. Any extraction algorithm using them may dump its results in such format without having to worry about formatting issues.

As in the Schema, a Descriptors file is divided into two sections: one for the high-level descriptors and another one for the low-level descriptors (see listing 4). Note that in the Descriptors file this difference is explicit by the existence of two different "Scopes", one with the name "Song" and size=1 (there is only one song for each song) and the other one with the name "Frame" and size=8917 (in this case there are 89671 frames in the song).

```
<DescriptorsPool>
 <ScopePool name="Song" size="1">
   ...
 </ScopePool>
 <ScopePool name="Frame" size="8961">
  ...
 </ScopePool>
</DescriptorsPool>
```

Listing 4: Sample Descriptors file

We will now explain how high and low-level descriptors are stored.

2.4.1 High-level Descriptors

In the Song scope we basically see a list of AttributePool elements. In any case each of those elements has an attribute with the name of the particular descriptor and its content is the content of the descriptor. Note that the type of the descriptor is implicitly resolved from the schema and must therefore not be given in the Descriptors file (see HLD description in listing 5).

Finally, segmentation information is also included in the high-level description. This descriptor must not be given in the schema as it is always supposed to be available. When including segmentation marks in the description you must give their size (i.e. how many segmentation marks are available) and the list of positions in number of samples.

```
<ScopePool name="Song" size="1">
 <AttributePool name="Title">
     Pension_Triana</AttributePool>
 <AttributePool name="Danceability">7.2</
     AttributePool>
```

```
 <AttributePool name="Key">C</
     AttributePool>
 <AttributePool name="BPM">100</
     AttributePool>
 ...
 <AttributePool name="Segments" size="43">
     202334 497049 ...</AttributePool>
</ScopePool>
```

Listing 5: Sample High-level Description

2.4.2 Low-Level descriptors

The low-level descriptors section of the Descriptors file is also a list of AttributePool elements where for each element we must define its name an a list of values. Note that in this case we must not give the size of each attribute because this is already defined by the size of the "Frame" scope. Therefore these vectors must all have as many elements as defined in the scope (8961 in the example given in listing 6).

```
<ScopePool name="Frame" size="8961">
 <AttributePool name="Pitch">63 68 60 ...<
     /AttributePool>
 <AttributePool name="SpectralDeviation">
     50 50 48 ...</AttributePool>
 ...
</ScopePool>
```

Listing 6: Sample Low-level Description

3 THE APPLICATION

The application has been developed within the CLAM framework, using qt [5] for the graphical user interface. Figure 1 is a capture of the whole interface running on an Debian GNU/Linux box with the KDE desktop environment, although its look is virtually the same in any of the major platforms and graphical environments (Windows, Mac OSX, GNOME...).

In the Annotator GUI, four different parts can be identified: on the upper-left the list of songs is shown, on the upper-right the waveform and segmentation information of the selected song is available, on the lower-left there is the information for the high-level descriptors, and finally on the lower-right there is the low-level descriptors information.

3.1 Loading a Project

Once the program is started, the first thing that the user must do is to load a project file. This project file will have a pointer to the Song List and the Schema files. Once loaded, the GUI is reconfigured and the list of songs and related descriptions is available.

The user can also start a project from scratch. In this case the Scheme and Song List files must be loaded by hand from the Project menu.

3.1.1 The Schema and the dynamic GUI

One of the most important features in the CLAM Annotator is its ability to dynamically adapt the GUI. The GUI shows the descriptors according to the Schema that is loaded with the Project.

In the case of low-level descriptors, the amount and label of each of the tabs corresponds to the schema. And in the case of high-level descriptors, the schema defines the label and also the kind of editing widget that is shown. The screenshot shown in figure 3 corresponds to the sample Schema introduced in section 2.3.

3.2 Viewing Song Properties

Once a song is selected from the Song List on the upper left, the audio file and the descriptors are loaded. After this loading process finishes the waveform including segmentation marks is available on the upper-right, the low-level descriptors are shown on the lower-right, and the high-level descriptors are on the lower-left. The user can listen to the sound file and start the edition process.

Low level descriptors view and segmentation view are synchronized in respect zoom, horizontal scroll and cursor position. That feature makes easy to take segmentation editing decissions taking into acount low level features values.

3.3 Editing Low-level Descriptors

Low-level descriptors are represented by equidistant connected points that you can drag to change its Y value. Each point represents the value for the descriptor in a given frame. Because point to point edition may be hard, some convenient edition modes such `trim` or `draw` are provided.

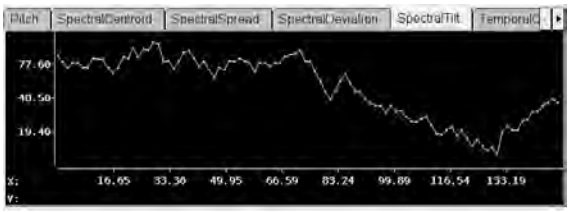

Figure 1: The low-level descriptors

3.4 Editing High-level Descriptors

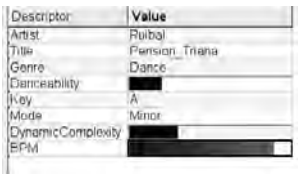

Figure 2: The high-level descriptors

The edition of a high-level description adapts on the "type" of the descriptor as defined in the project's schema. Figure 4 shows how integer and float descriptors may be edited by a slider that uses the range given in the schema, while *enumerated value* descriptors can be selected with a drop-down list widget with the allowed values. Regular strings use a simple text box widget were the user can enter free text.

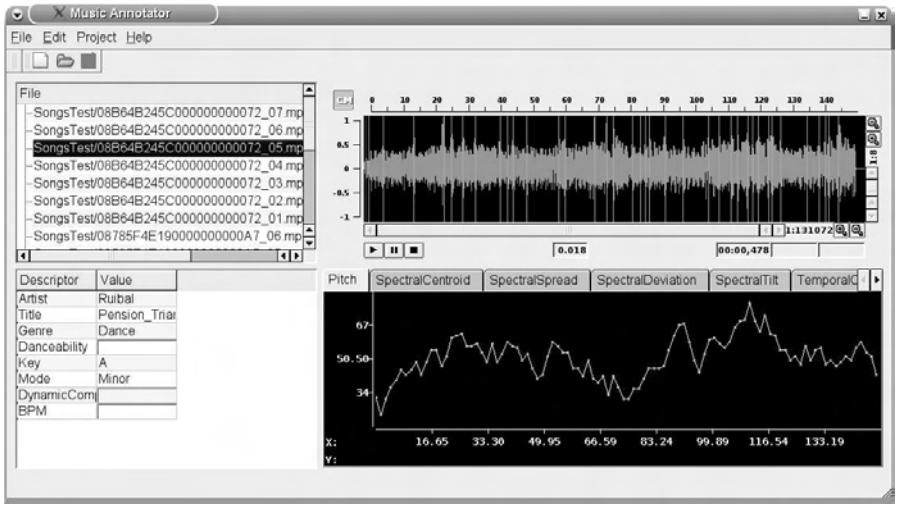

Figure 3: The CLAM Annotator GUI

Figure 4: Editing integer and enum high-level descriptors

3.5 Editing Segmentation Marks

Segmentation marks may be viewed/edited in the wave form view in the upper-right. They can be moved, deleted (with the Delete key pressed) and inserted (holding down the Insert key).

3.6 Auralizing annotations

Sometimes visual representation is not as meaningfull as an aural feedback could be. The CLAM Annotator provides auralization of descriptors. That means mapping annotation data to synthesizer controls so that we can listen the synthesized sound synchronized to the actual waveform. Currently we support segmentation marks auralization and floating point low level descriptors auralization.

Segmentation marks auralization consists on listening a percussive sound at every segment start point. Floating point low level descriptors may be auralized by modulating the pitch or the volume of a sound using descriptor values. Although low level descriptors auralization was intended for pitch related features, we found this kind of aural feedback being also meaninfull for other descriptors.

4 CONCLUSIONS AND FUTURE WORK

In the paper we have presented the first version of the CLAM Annotator, a tool that can be used in order to revise and fine-tune the result of automatic description extraction algorithms. This tool will be made available as Free Software in the next release of the CLAM framework.

Although we believe that in its current state the application is already useful and valuable, future plans include the possibility of dynamically loading automatic extraction algorithms. With this addition the CLAM Annotator will become a complete and flexible framework for audio description and annotation.

ACKNOWLEDGEMENTS

The authors wish to thank the rest of the CLAM development team for their contributions and all the members of the UPF Music Technology Group for their valuable feedback.

Part of this work has been done in the context of the SIMAC IST-507142 project.

References

[1] K.Sjölander and J. Beskow. Wavesurfer – an open source speech tool. In *Proceedings of the Eighth International Conference on Spoken Language Processing (ICSLP00)*, Beijing, China, 2000.

[2] X. Amatriain, P. Arumí and M.Ramírez. CLAM, Yet Another Library for Audio and Music Processing? In *Proceedings of the 2002 Conference on Object Oriented Programming, Systems and Application (OOPSLA2002)*, Seattle, USA, 2002. ACM.

[3] www-XMLSchema. World Wide Web Consortium (W3C)'s XML-Schema home page, *gramming with QT 3*. Pearson Education, 2004.

[4] X. Amatriain and P. Herrera. Transmitting Audio Content as Sound Objects. In *Proceedings of the AES 22nd Conference on Virtual, Synthetic, and Entertainment Audio*, Helsinki, 2001. Audio Engineering Society.

[5] J. Blanchette and M. Summerfield. *C++ GUI Programming with QT3*. Pearson Education, 2004.