An Extensible Representation for Playlists

Amar Chaudhary
Creative Advanced Technology Ctr.
1500 Green Hills Road
Scotts Valley, CA, USA
+1-831-440-2944

amar@atc.creative.com

ABSTRACT

The increasing availability of digital music has created a greater need for methods to organize large collections of music. The eXtensible PlayList (XPL) representation allows users to express playlists with varying degrees of specificity. XPL handles references to exact files or URLs as well as rules for selecting content based on metadata constraints. XPL also allows the transitions between tracks in a playlist to be specified. This paper describes the features of XPL, a system for rendering XPL specifications and use of an advanced XPL renderer in an existing application.

1. INTRODUCTION

The increasing availability of digital music on desktop PCs, the Internet and personal devices (e.g., portable MP3 devices) has created a greater need for methods to organize large collections of music. The traditional concept of a *playlist*, long employed by DJs and radio stations, has become an important element of digital music organization and playback. However, there are many ways to describe playlists. One could be very specific, listing the exact audio files to use (e.g., `Beatles_Yesterday.mp3`), artist and title names (e.g., *Yesterday* by the Beatles), or non-specific descriptions based on styles, tempi, etc (e.g., a down-tempo classic-rock ballad). Additionally, a sophisticated audio player can perform transitions between songs (e.g., cross fades, tempo matches, etc.). Thus, a complete playlist description may include information about transitions.

The "eXtensible PlayList" (XPL) representation allows users to express and exchange playlists with varying degrees of specificity. It extends the notion of a playlist-description format to include not only exact lists of audio files can be represented but also more general rule-based specifications based on artist, genre or tempo constraints and exact transitions can be specified between successive tracks.

2. The structure of XPL documents

This section describes the basic structure of XPL documents. It is not intended as a comprehensive specification, which can be obtained elsewhere [1].

2.1 Playlists

An XPL document begins and ends with `<xpl>` tags and consists of one or more playlist elements. A playlist element is enclosed in `<playlist>` tags, and consists of one or more track elements.

```
<?xml version="1.0" ?>
<xpl version="1.0">
<playlist attributes>
 track or playlist 1
 track or playlist 2
 ...
 track or playlist n
</playlist>
</xpl>
```

Playlist elements can also be nested, to form hierarchical structures. Similar to HTML documents, which can reference external sub-documents, images, etc., playlist elements can refer to other XPL files using a src attribute:

```
<playlist name="super_party_mix" >
  <playlist src="early_evening.xpl" />
  <playlist src="late_night.xpl" />
</playlist>
```

where the value of src can be a local file name or a URL.

2.2 Tracks

An XPL track element describes a piece of content to be played or a rule for selecting a piece of content. Track elements that specify a piece of content are called *static tracks* and formatted as follows:

```
<track [name=name]>
    <content src=filename />
</track>
```

where `src` is a filename or URL. The file references by `src` should be a "playable" piece of content, such as an MP3 file (of course, the definition of "playable" will differ among applications).

Track elements that specify rules are called *dynamic tracks* and can specified in different ways according to the type of rule being applied. The most common type of rule is a query that describes a set of constraints that an application can use to select a particular piece of content for the track. A *query track* is formatted as follows:

```
<track>
   <query> query specification</query>
</track>
```

The query specification uses an "SQL-like" language for describing constraints, for example the following track specification:

```
<track><query>
        genre="Acid Jazz" AND bpm > 120
</query></track>
```

results in the inclusion of a piece of content attributed to the genre "Acid Jazz" whose tempo is greater than 120 beats per minute. The query language allows arbitrarily complex constraints to be specified for satisfying queries in a playlist. XPL does not specify the method by which an application satisfies the constraints in `query` elements.

Another class of rules that can be used to describe tracks is weighted selection from one or more sub-playlists. Consider the playlists `fast.xpl` and `slow.xpl`, which contain songs above 120 bpm and below 120 bpm, respectively. A user can create a new playlist that blends the two playlists by using a "combination track" as follows.

```
<playlist name="random_mix" loop="1">
 <track name="track1">
```

```
  <combine>
   <playlist src="fast.xpl" weight=".6" />
   <playlist src="slow.xpl" weight=".4" />
  </combine>
 </track>
</playlist>
```

The `<combine>` element specifies that the content for the track should be randomly selected from the two external sub-playlists, with a .6 probability that the selection will be from `fast.xpl` and a .4 probability that it will be from `slow.xpl`. The loop attribute of the playlist is used to repeat the track and randomly select content until both playlists are exhausted. `Fast.xpl` and `slow.xpl` can themselves be dynamic playlists represented by the queries "`bpm > 120`" and "`bpm < 120`", respectively. Thus, a new more complex rule "60% of the songs should be greater than 120 bpm and 40% should be less than 120 bpm" is created.

2.3 Transitions

For some applications, it is not enough to simply declare what content to play next. A user might also want to control the *transition* between tracks, such as how to crossfade between the two tracks and whether or not to perform beat matching. Transitions can be included in a playlist by using a `<transition>` element between tracks:

```
<track><content src="funky.mp3" /></track>
<transition track1Start="3"
   fadeDurationMin="2"
   enableBeatMatching="1">
</transition>
<track><content src="groovy.mp3" /></track>
```

In the above example, a special transition is defined between the two tracks. It begins 3 seconds before the end of the first track, with the second track starting at this time. The duration of the crossfade is set at 2 seconds, and beat-matching is enabled. A complete list of available transition attributes is available in the XPL specification.

3. RENDERING XPL

An application that renders XPL playlists must include an XML parser that converts the textual representation into internal representation. The renderer must be able to iterate through the internal representation and render each explicitly specified or rule-based content file in order to a suitable output device, such as a soundcard, audio file or network port. Consider the following XPL representation:

```
<playlist name="mainlist">
   <track><content src="first.mp3" /></track>
   <transition …></transition>
   <playlist name="sublist" >
     <track><content src="second.mp3" /></track>
     <track><query>
        bpm > 150 and genre="Electronica"
     </query></track>
   </playlist>
</playlist>
```

The file is parsed and the interpreted tracks and transitions are then sequenced in depth-first order. A track that satisfies the constraints in the query specification is then located. The resulting sequence of content references is then sent to the application's rendering (i.e., playback) engine. If the engine supports transitions, the explicit transition is used to cross-fade between `first.mp3` and `second.mp3`

3.1 Metadata Support

In order to support the dynamic queries available in XPL, the renderer must have access to *metadata,* or data about the musical content in a user's library. Examples of metadata include textual elements such as track and album title, artist name, etc. as well as audio-derived data such as fingerprints or tempo specifications.

Such metadata can be made available to an XPL renderer via a metadata database, or *metabase*. The renderer sends each query to the metabase, which returns a list of content files that satisfy the query. The metabase acts as a central repository for metadata acquired from several sources, such as ID3 tags, CDDB records and audio analyses of the content. The acquisition and storage of metadata for use by XPL renderers is an application-dependent process. Multiple metabases are supported in XPL via the `database` and `language` attributes of `query` tags. In addition to support for multiple distributed metabases, these attributes allow queries to be defined that are not easily expressed via the default SQL-like language in XPL [2].

4. CONCLUSIONS AND FUTURE WORK

XPL provides an expressive, extensible and readable format for the specification and exchange of playlists. An XPL renderer has been successfully incorporated into a demonstration application codenamed "VDJ" that supports loading an rendering of static and dynamic XPL representations and user editing of queries and explicit transitions. All edit operations are live, allowing the user to dynamically shape the playlist in real time, and can also be saved to XPL files for later rendering and sharing with other users.

Successful rendering of dynamic XPL requires quality metadata. If tracks in a music library lack metadata or contain incorrect data (e.g., incorrect genre or tempo), rendered playlists will not match user expectations. Quality of metadata can be improved by use of authoritative quality-controlled repositories and linking of metadata to audio-derived features (e.g., fingerprints) [3].

Future work on XPL will include integration with content-based music-information-retrieval systems and distributed metabases.

5. ACKNOWLEDGMENTS

The author would like to thank the members of the Technovation department at Creative ATC for their contributions and feedback on the XPL specification and their impressive work turning the concepts of XPL and metadata-aware rendering into a real application.

6. REFERENCES

1. Chaudhary, A., *XPL Specification*, . 2002, Creative Advanced Technology Center: Scotts Valley, CA.
2. Reiss, J., J.-J. Aucouturier, and M. Sandler. *Efficient multidimensional searching routines for music information retrieval*. in *International Symposium on Music Information Retrieval*. 2001. Bloomington, IN.
3. Allamanche, E., *et al. Content-based identification of audio material using MPEG-7 low level description*. in *International Symposium on Music Information Retrieval*. 2001. Bloomington, ID. http://ismir2001.indiana.edu/pdf/allamanche.pdf.