Clubmixer: A Presentation Platform for MIR Projects

Alexander Schindler and Andreas Rauber

Department of Software Technology and Interactive Systems
Vienna University of Technology
{schindler,rauber}@ifs.tuwien.ac.at

Abstract. Evaluating solutions to many music IR problems – such as playlist generation, music similarity – in absence of formal evaluation measures frequently requires user studies to establish the benefits of one solution over the other. Building an according application framework to deploy and test user responses is a cumbersome and complex task. We present Clubmixer - an advanced client-server based audio system that could serve MIR researchers as presentation and prototyping platform. The project aims at providing a software framework that minimizes the effort of creating MIR based solutions. The open architecture and the use of open standards provide high flexibility for several MIR related areas (e.g. content based retrieval, collaborative retrieval, etc.). We describe the current state of the system and outline the main functionality as well as the advantages of Clubmixer for MIR research.

Keywords: MIR systems and infrastructure, user interfaces and music access

1 Introduction

Although the discipline of music information retrieval (MIR) has matured since the early 1990s, MIR technology is not yet as widely used as research would like to see it. One part of the challenge may lie in the gap between the availability of sophisticated algorithms and research results in the prototype stage that promise superior performance and advanced features, and the evaluation in how far these promises live up to their expectations and meet user demands. The challenge, in most cases, lies in the fact that the approaches resulting from sophisticated research need to be deployed within a real system environment offering a rather large number of – by now standard – features expected by users in addition to the functionality offered by the research prototype. Building and deploying such a complex system constitutes a significant challenge on its own, putting a significant burden on researchers in music IR.

Examples of publicly available MIR solutions are still rather limited. Nowadays computers already have the appropriate resources to analyze average sized private music collections with state of the art MIR technologies - Yet there are

virtually no software audio players, nor plugins for commonly available software implementing MIR technology, despite the existence of a significantly large number of research prototypes.

To gather broader acceptance and recognition outside the research community, new solutions have to be presented and made available in a commonly acceptable form. Prototypes should provide user interfaces that correspond to the look and feel of commonly available audio software.

To meet these goals we present *Clubmixer*, a cross platform client-server audio jukebox system that can serve as a presentation platform for MIR research prototypes. It offers a number of features that are expected by users as default requirements, both on the functional as well as user interface level. Combined with a flexible architecture, existing MIR solutions can be plugged in, offering a sophisticated basis for the evaluation and deployment of MIR solutions in a setting acceptable by consumers.

The remainder of this paper is organized as follows: Section 2 presents related work on MIR systems, Section 3 describes the Clubmixer system followed by some example scenarios how to implement MIR solutions in Clubmixer in Section 4. Section 5 presents our conclusions.

2 Related Work

A good summary of music information retrieval systems is presented by Typke et. al [9]. Several prototyping frameworks have been introduced, like the well known C++ software framework CLAM [1]. It offers tools and repositories, as well as visual components, which can be used to rapidly develop research prototypes in the audio and music domain. The rapid prototyping environment ChucK [2] is a high-level programming language for music and sound synthesis including content analyzing and learning frameworks. Jmir [5] is a free and open-source software suite for automatic music classification, including audio, symbolic and Web content feature extractors. These projects generally focus on providing algorithmic components, whereas Clubmixer aims at providing a representative user interface combined with an open framework where further solutions can be easily integrated. Kurth et. al [3] presented SyncPlayer - a client-server based framework for multimodal presentation of audio and associated music-related data, which is conceptually similar to Clubmixer. The multiuser concept, used by Clubmixer was also introduced by [6] and [8] for collaborative playlist generation. These projects focus on collaborative balanced playlist generation with little or no use of content based retrieval techniques.

Songbird[1] is a cross-platform media player built on the Mozilla application framework. [4] gives a brief introduction into Songbird and details how to write add-ons by the example of the automatic playlist generation and music library visualization add-on *Soundbite for Songbird*.

[1] http://www.getsongbird.com/

3 Clubmixer Framework

The Clubmixer framework is a cross platform audio jukebox system based on a client-server architecture. The chosen architecture provides the major advantage that common MIR related tasks such as processing intensive calculations or time consuming feature extractions, can be executed on a powerful computer while presentations or evaluations are carried out remotely over the network.

The framework provides also a client user interface that is aligned to the look and feel of currently available software audio players. This benefits rapid prototyping. Clubmixer client and server are both built on the Java Plugin Framework (JPF) and provide several points, where the software can be extended.

Further the possibility to spawn multiple distributed clients for a single server instance accounts for research areas related to collaborative information retrieval. Clubmixer is based on open standard Web technologies and protocols, which provides a maximum of flexibility for researchers in building or integrating client solutions on nearly every platform (e.g. on mobile phones, Web pages).

Music Information Retrieval is highly dependent on musical information that is extracted from multiple sources and stored as metadata. Clubmixer provides a predefined set of song metadata and uses a solid database system for data storage. MIR prototypes can access and extend the initial database schema and attach their extracted metadata directly to the media library. Though there are several Java implementations of audio content extraction algorithms !!!ZITATE!!!, Clubmixer is not limited to Java based solutions only. Clubmixer provides a console mode with full database access and a predefined command set. This set can be extended and provides a convenient way to write small commands that are sparsely executed and don't need a full integration into the framework (e.g. test or data import routines).

3.1 Clubmixer Server

Clubmixer Server is the main component of the framework with an intended usage as standard software audio player or jukebox system. Played back audio files (currently only MP3 and WAV audio are fully supported) have to be locally available on the hosting computer. The files are automatically imported from user-specified directories and the extracted metadata is stored in the media library. The standard import routines can be extended to extract any kind of metadata that is needed for a certain MIR prototype (see Section 3.1).

In accordance with the computational requirements of MIR research prototypes, the server could be hosted on a high-performce server. The application hides to the system tray and can even be run in a headless terminal. Only for launching the configuration window (see Fig. 1) a window manager is needed.

The following sections describe the subcomponents of Clubmixer Server. Some of these components provide extension points - defined program sections that can be functionally extended by plugins. Brief descriptions of these points as well as their benefits for MIR research are given.

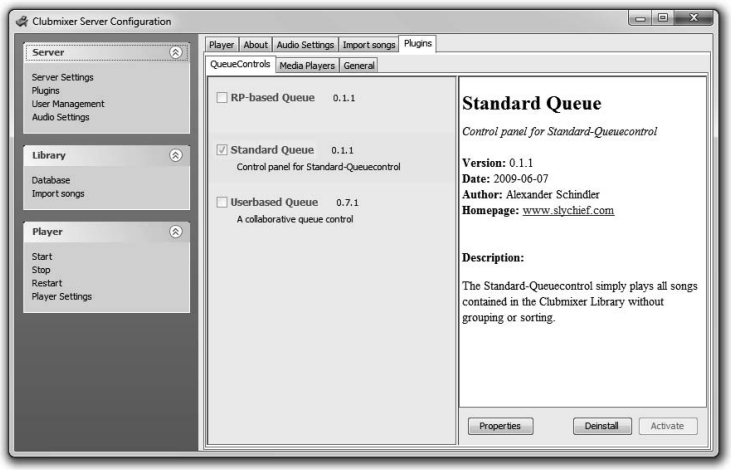

Fig. 1. Clubmixer Server Configuration Window

Data Storage. Clubmixer uses a Hibernate[2] persistence layer with a HyperSQL[3] (HSQL) database as data storage. This overcomes the commonly reported performance degradation on comparable audio software with huge song collections. The intermediate persistence layer additionally provides the advantage to exchange the underlying database system and to extend the initial database scheme by plugins. This gives developers the opportunity to store their data directly in the applications database.

Configuration properties (e.g. database connection settings) are stored separately utilizing the Java Preferences API, which stores these properties according to the underlying operating system (Windows Registry on Windows systems, config-files on Linux systems).

Communication. Communication is based on standard Web service technology. Three communication channels (see Fig. 2) are implemented as SOAP Web services. The first Web service provides standard audio player functionality (play, stop, next, add to playlist, etc.). The library service provides an extended search interface, with filters on multiple song attributes. All inputs are wildcarded to enable searching for keywords.

- **Control** - provides standard access controls for audio playback and playlist handling.
- **Library** - provides restricted access to the media library.
- **Plugin Communication Channel** - enables communication for plugins.

[2] http://www.hibernate.org
[3] http://hsqldb.org

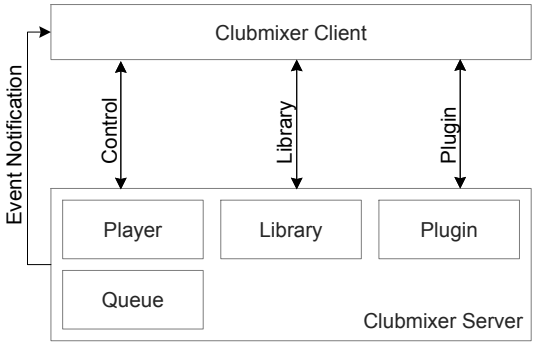

Fig. 2. Clubmixer Communication Channels

Special attention has been set on plugins, because they can also be split into a client and a server part. To enable intra-plugin communication, a custom Plugin Communication Channel (PCC) - a lightweight distributed middleware - has been introduced. The standard invocation of remote server-methods requires clients to wrap primitive parameters into sets of key-value pairs (e.g. in a HashTable). This approach provides best compatibility for non-Java clients. A special generic invocation technique is accessible if the parameters of the remote methods are JAXB[4] annotated data objects. Such objects can be directly marshaled into XML which in addition is passed on to the Web service. Further protocols (e.g. REST, JSON) are currently not supported, but can easily be added through plugins.

To propagate state change events (song change, playlist change) Clubmixer Server uses an Apache ActiveMQ message queue. Clients can subscribe to this queue and invoke Web service methods to synchronize their states accordingly. For example, if the client receives a 'song changed' event, it calls the Web service method `getCurrentSong()` to get the currently playing song.

Queue Control. A queue control is an extended automatic playlist generator. The general requirement is to provide a constant queue of songs. The standard queue, which just loads a randomized list of all songs in the library, enqueues each played song at the end of the playlist. The queue control is a defined extension point and a perfect starting point for MIR researchers to implement their solutions.

User Management. Clubmixer is a multiuser system which implements user management and access controls. Users can be assigned several access rights (e.g. start/stop the server, skip songs, etc.). This could offer a guest in a bar the possibility to search and enqueue a certain song but prevent him from stopping

[4] http://jaxb.dev.java.net/

the server. An integrated user management enables plugins to store user preferences. This can be addressed by MIR topics like playlist generation, collaborative filtering and relevance feedback.

Extension Points. Extension points are predefined points of the application, where the main functionality can be extended by plugins. Currently the following extension points are implemented:

- **General** - general functionality (e.g. opening sockets, running tasks)
- **Importer** - if the plugin requires additional data (e.g. audio feature vectors), a custom importer can be added.
- **Persistence** - extends the standard database schema by simply providing further JPA[5] annotated entity classes.
- **Queue Control** - provides a new queue control.
- **Console** - extends the standard command set of the console mode

A player extension point to exchange the current Java based player with native audio player implementations is planned. This will provide more flexibility and will overcome known issues concerning the Java audio libraries (Javalayer[6] and Tritonus[7]).

Annotation based dependency injection is used to provide all necessary functionality within plugins (e.g. database access, player control, etc.). Listing 1.1 gives an example of how to use annotations to get the references to the required components.

Console Mode. The console mode provides partially access to functions and services of Clubmxer server without the requirement of a fully running system. It further implements a set of commands to invoke certain parts of the server (e.g. loading plugins into the environment, starting the database). This command set can be easyly expanded - the provided interface invokes the ccommands and provides the supplied parrameters as string array - this is similar to standard main-methods of common programming languages.

3.2 Clubmixer Client

A Clubmixer Client acts only as a frontend to the server. It can be used to control the server and search for songs in the library. The aim is to provide a user interface that implements the average look and feel of currently available audio software. The default Clubmixer Client is a Java Swing client (see Fig. 3) which provides commonly known features of an audio software player. It can be used to control the playback of recordings, query for songs, manipulate playlists and display additional information about songs and artists.

[5] Java Persistence API, http://java.sun.com/javaee/technologies/persistence.jsp
[6] http://www.javazoom.net/javalayer/javalayer.html
[7] http://www.tritonus.org

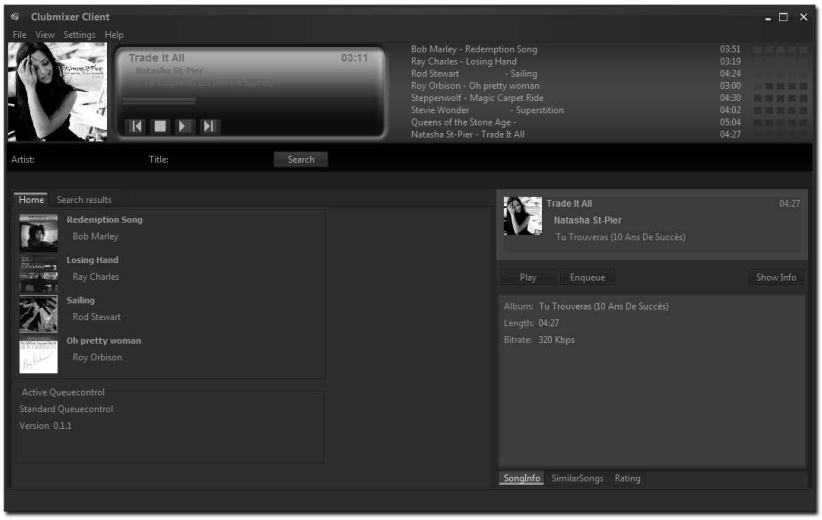

Fig. 3. Clubmixer Client

To enable fast development of client side plugins and to provide a common look and feel, several GUI elements are provided in a custom GUI components library - the *Common GUI Elements*. It provides among others, components to display song metadata with an albumart image, popup menus and diverse event handlers. Clubmixer Client currently provides three extension points, that can be used by plugins to add components for displaying data or to trigger server side methods.

Creating a Custom Client. Due to the open standards and libraries (SOAP, ActiveMQ) clients for Clubmixer Server can be implemented on almost every platform in almost any programming language. There are already initial implementation for Windows Mobile and JavaME. Plugin projects that are intended to provide information to non-Java clients should refrain from using complex data types as method parameter, due to the constraints described in Section 3.1

3.3 Clubmixer Library Editor

Clubmixer Library Editor (see Fig. 4) is intended to be the central place for querying and editing every information that is stored in the library. It provides a file system browser and displays imported metadata for MP3 files.

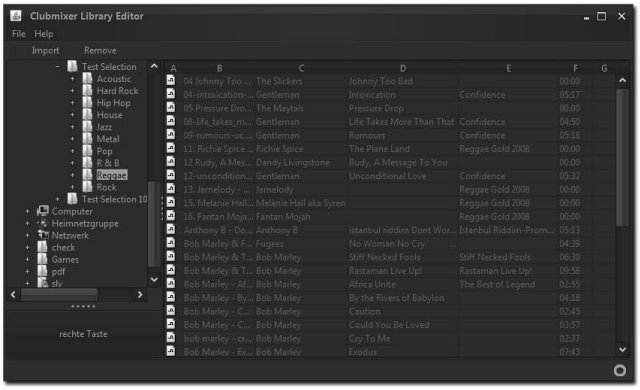

Fig. 4. Clubmixer Library Editor

4 Creating a new MIR based Clubmixer Plugin by Example

This section gives a brief overview of how to implement MIR projects as Clubmixer plugins. The exemplified scenario describes the common task of calculating song similarities. A content based solution has been applied which requires a plugin that takes advantage of several extension points.

The main component of a Clubmixer plugin is represented by a class that directly extends from the JPF-class *Plugin*. Listing 1.1 shows an example implementation of the fully operational plugin. It provides database access as well as a storage container for configuration properties.

```
1
2  public class MirPlugin extends Plugin {
3
4          @CommunicationChannel(pluginname = "MirPlugin")
5          private ICommunicationChannel com;
6
7          @ServerLibrary
8          private ClubmixerServerLibrary lib;
9
10         @Preferences
11         private ClubmixerPreferences  prefs;
12
13         public MirPlugin() {
14
15         }
16
17         public List<Song> findSimilar(Song s) {
18
19                 // MIR based algorithms
20                 ...
21
22         }
23
24 }
```

Listing 1.1. Example Plugin Implementation

To provide content based similarity calculations, further information has to be extracted from the audio files. Thus, the plugin has to provide a custom importer that extracts feature vectors and calculates the similarity matrices. To store this data efficiently, the the preexisting database schema has to be extended by a set of new entity classes.

The previous two paragraphs outlined all relevant code that has to be implemented at the server side. In order to display the extracted results, client side extension points have to be addressed. Fig. 5 a) shows a standard popup menu that is provided by the Common GUI Elements library. This popup menu can be linked to several song-related components and offers standard actions for the related song. It can be easily extended by adding further menu items. Fig. 5 b) shows the menu extended by the entry 'find similar' which offers to search for songs similar to the selected one. Listing 1.2 depicts the entire source code for this extension. The custom menu uses the Plugin Communication Channel to invoke the server side method 'findSimilar' from Listing 1.1, which processes the request and returns a list of similar songs. The retrieved result is passed on to a client component which is responsible for updating and displaying the search result table.

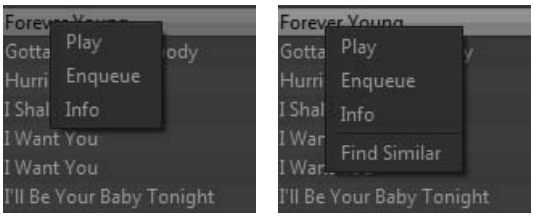

Fig. 5. Image a) shows the standard popup menu that is provided by the Commons GUI Elements library. Image b) shows the menu with an additional item that has been added by a plugin.

Figure 6 depicts the architecture of the plugin, which consists of the previously described components, required external libraries and the plugin description file. The latter advices the plugin framework, which libraries to load and which components correspond to which extension points.

```
1
2
3  public class MenuItem extends JMenuItem implements IMenuSong {
4
5          @CommunicationChannel(pluginname = "MirPlugin")
6          private ICommunicationChannel com;
7
8          @SearchresultHandler
9          private SearchResultHandler srh;
10
11
12   private Song currentSong;
```

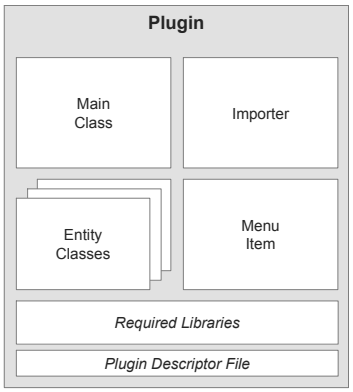

Fig. 6. Architecture of the Example Plugin.

```
13
14   public MenuItem() {
15
16     ActionMap map = ApplicationContext.getActionMap();
17     this.setAction(map.get("getSimilarSongs"));
18     this.setText("Find Similar");
19   }
20
21   @Override // from interface IMenuSong
22   public void setSong(Song song) {
23     this.currentSong = song;
24   }
25
26   @Action
27   public Task getSimilarSongs() {
28
29     // get reference to server method
30     GenericRemoteMethod<List<Song>, Song> findSimilar =
31         com.getGenericRemoteMethod("findSimilar");
32
33     // invoke remote method
34     List<Song> similarSongs = findSimilar.invoke(currentSong);
35
36     // output result list
37     srh.fireSearchResultChanged(similarSongs);
38
39   }
40 }
```

Listing 1.2. Extending the Standard Popup Menu

5 Conclusion and Future Work

The proposed Clubmixer framework is a solid and easy to extend audio player software, that has been developed in consideration of being used for music information retrieval research. It combines a good architecture with an appealing graphical user interface.

We have demonstrated how a MIR project can be turned into a Clubmixer plugin by only a few steps. Thus, new algorithms can be presented in an audio framework that incorporates the standard look and feel of currently available audio software.

There are various areas of application that are currently being investigated and evaluated:

- Combining the SOMejB Music Digital Library Project [7] and Clubmixer by integrating SOMejB as a customized plugin.
- Extending the Library Editor with analytical functions to statistically analyze extracted audio features.

6 Software and Source Code

Clubmixer is hosted as SourceForge project. Software installer packages and project source code can be found at http://sourceforge.net/projects/clubmixer/.

References

1. Xavier Amatriain, Pau Arumi, and David Garcia. A framework for efficient and rapid development of cross-platform audio applications. *Multimedia Systems*, 14(1):15–32, jun 2008.
2. Rebecca Fiebrink, Ge Wang, and Perry Cook. Support for mir prototyping and real-time applications in the chuck programming language. In *9th International Conference on Music Information Retrieval*, 2008.
3. Frank Kurth, Meinard Müller, David Damm, Christian Fremerey, Andreas Ribbrock, and Michael Clausen. Syncplayer - an advanced system for multimodal music access. In *ISMIR*, pages 381–388, 2005.
4. Steven Lloyd. Automatic playlist generation and music library visualisation with timbral similarity measures. Master's thesis, Queen Mary University of London, August 2009.
5. Cory McKay and Ichiro Fujinaga. jmir: Tools for automatic music classification. In *Proceedings of the International Computer Music Conference*, 2009.
6. Kenton O'Hara, Matthew Lipson, Marcel Jansen, Axel Unger, Huw Jeffries, and Peter Macer. Jukola: democratic music choice in a public space. In *DIS '04: Proceedings of the 5th conference on Designing interactive systems*, pages 145–154, New York, NY, USA, 2004. ACM.
7. Andreas Rauber, Elias Pampalk and Wolfdieter Merkl. The SOM-enhanced JukeBox: Organization and Visualization of Music Collections based on Perceptual Models.
8. David Sprague, Fuqu Wu, and Melanie Tory. Music selection using the partyvote democratic jukebox. In *AVI '08: Proceedings of the working conference on Advanced visual interfaces*, pages 433–436, New York, NY, USA, 2008. ACM.
9. Rainer Typke, Frans Wiering, and Remco C. Veltkamp. A survey of music information retrieval systems. In *IN ISMIR*, pages 153–160, 2005.