SHARING AND COMBINING LISTENING EXPERIENCE: A SOCIAL APPROACH TO WEB RADIO

Claudio Baccigalupo and Enric Plaza
IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia (Spain)
Vox: +34-93-5809570, Fax: +34-93-5809661
{claudio,enric}@iiia.csic.es

ABSTRACT

We present *Poolcasting*, a social Web radio architecture in which the group of listeners is able to influence in real time the music played on each channel. Poolcasting users contribute to the radio with songs they own, create radio channels and evaluate the proposed music, while an automatic intelligent technique schedules each channel with a group-customised sequence of musically associated songs. This technique works both with active users, who explicitly state their musical preferences, and with passive users, whose musical preferences are inferred from the analysis of their personal music libraries. The radio dynamically generates and streams on each channel a sequence of songs where the preferences of all the current listeners are combined together, with an approach that promotes fairness among listeners with different tastes.

1. INTRODUCTION

We present a network architecture that both simplifies the management of Web radios and improves the user listening experience and fidelity towards Web radio stations. This architecture, called *Poolcasting*, takes into account listeners not as a purely passive audience, but rather as a proactive public who is given the tools to interact and to collaborate, in order to adapt the music played on each channel according to their tastes.

In 2005, online radio listening increased 18 percent and free streaming of online music increased 37 percent [4]. About 52 million people tune in to Internet radios each month [1]. Several Web-based music recommenders have recently appeared providing users with personalised music streams, such as Pandora or Last.fm [1] . These recommenders *narrow*cast music, creating one personalised but isolated context for each user. On the contrary, Web radios as such are supposed to *broad*cast their signals to a large audience. Poolcasting merges both approaches, providing public channels that everyone can listen to, where the musical content of each channel is customised in real time for the whole *group of listeners*. At the same time, Poolcasting improves the administrators' work, who can spend less effort in set up and maintenance, and learn more about their listeners' musical preferences, to eventually focus on promoting new targeted music to their audience.

[1] Available at: http://www.pandora.com and http://www.last.fm

Figure 1. A Poolcasting Web radio.

2. SHARING LISTENING EXPERIENCE

Tuning in to a Poolcasting Web radio channel is as simple as navigating with a Web browser to the radio home-page (see Fig. 1) and clicking on one of the available channels, so that users who just want to passively listen to music can do so. However, users who want to actively participate in the selection of the music played can do so swiftly as well with the different functions offered by the Web interface.

First, Poolcasting introduces the ability for listeners to *share their personal music libraries*, that is, to contribute with songs they own to the collection of music the radio can broadcast. The advantages of this *distributed* approach over a classical approach — where songs are broadcast from a unique *centralised* library — are numerous. A distributed collection of songs is more dynamic (at each moment, it contains only the music of the participants who are sharing their libraries), more up-to-date (as users add new songs to their libraries, these can instantly be broadcast) and contains more "audio not available elsewhere" (personal libraries often include songs not pub-

licly distributed, such as personal audio material, uncommon records, alternative versions of known themes). To share one's music library and become an *Active Participant*, a user clicks on the Start Poolcasting link and indicates the path to the local folder where her music library is stored. Right now, sharing is implemented only for users whose library is managed using Apple iTunes and is stored in a folder accessible via HTTP. Poolcasting accesses this folder and retrieves the index file where library information is stored (a file called iTunes Music Library.xml for iTunes libraries). Once this file has been loaded, the title and artist of each track are matched against a public large music database to potentially obtain a unique ID and additional metadata (genre, tags, cover art); in our current implementation we use the catalogue of the Web-based music community MyStrands [2] which is made of over 6 million songs. From this moment on, all the songs in the user library are virtually included in the *Music Pool* of the radio. This means that, as long as the user is connected, Poolcasting can pick from her library any song to be played on any channel. No file is retrieved at this step.

Another advantage of having users share their personal libraries is that Poolcasting is able to *implicitly* infer the users' musical preferences by analysing *listening experience* data stored in the library index file. For instance Apple iTunes stores the play count of each song (how many times a song was played) and the explicit rating (1 to 5 stars) assigned by the user. Exploiting these data, Poolcasting builds a partial *preference model* for each active participant, under the intuitive assumption that the higher the rating assigned to a song and the higher its play count, the more the user likes that song. Briefly, the *preference degree* $g(P, S)$ of a participant P for a song S is estimated combining the rating that P assigned to S and the play count of S in the library of P, both normalised according to the average listening behaviour of P (average rating and average play count). For songs not included in the personal library of P, $g(P, S)$ is estimated following the assumption that if P does not own song S but owns other songs *from the same artist* of S, then her preference for S can be estimated as her average preference for those songs.

The Web interface also allows users to *explicitly* state their preferences towards any song played or scheduled on a channel. For each channel, Poolcasting provides a detailed Web page (see Fig. 2) with real-time information (current track, history of played tracks, upcoming tracks) and links to explicitly evaluate and request songs. When a participant assigns a rating to a song, Poolcasting accordingly updates her preference model for that song. When a participant requests a track to be played on a channel, Poolcasting updates her preference model as well, assigning a high preference to that track. Preference models (made of both implicit and explicit musical preferences) are internally used by Poolcasting to influence the music selection of the channel a participant is listening to, by playing more songs she probably likes (e.g., songs from her best-rated artists) and less songs she probably dislikes.

Poolcasting also introduces the ability for listeners to *create channels*. A listener creates a channel by clicking on the Create a new channel link and defining its name, description and *Channel Pool*, that is, the subset of songs that is allowed to play on that channel. The Channel Pool is defined in terms of metadata restrictions; for instance "Jazz from 1967" or "Electronic Dance at 135 BPM". The new channel appears in the list of the available channels, and is associated with a music stream that is automatically programmed with music that fulfils the specified restrictions.

3. COMBINING LISTENING EXPERIENCE

Poolcasting utilises the knowledge collected from the Web interface and the user libraries to automatically schedule and play on each channel a sequence of songs that possibly matches the taste of the listeners connected at each moment. Because of the dynamic nature of the Music Pool, Poolcasting cannot pre-build the whole sequence of songs that will play on a channel, but has to select in real time the song that will play next, based at each moment on the current set of songs shared by the participants. To select at time t the song that will be played next on a channel H, Poolcasting first builds a subset of *possible candidates*, made of all the songs that are in the Music Pool at time t and that match the Channel Pool of H (e.g., songs whose Genre is Jazz for a "Jazz" channel). Then, the candidates are ranked according to how much they are *musically associated* with the last song played on the same channel. This is intended to emulate the behaviour of terrestrial radios, where human DJs programme sequence of songs that musically "flow" one after the other. Finally, Poolcasting utilises the Preference Models to pick the candidate song that most satisfies the musical tastes of the current listeners of channel H.

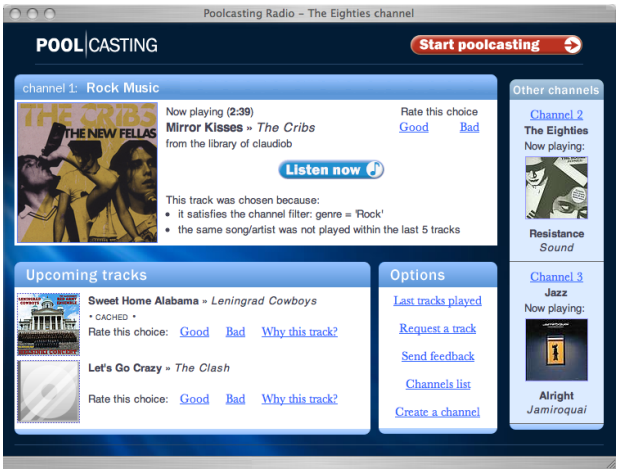

Figure 2. A Poolcasting Web radio channel.

[2] http://www.mystrands.com

3.1. Musical Associations Model

In order to programme for each channel a sequence of songs that musically "flow" one after the other, Poolcasting needs a musical knowledge about which songs can be seen as *musically associated*. While a human DJ gathers this knowledge from experience, Poolcasting utilises a pattern mining technique [2] that, by analysing the co-occurrences of songs and artists in a large collection of human-built playlists, gives an estimation of the *song association degree* of each pair of songs and of the *artist association degree* of each pair of artists. Briefly, this technique evaluates that two songs or artists are strongly associated (or else, musically "flow") when they occur together in many human-built playlists, few songs separate them in each of these playlists and they do not occur separately (that is, only one of the two occurs) in many playlists. We chose human-built playlists as a source of knowledge because they include cultural information that cannot be extracted from audio signal, and also contain songs in a specific order, which can be preserved when programming a radio channel.

3.2. Group customisation and satisfaction

In order to select the candidate song that most satisfy the group of listeners as a whole, Poolcasting ranks the candidates according to the Preference Models of the current listeners. A critical challenge is how to combine these individual preferences when they are contrasting. To solve this issue, we have elaborated an Average Without Misery [3] technique that considers *individual satisfaction*, so that in a limited amount of time, every participant in a channel gets eventually to listen to some songs she likes [2]. Briefly, this technique ranks each candidate song S calculating a weighted *average preference degree* for all the current listeners. In this average, songs that have a strongly negative preference for some listener are demoted, while the individual preference $g(P, S)$ of each listener P for each song S is biased with a weight that reflects how P was satisfied with the last songs played on the channel: if the previous songs matched (resp. did not match) her musical preferences, then her individual preferences have a weaker (resp. stronger) impact in the average.

3.3. The Song Scheduler

To schedule each channel with a group-customised sequence of musically associated songs, we have implemented a Case-Based Reasoning (CBR) technique [2], which is in our expectations the most efficient approach for scheduling many channels in real-time, balancing the contributions of both musical associations and listeners' preferences. This CBR technique works in three consecutive steps: 1) a *Retrieve Process*, where a new song has to be scheduled on a channel and the Musical Associations model described in Sect. 3.1 is used to retrieve from the Channel Pool a subset of candidate songs that are musically associated with the last scheduled song; 2) a *Reuse Process*, where Preference Models are used to sort the candidate songs and to pick the one that best matches the group preference of the current listeners, according to the strategy described in Sect. 3.2; 3) a *Revise Process*, where participants can evaluate the proposed song before it is streamed; when user feedback is strongly negative, the song is substituted in real-time for the next best candidate of the Reuse process.

3.4. Poolcasting at work

The architecture of a Poolcasting Web radio is illustrated in Fig. 3. A radio is idle until a participant (or an administrator) creates a channel using the Web interface. When this occurs, the *Streaming Server* opens a new Internet stream for the channel and waits for the *Stream Generator* to fill it with music. The Stream Generator, in turn, asks the *Song Scheduler* to select the next song to be played on the channel. The Song Scheduler checks in the *Database* the Channel Pool (the subset of songs of the *Music Pool* that fulfils the restrictions of the channel) and the *Preference Models* of the current listeners and selects a song X as described in Sect. 3.3; then connects via HTTP to the user library where X is stored and downloads the file into a local *Song Buffer*. In turn, the Stream Generator transforms that file into an uncompressed audio signal that the Streaming Server finally broadcasts to the listeners.

Once the channel is up and running, the Stream Generator has to continuously feed the Streaming Server with music; any delay would cause an annoying interruption in the stream broadcast to the audience. For this reason, the Stream Generator works "two songs in advance": while song X is being broadcast to the audience, and the next song Y has been cached into the Song Buffer, ready to be played, the Stream Generator asks the Song Scheduler to select a song Z that will play on the channel after Y. When X ends, Y is loaded to the Streaming Server, Z replaces Y in the Song Buffer, and a new song is selected by the Song Scheduler. With this approach, the Web radio can serve for each channel an uninterrupted music stream, without gaps between songs, even when a personal library becomes suddenly unavailable because a user has disconnected from the system. The Streaming Server also maintains a list of the IP addresses of the listeners of each channel; channels that have not had any listener are automatically disabled after some time, in order to save bandwidth.

The architecture described in this paper has been implemented and an experimental Poolcasting Web radio is currently running within our Institute. The radio runs on an Intel-based Mac Pro server and was developed with open source software and standard formats: MySQL for the Database, liquidsoap for the Stream Generator, icecast for the Streaming Server, Perl for both Song Scheduler and Library Analyser, Apache for the Web Server, PHP, XHTML/CSS for the Web interface. A private Web interface allows the administrators to start/stop the server, list the active channels, browse the listeners' Preference Models and update the Musical Associations Model. Each

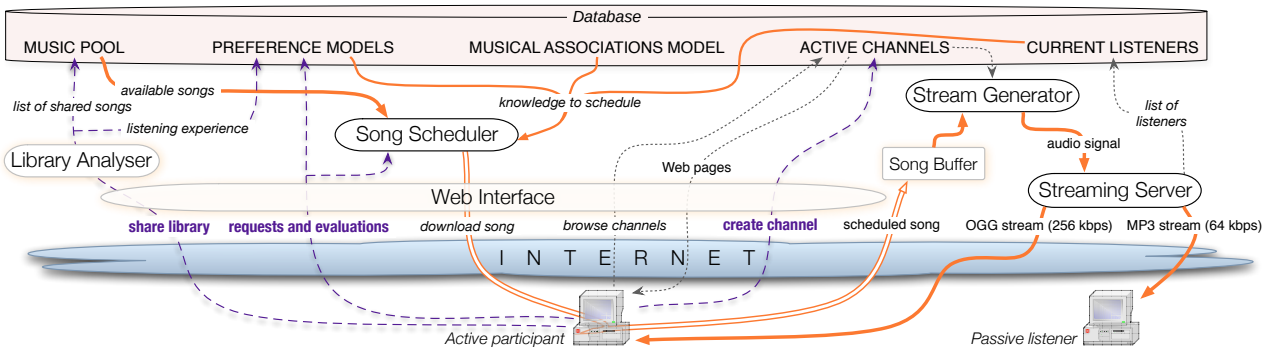

Figure 3. Architecture of Poolcasting.

channel is provided in both MP3 and OggVorbis encoding formats.

4. AN EXAMPLE SCENARIO

Let us consider the following scenario: a German musician performs as a DJ in a night club and needs to stay up-to-date with the last "Progressive Trance" hits (a subset of electronic dance music where BPM is within 125 and 135). He is disappointed for most "Dance" Web radios mainly broadcast mainstream songs. He enters Poolcasting, creates a niche-targeted "Progressive Trance" channel and starts listening to it. He also shares his library, which contains published tracks, copyright-free unpublished tracks produced by himself and live performances. Initially, the Channel Pool only contains themes from his library; this means he will hear a sequence of his songs, automatically selected and ordered without the need for him to manually compile a playlist. As more people from all over the world join the same channel and share their music, he suddenly gets to listen to Progressive Trance tracks that are completely unknown in Germany but might be appropriate for his upcoming performances. He will listen to songs he does not own, but all the same appreciates, mixed with other songs he already knows and likes. Meanwhile, he can request his own tracks to be played on the channel, which will be listened by a specific niche of public that will possibly appreciate his works better than a generic audience, and send feedback in real time using the Web interface. Positive ratings from the audience for the music he proposes will implicitly increase his reputation as a good human recommender in the context of Progressive Trance music. Music labels can also benefit from this architecture: to promote new dance hits, they just need to join the Poolcasting radio and share their own music libraries; Progressive Trance tracks will be automatically scheduled on the right channel and listened by the right group of participants.

5. CONCLUSIONS

Poolcasting is a social Web radio architecture where a) users can create channels that best match their tastes, b) user preferences are inferred from the listening experience data contained in their personal libraries and c) users can actively participate in the music scheduling process by sharing their personal libraries and by requesting or evaluating the scheduled songs.

Poolcasting provides a "group-oriented recommendation" service which endeavours to satisfy an entire community of listeners not by looking for *one* individual solution (a song) able to satisfy the whole group, but by generating a good *sequence* of songs, where different individual preferences are combined, and each participant gets to listen to both songs she knows (and will possibly like) and songs she does not know (and will probably appreciate, since they are liked by others in the same channel).

As in Web radios, music is streamed (not copied) to the audience, who may eventually buy the best of music they have just discovered. Web radio administrators are also facilitated by the Poolcasting architecture for they do not need to manually prepare a large collection of songs to stream; also, user interaction allows the administrators to instantly know the reaction and preference of the audience towards the music played.

6. ACKNOWLEDGEMENTS

This research is partially supported by the MID-CBR (TIN2006-15140-C03-01) project and by a MyStrands scholarship.

7. REFERENCES

[1] Arbitron-Edison Media Research. Web Radio Listening Up 50%. April 2006.

[2] C. Baccigalupo and E. Plaza. A Case-Based Song Scheduler for Group Customised Radio. In *Proceedings of the ICCBR '07 Conference*, 2007.

[3] J. Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Modeling and User-Adapted Interaction*, 14:37–85, 2004.

[4] The NPD Group, Inc. US Consumers 13+ Music Listening. May 2005.