A Case-Based Song Scheduler for Group Customised 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
Email: {claudio,enric}@iiia.csic.es

Abstract. This paper presents a knowledge-intensive Case-Based Reasoning system to generate a sequence of songs customised for a community of listeners. To select each song in the sequence, first a subset of songs musically associated with the last song of the sequence is retrieved from a music pool; then the preferences of the audience expressed as cases are reused to customise the selection for the group of listeners; finally listeners can revise their satisfaction (or lack thereof) for the songs they have heard. We have integrated this CBR system with *Poolcasting*, a social group-based Web radio architecture in which listeners can actively contribute to the music played and influence the channels programming process. The paper introduces the Poolcasting architecture, presents the CBR technique that tailors in real-time the music of each channel for the current audience, and discusses how this approach may radically improve the group-satisfaction of the audience for a Web radio.

1 Introduction

Although digital distribution has revolutionised the way in which we buy, sell and share music, not much has changed in the way we listen to songs in shared environments. In different situations, groups of people with similar tastes gather to listen to a unique stream of music, but none of these situations is customised to the audience. In a music club, for instance, a DJ can be too busy mixing to check the reaction of the public; in a radio, broadcasters have it difficult to meet the taste of all the listeners; with a juke-box, the available records can be very limited for the audience to appreciate; in a home-party, anyone can easily monopolise the control over the music, and songs can be played in any sequence, with annoying disruptions between genres.

In this paper, we present an interactive social framework to overcome the problems of a group scenario similar to the ones above, with the goal to improve the group-satisfaction of an audience. In short, we propose a novel group-based

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

Web radio architecture, called Poolcasting, where the music played on each channel is not pre-programmed, but influenced in real-time by the current audience. In addition, users can submit explicit preferences: via a Web interface they can request new songs to be played, evaluate the scheduled songs and send feedback about recently played ones. A main issue is how to guarantee fairness to the members of the audience with respect to the songs that are broadcast. For this purpose, we have implemented a CBR technique that schedules songs for each channel combining both musical requirements (such as variety and continuity) and listeners' preferences. In order to keep fairness in the presence of concurrent preferences, we use a strategy that favours those listeners that were less satisfied with the last songs played.

The contribution of this paper is two-fold. First we present the Poolcasting Web radio architecture, where users can interact to influence the music played. Then we present a CBR technique that, for each Web radio channel, schedules a sequence of songs customised towards the group-satisfaction of the listeners.

2 Poolcasting

Poolcasting is a novel framework providing a Web radio service, with an architecture based on group customisation and interaction. Poolcasting takes inspiration from home-parties, where participants can contribute with their own records to the *pool* of music and can control in turn which songs are played. In Poolcasting, any user can share her personal music digital library, adding her songs to *Music Pool*, and can interact via a Web interface, to evaluate the songs played and propose new songs to play (see Fig. 1). These interactions allow the sequence of songs played on each channel to be *customised* for the current listeners. Let us present an example of how a user can interact with a Poolcasting Web radio.

Example 1. Mark checks via the Web interface the list of channels of a Poolcasting radio and joins the *'80s Music Channel*, which has 3 other participants and a pool of 90 songs. The Reflex *(Duran Duran)* is currently playing, and True Blue *(Madonna)* has been scheduled to play next. Mark shares his music library: the songs he owns become part of the Music Pool of the radio. At some moment, the system has schedule which song to play after True Blue. First, it retrieves from the Music Pool a subset of songs that fit the channel context (songs from the '80s) and are musically associated with the last scheduled track; these are: Heaven Is A Place On Earth *(B. Carlisle)*, True Colors *(C. Lauper)* and Love Shack *(The B-52's)*. Next, the preferences of each listener towards these three songs are evaluated; this is done by analysing the content of each listener's library. For example, the system discovers that Mark does not have True Colors in his library, but owns other themes from C. Lauper, and has given them positive ratings; thus it deduces a preference of Mark for this song over the other two. Then, the system merges the individual preferences of all the listeners with a strategy that balances fairness; for instance, it schedules Mark's preferred song at this turn, because he has just entered the channel, but will favour someone else on the next turn.

Using the Web interface, Mark sees that True Colors has been scheduled; he approves of this choice and sends a positive feedback, which is stored as new knowledge acquired over his preferences. Other listeners give a positive rating to this choice as well; this reinforces the knowledge that (True Blue, True Colors) is a good association

Fig. 1. The Poolcasting radio Web Interface.

of songs for the *'80s Music Channel*. While listening to True Colors, another listener, Lisa, recalls that her library contains Time After Time *(C. Lauper)* performed by Miles Davis, and figures that other participants will like to hear this track, for they will listen to an uncommon version of a known song from the Eighties. Using the Web interface, Lisa recommends this song for the channel; her proposal is accepted and after a while the song is played. Mark gets to listen to one of his favourite songs in a version he was unaware of, and all the way appreciates very much. He assigns a positive rating to this choice, increasing both the association between this and the previous song, and the reputation of Lisa as a good recommender.

Thus, Poolcasting combines both bottom-up and top-down approaches: users can contribute to the available music and influence the programming, while the actual choice of music played is taken by a technique that combines knowledge about songs' associations and listeners' preferences. In the rest of this section, we will first outline the innovative components of Poolcasting that allow listeners to influence the music played (Sect. 2.1), and next the requirements for a technique able to customise the music for the current audience (Sect. 2.2).

2.1 The Poolcasting Web Radio Architecture

The two main components of a typical Web radio are the *Music Library* (a large static collection of songs in a digital format) and the *Streaming Server* (the machine where users connect from the Internet to listen to the radio). Many Web

radios have several channels; each corresponds to an Internet stream, where the Streaming Server continuously broadcasts songs from the Music Library. *Listeners* can connect to these streams with an appropriate stream-enabled media player. Two more components in a common Web radio are the *Song Scheduler* and the *Stream Generator*. The first is responsible for determining the sequence of songs for each channel, and generally is very simple, either random or time/genre-related (e.g., from 6pm to 8pm only play classic music). The second continuously retrieves the next scheduled song from the Music Library, transforms it in an uncompressed audio signal, and loads it to the Streaming Server, that will broadcast it once the previous song ends.

In the Poolcasting Web radio architecture (see Fig. 2), there is no centralised collection of audio files, but rather a virtual *Music Pool*, made of the songs contained in the personal music libraries shared by the participants. Another important difference is that the *Song Scheduler* does not just select each song to be played, but also has to connect via the Internet to the library containing that song and to download it in a local *Song Buffer*, from where the Streaming Server will read it once the previous song ends. The Song Buffer ensures that an uninterrupted music stream can be served, without gaps between songs. A central *Database* is continuously updated to keep track of the current participants, the songs they share and the channel they are listening to. Poolcasting offers a *Web Interface* where *Visitors* can check information about channels, and interact to share libraries, request songs and send feedback. The Poolcasting *Administrator* only controls the server components of the Web radio (e.g., restarting servers, administrating accesses, managing channels).

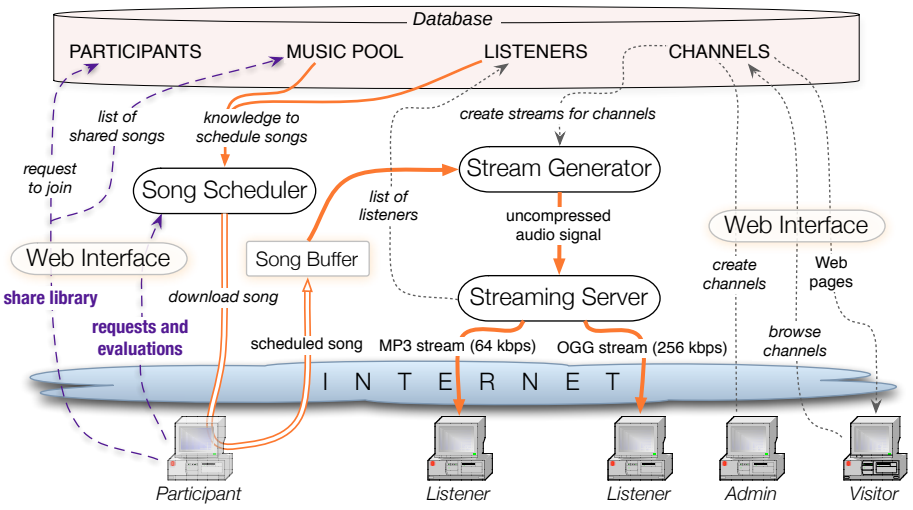

Fig. 2. Architecture of the Poolcasting Web radio.

2.2 The Task of the Song Scheduler

The Song Scheduler is responsible for programming the sequence of songs for each channel, following a policy of "scheduling two songs in an advance": while song X is playing, and song Y is in the local buffer ready to be played, song Z is decided by the Song Scheduler to play after Y. Once X ends, song Y is reproduced, song Z is downloaded to the local buffer (replacing Y), and a new song is scheduled to play after Z.

The goal of the Song Scheduler is to provide a satisfactory and customised listening experience to the participants that are simultaneously listening to a channel. To achieve this goal, we argue that a combination of four properties is required: 1) no song or artist should be repeated closely on a channel (*variety*); 2) each song should be musically associated with the song it follows (*continuity*); 3) each song should match the musical preferences of the current listeners, or at least of most of them (*individual satisfaction*); 4) the more a listener is unsatisfied with the songs recently streamed, the more her preferences should influence the selection of the next songs that will be played so that, throughout the whole broadcasting, she will listen to songs she likes (*fairness*).

The advantage of the Poolcasting architecture is that user interaction allows the Song Scheduler to *model the musical preferences* of each listener and exploit them to customise the content of the channels. In fact, the *explicit evaluations* made by the users via the Web interface offer the system an overview of the listeners' preferences (e.g., Mark approves of the selection of True Colors, the system infers that he likes this song). In addition, Poolcasting is able to work *without* any user interaction, by exploiting the *implicit knowledge* contained in the user shared music libraries in the form of *listening experience* data.

3 A Case-Based Reasoning Song Scheduler

We present now the Case-Based Reasoning technique we have developed to accomplish the task of the Song Scheduler in a Poolcasting Web radio. Let $\mathcal{P}(t)$ be the set of Participants at time t, let $\mathcal{L}(P)$ be the set of songs in the library of a Participant P, and let $\mathcal{C}(t)$ be the Music Pool at time t: $\mathcal{C}(t) = \bigcup_{P \in \mathcal{P}(t)} \mathcal{L}(P)$. Let H be a channel of the Web radio; let $\phi(H)$ be the Channel Pool of H, that is, the subset of songs of $\mathcal{C}(t)$ that comply with the definition of channel H (e.g., the Channel Pool of the *'80 Music Channel* contains only songs from 1980 to 1989). Let Y be the last song scheduled on channel H. The task of the Song Scheduler is to select, among all the songs in $\phi(H)$, a song Z to schedule after Y on channel H that satisfies the four properties above. To fulfil this goal, we employ a CBR approach that comprises three steps (see Fig. 3):

1. *(Retrieve Process)* Retrieves from $\phi(H)$ a subset of songs (the *retrieved set*) either recommended by some participant via the Web interface or that have not been played recently and are musically associated with Y.
2. *(Reuse Process)* Ranks the retrieved set combining the preferences of the current listeners, giving more importance to those listeners less satisfied with the

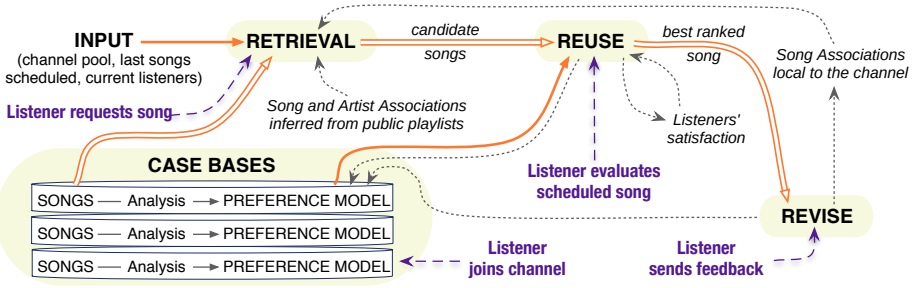

Fig. 3. The CBR schema.

music recently played on H; the song that best matches the four properties of Sect. 2.2 is scheduled to play on H after Y.

3. *(Revise Process)* Listeners can evaluate the songs played on H; a positive/negative feedback increases/decreases the degree of association of this song with the previous one played, relatively to channel H.

We consider the library of each participant as a Case Base. Each case is a tuple (song, artist, preference degree), where the preference degree reflects how much a participant likes a song. In Sect. 3.1 we will explain how, when a new user joins a channel, her musical preferences are inferred from the listening experience of the songs contained in her personal music library. In Sect. 3.2 we will explain the concept of musical association and how to infer which songs or artists are associated from the analysis of a large public collection of playlists. In Sect. 3.3 we will present the Retrieve Process, that selects from the Case Bases a subset of songs to achieve the goals of variety and continuity. In Sect. 3.4 we will detail the Reuse Process, that combines individual preferences to choose a song that fairly satisfies the group as a whole. Finally (Sect. 3.5), we will present the Revise Process, where users can evaluate the songs played on each channel.

3.1 The Participants' Case Bases

Every Case Base contains the list of songs in the shared library of a Participant, and a preference degree for each song. We define, for each participant $P \in \mathcal{P}(t)$, and for each song $S \in \mathcal{L}(P)$, a *preference degree* $g(P, S)$ with values in $[-1, 1]$, where -1 means P hates S, 1 means P loves S, and 0 reflects indifference. To assess the preference degrees of P, we use her library to extract information about her listening experience, namely the rating she assigned to each song and the number of times she listened to them. We assume that the higher the rating and the higher the play count, the stronger the preference. However, the *absolute* values of rating and play count are not relevant, for a "high" play count or rating for one user (e.g., 10 times, 3 stars) could be "low" for another user. For this reason, we normalise both values according to the average listener behaviour, in the following way. Let ϱ_{min} and ϱ_{max} be the minimum and maximum possible

ratings (e.g., 1 and 5 in iTunes), let $\hat{\varrho} = \frac{1}{2}(\varrho_{max} + \varrho_{min})$, let $\overline{\varrho_P}$ be the average rating assigned by P, and let $\varrho_{P,S}$ be the rating assigned by P to S; the *normalised rating* $n(P,S)$ of P for S is the function:

$$n(P,S) = \frac{2}{\varrho_{max} - \varrho_{min}} \left[\varrho_{P,S} - \hat{\varrho} - \frac{(\varrho_{P,S} - \varrho_{max})(\varrho_{P,S} - \varrho_{min})(\overline{\varrho_P} - \hat{\varrho})}{(\overline{\varrho_P} - \varrho_{max})(\overline{\varrho_P} - \varrho_{min})} \right]$$

that takes values in $[-1, 1]$ and equals 1 (respectively -1) when the absolute rating for S is ϱ_{max} (respectively ϱ_{min}). For any non-rated song, we define $n(P,S) = 0$.

Let ν_{min} and ν_{max} be the minimum and maximum play counts in the library of P, let $\hat{\nu} = \frac{1}{2}(\nu_{max} + \nu_{min})$, let $\overline{\nu_P}$ be the average play count of P, and let $\nu_{P,S}$ be the play count of song S; the *normalised play count* $m(P,S)$ of S for P is:

$$m(P,S) = \frac{2}{\nu_{max} - \nu_{min}} \left[\nu_{P,S} - \hat{\nu} - \frac{(\nu_{P,S} - \nu_{max})(\nu_{P,S} - \nu_{min})(\overline{\nu_P} - \hat{\nu})}{(\overline{\nu_P} - \nu_{max})(\overline{\nu_P} - \nu_{min})} \right] .$$

We assign $m(P,S) = 0$ if P has never listened to the song S. For any $S \in \mathcal{L}(P)$ present in the library of P, we define the *preference degree of* P as: $g(P,S) = \theta n(P,S) + (1-\theta) m(P,S)$, where θ is a parameter in $[0,1]$ to give more importance to the rating or to the play count (in our current implementation, $\theta = 0.5$).

This measure can be extended to songs not included in the library of P, following this assumption: if $\mathcal{L}(P)$ does not contain a song S but contains other songs *from the same artist* of S, then the preference of P for S is estimated as her average preference for those songs (e.g., Mark has rated positively many songs by C. Lauper, we assume he will like songs by C. Lauper he doesn't own as well). Let S be a song not included in the library of P, and let $\mathcal{G}(P,S)$ be the set of songs in $\mathcal{L}(P)$ from the same artist of S: $\mathcal{G}(P,S) = \{S' \in \mathcal{L}(P) \mid a(S') = a(S)\}$, where the function $a(S)$ returns the artist of song S. We define the *preference degree* of P for any song S as follows:

$$g(P,S) = \begin{cases} \theta n(P,S) + (1-\theta) m(P,S) & \text{if } S \in \mathcal{L}(P), \\ \frac{1}{\#(\mathcal{G}(P,S))} \sum_{S' \in \mathcal{G}(P,S)} g(P,S') & \text{if } S \notin \mathcal{L}(P) \wedge \#(\mathcal{G}(P,S)) > 0, \\ 0 & \text{otherwise} \end{cases}$$

3.2 Musical Domain Knowledge

One of the goal of the Song Scheduler is to program on each channel a sequence of musically associated songs (*continuity*). While a human DJ knows from experience which songs are associated, we use an automatic process to extract this knowledge from a large collection of playlists available on the Web. In brief, we check which songs and artists co-occur more often in these playlists, and assume that the more the playlists where they co-occur and the closer the distance at which they occur, the higher their association. Extracting such knowledge from playlists is much better for our goal than using a content-based method (e.g., extraction of acoustic features) because playlists include cultural and social information that cannot be reduced to audio signal, and also contain songs in a specific order, which can be preserved when scheduling songs for a radio channel.

Let $s(X, Y) \in [0, 1]$ be the *song association degree* from a song Y to a song Z. Counting just the frequency with which two songs appear together in a collection of playlists is not sufficient to estimate their association degree, for some songs are quite rare, but still are strongly associated with other rare songs. One solution is to consider the association strength from song X to song Y as the conditional probability to find song Y, given a playlist that contains song X, i.e., $P(Y|X) = \frac{f(X,Y)}{f(X)}$, where $f(X)$ is the *popularity* of X (number of playlists where X appears). Notice that $P(X|Y) \neq P(Y|X)$: the relation is not symmetric. This measure is biased towards having high conditional probabilities with songs that are very popular. That is, $P(Y|X)$ may be high, as a result of the fact that Y occurs very frequently and not because X and Y are strongly associated. We correct this problem dividing $P(Y|X)$ by a quantity that depends on the popularity of Y: if Y is very popular (say, more than the average), the association degree is decreased, otherwise it is increased; the exact degree of scaling depends on the playlists and on the distribution of popularity among songs. The following formula takes into account these factors to compute the association between two songs X and Y:

$$\frac{f(X, Y)}{f(X) \cdot (f(Y)/\overline{f})^{\beta}} \tag{1}$$

where $\overline{f}$ is the average song popularity, and β is a parameter that takes a value between 0 and 1; when $\beta = 0$, the function is identical to $P(Y|X)$.

We improve this measure by taking into account how far apart two songs are in a playlist, and their relative order. We make three assumptions: 1) the farther two songs occur in a playlist, the smaller is their association; 2) if two songs are separated by more than a threshold of $\delta \geqslant 1$ songs in a playlist, their association is null; 3) any song X is more associated to the songs it follows in a playlist than to the songs it precedes. The last point can be explained as follows: our final goal is to program a channel of music by incrementally adding one song *after* the other, and since the order between songs can be meaningful (e.g., the end of a track mixes into the beginning of the next one), we endeavour to preserve it.

Let $\mathcal{Q}$ be a collection of playlists and $q \in \mathcal{Q}$ be one of these playlists, $q = (S_1, S_2, \dots)$. Let X and Y be two songs; we denote as $d(q, X, Y)$ the distance that separates them in q, e.g., $d(q, S_i, S_j) = j - i$. If either X or Y does not occur in q, $d(q, X, Y) = \infty$. The songs X and Y are associated in q if $d(q, X, Y) \leqslant \delta$; formally we define their *song association degree in q* as:

$$w(q, X, Y) = \begin{cases} 0 & \text{if } |d(q, X, Y)| > \delta \\ 1/|d(q, X, Y)| & \text{if } |d(q, X, Y)| \leqslant \delta \wedge d(q, X, Y) > 0 \\ \alpha/|d(q, X, Y)| & \text{if } |d(q, X, Y)| \leqslant \delta \wedge d(q, X, Y) < 0 \end{cases}$$

where $\alpha \in (0, 1)$ is a parameter to assign higher associations to post-occurrences than to pre-occurrences. Finally, to estimate the *song association degree* between X and Y, we substitute in Eq. 1 the numerator with $\sum_{q \in \mathcal{Q}} w(p, X, Y)$. That is, rather than accumulating 1 for each playlist q where X and Y co-occur, we accumulate $w(q, X, Y)$, which equals 1 only if Y occurs contiguously after X in

q, otherwise $0 \leqslant w(q, X, Y) < 1$:

$$s(X, Y) = \frac{\sum_{q \in \mathcal{Q}} w(q, X, Y)}{f(X)(f(Y)/\overline{f})^{\beta}} \quad .$$

With this measure we have estimated the association for every pair of songs in a large database of playlists retrieved from the Web-based music community MyStrands (http://www.mystrands.com). We chose MyStrands because it offers a Web API called OpenStrands that helps us automate the retrieval process. The average length of the playlists was 17 songs; the average popularity was 37 for songs and 235 for artists. We set the parameters to: $\alpha = 0.75$, $\beta = 0.5$, $\delta = 3$ and ignored any song that occurred just once, to guarantee a valid statistical significance. We also discarded associations within the same artist, for their obviousness. The result was a set of 112,238 distinct songs that have a positive association with some other song; for instance, the top associated tracks found for Smoke On The Water *(Deep Purple)* were: Cold Metal *(Iggy Pop)*, Iron Man *(Black Sabbath)*, China Grove *(The Doobie Brothers)*, Crossroads *(Eric Clapton)*.

We have mined the same collection of playlists from MyStrands to gather knowledge about associated artists. Given a playlist $q = (S_1, S_2, \dots)$ and two artists A and B, we denote as $d'(q, A, B)$ the minimum distance that separates a song of A and a song of B in q, e.g., if $a(S_i) = A$ and $a(S_j) = B$, $d'(q, A, B) = j - i$. If q does not contain both a song from A and a song from B, then $d'(q, A, B) = \infty$. We define the *artist association degree in* q from A to B as: $w'(q, A, B) = \frac{1}{|d'(q,A,B)|}$ if $|d'(q, A, B)| \leqslant \delta'$, otherwise $w'(q, A, B) = 0$. Notice that the order is not important when we deal with artists. To estimate the *artist association degree* from any artist A to any artist B, we use an approach similar to the one used for the song association degree: we substitute in Eq. 1 the numerator with $\sum_{q \in \mathcal{Q}} w'(q, A, B)$ in the following way:

$$s'(A, B) = \frac{\sum_{q \in \mathcal{Q}} w'(q, A, B)}{f'(A)(f'(B)/\overline{f'})^{\beta}}$$

where $f'(A)$ is the number of playlists where any song by A appears, and $\overline{f'}$ is the average artist popularity. From the dataset of MyStrands, using $\delta' = 2$ as the maximum distance, $\alpha = 0.75$, $\beta = 0.5$, and ignoring any artist that occurred just once, we have obtained that 25,881 distinct artists have a positive association with some other artist. The value $\beta = 0.5$ was decided after several experiments, in order to obtain a nice mix of more and less popular artists in these associations. For instance, the top associated artists found for Abba were: Agnetha Faltskog, A-Teens, Chic, Gloria Gaynor, The 5th Dimension. Notice that the first two names (Agnetha Faltskog and A-Teens) are not very popular, but are very much associated with Abba: the first was their lead singer, the second is a cover band of Abba. As the sequence continues, more popular names appear, still associated with Abba, but in a weaker degree.

3.3 The Retrieve Process

This process has two subsequent steps: first each song $Z \in \phi(H)$ is rated with a *relevance value* $r(Y, Z)$ in $[0, 1]$ that expresses how much a song Z satisfies the conditions of variety and continuity; then the κ best rated songs are retrieved.

For every song Z requested via the Web interface, $r(Y, Z) = 1$, these songs are always retrieved. For every song Z either recently scheduled on H (namely, within the last ι songs) or from an artist recently scheduled on H (namely, within the last ζ songs), $r(Y, Z) = 0$, these songs are never retrieved. Notice that the values of ι and ζ are defined for each channel; for instance a "Frank Sinatra only" channel would be created with $\zeta = 0$ (artists repeated without reserve), while a "Nice Dance Mix" channel would probably have a high value for ι (songs rarely repeated).

For any other song Z (neither requested nor repeated), we define the relevance value on the basis of the musical association between Y and Z, as follows: $r(Y, Z) = s(Y, Z) + \epsilon u(Y, Z) + \epsilon^2 v(Y, Z) + \epsilon^3 s'(a(Y), a(Z))$, where $s(Y, Z)$ measures the song association from Y to Z, $u(Y, Z)$ evaluates the association from songs of the artist of Y to Z, $v(Y, Z)$ evaluates the association from songs of artists associated with the artist of Y to Z, $s'(a(Y), a(Z))$ measures the association from the artist of Y to the artist of Z, and the parameter ϵ in $[0, 1]$ controls the decreasing importance of these four conditions. Precisely, $u(Y, Z)$ is the average song association degree from every song whose artist is $a(Y)$ to Z: $u(Y, Z) = \frac{1}{\#(\mathcal{U}(Y,Z))} \sum_{W \in \mathcal{U}(Y,Z)} s(W, Z)$, where $\mathcal{U}(Y, Z) = \{W \in \mathcal{C}(t) \,|\, s(W, Z) > 0 \wedge a(Y) = a(W)\}$, and $v(Y, Z)$ is the average song association degree from every song whose artist is associated with $a(Y)$ to Z, combined with the relative artist association degree: $v(Y, Z) = \frac{1}{\#(\mathcal{V}(Y,Z))} \sum_{W \in \mathcal{V}(Y,Z)} (s(W, Z)\, s'(a(W), a(Y)))$, where $\mathcal{V}(Y, Z) = \{W \in \mathcal{C}(t) \,|\, s(W, Z) > 0 \wedge s'(a(W), a(Y)) > 0\}$.

The Retrieve process returns the first κ songs of $\phi(H)$ ranked along $r(Y, Z)$.

3.4 The Reuse Process

This process ranks the retrieved set according to the preferences of the current listeners of the channel and their "group satisfaction", and returns the best ranked song as the next song to be scheduled on the channel. The most critical challenge is how to combine different individual preferences into one group satisfaction value. To guarantee fairness among listeners, we propose a weighted average of the individual preferences, where the weight associated to each listener depends on her satisfaction about the last scheduled songs.

Let $\mathcal{O}(H, t) \subseteq \mathcal{P}(t)$ be the Participants who are listening to channel H at time t; let $\mathcal{R}(H, t) \subseteq \mathcal{C}(t)$ be the retrieved songs, and let $S \in \mathcal{R}(H, t)$ be one of these songs. The *group preference* of $\mathcal{O}(H, t)$ for S is a function $G(S, H, t)$ in $[-1, 1]$ defined by two cases:

(Average) if none of the current listeners *hates* song S (that is, all the individual preferences for S are beyond a threshold μ), then the group preference

is calculated as a weighted average of the individual preferences:

$$G(S,H,t) = \frac{1}{\#(\mathcal{O}(H,t))} \sum\nolimits_{P \in \mathcal{O}(H,t)} g(P,S)\,(1-\omega(P,H,t)) \qquad (2)$$

(WITHOUT MISERY) otherwise, if $\exists P \in \mathcal{O}(H,t) : g(P,S) < \mu$, then the group preference is set to the minimum possible value: $G(S,H,t) = -1$.

The weight $\omega(P,H,t) \in [0,1]$ in Eq. 2 is a function that biases the average in favour of the listeners more unsatisfied with the songs recently scheduled on channel H. Hereafter, we explain how the weight $\omega(P,H,t))$ is calculated. First, let us remark two important properties of this *Average Without Misery* [8] strategy: it is Pareto optimal (if at least one listener prefers S to S' and nobody prefers S' to S, then $G(S,H,t) \geqslant G(S',H,t))$ and it avoids misery: if at least one listener has a *bad* preference for S' (lower than a threshold μ), and no listener has a *bad* preference for S (lower than μ), then $G(S,H,t) \geqslant G(S',H,t)$.

The measure $\omega(P,H,t)$ estimates the individual *channel satisfaction degree* of a participant P at time t. To evaluate $\omega(P,H,t)$ we first need to know the satisfaction degree of P for each of the songs scheduled on H while P was listening. Let $\mathcal{X}(P,H,t) = (X_1, X_2, \ldots, X_z)$ be this set of songs (X_1 is the song scheduled when P entered the channel, X_z the last song scheduled), and let $X_i \in \mathcal{X}(P,H,t)$ be one of these songs, scheduled at a time $\hat{t} < t$; we define the *song satisfaction degree* of P for X_i as: $e(P,X_i,H) = g(P,X_i) - \max_{S \in \mathcal{R}(H,\hat{t})} g(P,S) + 1$. This function takes values in $[-1,1]$ and equals 1 only when the scheduled song X_i was the most preferred song by P in the retrieved set $\mathcal{R}(H,\hat{t})$.

By combining the song satisfaction degrees of P for the songs in $\mathcal{X}(P,H,t)$ we can estimate the value of $\omega(P,H,t)$. Since satisfaction is an emotion that wears off with time, we combine the satisfaction degrees assigning more importance to the most recent songs. To achieve this goal we use a geometric series: $\sum_{i=1}^{z} \chi^{z-i}\, e(P,X_i,H)$, where $\chi \in [0,1]$ measures the decay rate of satisfaction over time (e.g., $\chi = 0.8$). Since this series has values in $[-1,1]$, and we require $\omega(P,H,t)$ to have values in $[0,1]$, we rewrite the series normalised to this interval of values, and finally define the *channel satisfaction degree* for P as:

$$\omega(P,H,t) = \frac{1}{2}\left(\sum\nolimits_{i=1}^{z} \frac{\chi^{z-i+1}}{1-\chi}\, e(P,X_i,H) + 1\right) \quad .$$

Depending on this value, the individual preferences of P have more or less impact on the group preference at time t: the less satisfied is a listener with the songs previously scheduled on channel H, the more the Reuse Process endeavours to satisfy her with the current selection. This strategy is much fairer than a common *Plurality Voting* strategy, which would always select the item with more individual preferences, independently from the past satisfaction of users. The strategy we propose guarantees that every listener will eventually be satisfied during the broadcasting; Plurality Voting, on the other hand, would only satisfy the majority, eventually leaving the minority totally unsatisfied with the scheduling of a channel. We show this case with an example.

Example 2. A channel H has a Channel Pool of 8 songs $\phi(H) = (S1, \ldots, S8)$ and 3 listeners $(P1, P2, P3)$, whose individual preferences $g(Pi, Sj)$ are shown in the tables:

g(Pi,Sj)	S1	S2	S3	S4
P1	**0.8**	-0.2	0	0.2
P2	**0.6**	0.2	**0.6**	-0.8
P3	-0.2	**1**	0.4	0.8
G(Sj,H,t1)	**0.4**	0.3	0.3	-1

g(Pi,Sj)	S5	S6	S7	S8
P1	**0.6**	0.4	-0.4	-0.6
P2	**0.6**	-0.2	0	0.4
P3	0	0.8	**1**	-0.8
G(Sj,H,t2)	0.08	0.15	**0.17**	-1

At a time $t1$, the Retrieve Process returns the set $\mathcal{R}(H, t1) = (S1, S2, S3, S4)$, from which we have to select a song to schedule. First, we calculate the group preference degree $G(Sj, H, t1)$ for each of these songs (we set $\mu = -0.75$ and the initial channel satisfaction weights to 0.5), and schedule $S1$ because it is the most preferred song by the group: $G(S1, H, t1) = 0.4$. Then we calculate the listeners' satisfaction degrees; since $S1$ is the preferred song of both $P1$ and $P2$, their satisfaction degree is maximum: $e(P1, S1, H) = e(P2, S1, H) = 1$; however $S1$ is not the most preferred song of $P3$, so her satisfaction degree is smaller: $e(P3, S1, H) = -0.2 - 1 + 1 = -0.2$.
At a time $t2 > t1$, the new retrieved set is $\mathcal{R}(H, t2) = (S5, S6, S7, S8)$. This time, the channel satisfaction weights are not equal for all listeners: since $P3$ was previously unsatisfied, $P3$ is the listener with the smallest channel satisfaction: $\omega(P3, H, t2) = 0.42$, while $\omega(P1, H, t2) = \omega(P2, H, t2) = 0.9$ (we set $\chi = 0.8$). After calculating the group preference degree $G(Sj, H, t2)$ for each song in $\mathcal{R}(H, t2)$, we schedule $S7$ because it is the most preferred song by the group: $G(S7, H, t2) = 0.17$. Notice that $S7$ is the preferred song of $P3$, who fairly gets the highest satisfaction degree at this turn. On the contrary, a Plurality Voting strategy would have selected $S5$ in order to satisfy the majority of listeners ($S5$ is the preferred song at this turn of both $P1$ and $P2$), without memory of the fact that they had already been satisfied on the previous turn.

So far, the Reuse Process works without any interaction from the listeners. The retrieved set is ranked using only the *implicit knowledge* contained in the user personal libraries, and the best ranked song is scheduled. From this moment, participants can interact to explicitly state whether they like the selection made or not. As explained in Sect. 2.2, a certain time has to pass from when a song Z is scheduled to when it is actually broadcast. During this time, any listener P can send via the Web interface her explicit preference towards Z. If this occurs, the *implicit preference* $g(P, Z)$ that was stored in the Case Base of P (inferred from the music library) is replaced with this new *explicit evaluation* provided. For example, if P disapproves of the scheduling of Z, then the implicit value of $g(P, Z)$ in the Case Base of P is replaced with the explicit value -1. Next, since the Case Base has changed, the retrieved set is re-ranked to include this new value in the evaluation of the group preferences. This can lead to Z not being the most group-preferred song anymore; in this case, the scheduled song changes to the one that maximises the group preference. This process (user evaluations and re-ranking) continues until song Y starts playing on the channel. Then, Z is downloaded to the local buffer, and the CBR component restarts, to schedule the next song.

3.5 The Revise Process

While a song is playing on a channel, the Web interface shows its title, artist, cover-art, remaining time, and allows listeners to rate whether they like that song

or not (see Fig. 1). The assumption is that if a user rates positively (respectively negatively) a song played on a channel, then she likes (dislikes) that song and/or the song fits (does not fit) in the sequence of music programmed for that channel. Using this feedback, Poolcasting updates both the listeners' preference models and the musical knowledge about song associations.

When a listener sends a feedback about a song, the preference model in the Case Base of P is updated with this new explicit evaluation. For example, if P had never listened to song Z and sends a positive (resp. negative) feedback about it, the system learns that P has a high (low) preference for Z, and stores in her Case Base a new preference degree $g(P, Z) = 1$ ($g(P, Z) = -1$). As a result, the Reuse Process will be influenced by this new value, and eventually will (will not) schedule other songs associated with Z.

The feedback from the listeners is also used to revise the song associations extracted from the collection of playlists, and to customise them for the current channel. Indeed, two songs can be associated in one context, and not in another; for instance (True Blue, True Colors) is a meaningful association for a *'80 Music* channel, but not for a *Cheerful Tunes* channel. For this reason, Poolcasting builds a local domain knowledge model for each channel, where song associations *relative to that channel* are stored. Initially this domain model is empty, and only the associations inferred from the external playlists are used. As long as listeners send feedback about songs played on the channel, this model is updated accordingly. For example, if song Z is played after song Y on channel H, and listeners send negative feedback about it, the system learns that (Y, Z) is not a good song association *relatively to channel* H and locally updates the value of $s(Y, Z)$. As a result, the Retrieve Process for channel H will be influenced by this new value, and eventually will refrain from re-selecting song Z as a good successor for song Y on that channel.

4 Related Work

SmartRadio [5] employs a CBR approach to provide listeners with *individual personalised* radio channels; however the goal of Poolcasting is to provide *group-customised* radio channels. AdaptiveRadio [3] is a group-based Web radio where, if a listener shows discontent for a song, no other song from that album is broadcast. Thus, interaction is limited to vetoing songs, while Poolcasting users can also promote songs. Also, musical associations exist only for songs within an album, while in this paper we have expanded the approach introduced in [2] to build an extended musical associations model that contains song and artist association degrees, inferred from the co-occurrency analysis of a large collection of playlists. Virtual Jukebox [4] is another group-based radio, where the *majority* of votes (positive or negative) determines the preference of the group for the currently playing song. Poolcasting strategy is to combine *all* the listeners' preferences, favouring users less satisfied in the recent past. This mechanism increases fairness and is also easily understandable by the public — a favourable property for a group-aggregation technique according to [7].

MusicFX [9], CoCoA-Radio [1] and Flycasting [6] are three more systems focused on generating a sequence of songs that maximises the satisfaction of a group of listeners. The first broadcasts music in a gym centre attempting to maximise the "mean happiness" of the group; the second adapts the programming of a Web radio station according to the public; the third generates a playlist for an online radio based on the listeners' request histories. Users wishing to influence the music in these systems need to *explicitly* state their preferences, either by manually rating genres/songs, submitting a playlist as a proposal or requesting specific songs to be played. Poolcasting, on the other hand, allows users to both *implicitly* influence the music played (by sharing one's personal music library) and evaluate the proposed songs, which are substituted in real time for the next best candidates in the Reuse step if the feedback is strongly negative.

A Web-based group-customised CBR system is CATS [10], that helps a group of friends find a holiday package that satisfies the group as a whole. The task of CATS is to provide a good *one-shot* solution customised for the group, while the task of Poolcasting is to provide a good *sequence* of solutions, customised for the group over time. Also, CATS contains *one* CBR process, while in Poolcasting there are *multiple* CBR processes (one for each channel). Finally, the group of users in CATS does not change during the recommendation process, while Poolcasting participants are free to enter or leave at any moment.

5 Conclusions

The contribution of this paper is two-fold: we present a novel Web radio architecture called Poolcasting and a CBR technique to customise the content of each radio channel for the current audience. Poolcasting proposes a new paradigm for Web radios, shifting from a classical monolithic approach where "One controls, many listen", to a new decentralised approach where "Many control, many listen". The system is robust in the sense that it generates satisfactory results both for passive users (inferring their implicit preferences), and for active users (resulting in more customised channels). We have developed the internal components with open source software (Apache, MySQL, icecast, liquidsoap, tunequeue)[1], and the CBR process using Perl and PHP. A Poolcasting Web radio is currently running in our Intranet with three music channels and about 20 users. Our first tests show that users are willing to listen to songs not contained in their libraries to possibly discover new music they might like; however further tests are required, possibly in a public Internet environment, to deeply evaluate properties such as average user satisfaction, individual satisfactions, or user loyalty to channels. According to the existing copyright legislation, public Web radios pay a license fee related to the number of listeners or streamed songs, but independent from where the songs are stored. As such, deploying a public Poolcasting Web radio would require the same fees currently applied to Web radios.

Our contribution to CBR has several aspects. First, the Song Scheduler works with *multiple* participants' case bases and with domain knowledge acquired from

[1] Available at: apache.org, mysql.com, icecast.org, savonet.sf.net and tunequeue.sf.net.

playlists containing listening experiences of a large number of users. Moreover, the collection of case bases is open and dynamic: when a user enters (resp. leaves), the system immediately integrates (removes) her case base from the system, hence it responds at each moment to the *current* radio audience. Another contribution is using the Reuse process to combine data and preferences coming from different case bases (modelling users' listening experiences). Moreover, the goal of the Reuse process is to generate a *globally good sequence* of solutions over a period of time — not just one valid "group solution" for one problem. Our approach has been to view this solution as a trade-off between desirable properties for a radio channel (variety, continuity) and community-customisation properties such as individual satisfaction and fairness. Finally, both intensive knowledge and musical preference models are used in the Retrieve and Reuse processes, while user feedback is used in the Revise process to improve the customisation by updating these models in the CBR system.

Future work includes: testing the system with different parameters, evaluating the quality of the proposed technique, dealing with the issues of copyright and privacy, introducing a reputation degree for the listeners of a channel, extending the users' preference models with other listening experience data (e.g., personal playlists, time since a song was last played). Although the work we have described is specific to Web radio, we believe that the proposed idea of satisfying a group by guaranteeing both individual preferences and fairness among users can be applied to many other contexts where a group of persons gathers to listen to the same stream of music.

References

1. P. Avesani, P. Massa, M. Nori, and A. Susi. Collaborative radio community. In *Proc. of Adaptive Hypermedia*, 2002.
2. C. Baccigalupo and E. Plaza. Case-Based Sequential Ordering of Songs for Playlist Recommendation. In *Proc. of the ECCBR '06 Conference*, 2006.
3. D. L. Chao, J. Balthrop and S. Forrest. Adaptive Radio: Achieving Consensus Using Negative Preferences. In *Proc. of the GROUP '05 Conference*, 2005.
4. C. Drews and F. Pestoni. Virtual Jukebox. In *Proc. of the 35th Hawaii Intl. Conf. on System Sciences*, 2002.
5. C. Hayes, P. Cunningham, P. Clerkin and M. Grimaldi. Programme-Driven Music Radio. In *Proc. of the ECAI '02 Conference*, 2002.
6. D. B. Hauver and J. C. French. Flycasting: Using Collaborative Filtering to Generate a Playlist for Online Radio. In *Proc. of the WEDELMUSIC '01 Conf.*, 2001.
7. A. Jameson and B. Smyth. What a Difference a Group Makes: Web-Based Recommendations for Interrelated Users. In P. Brusilovsky, A. Kobsa, and W. Nejdl (eds.) *The Adaptive Web: Methods and Strategies of Web Personalization*, 2007.
8. 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.
9. J. F. McCarthy and T.D. Anagnost MusicFX: An Arbiter of Group Preferences for Computer Supported Collaborative Workouts. In *Proc. of the 1998 Computer Supported Cooperative Work Conference*, 1998.
10. K. McCarthy, L. McGinty, B. Smyth and M. Salamó. The Needs of the Many: A Case-Based Group Recommender System. In *Proc. of the ECCBR '06 Conf.*, 2006.