Media AI
Adaptive. Intelligent.

Architectural Design Document

Nick Burwell
CS 130 – Software Development
Thursday, December 16, 2004
Table of Contents

1. Introduction ..............................................................................................................1
2. Architecture .............................................................................................................1
3. Component Design ..................................................................................................2
   3.1 User login & administration ..............................................................................2
   3.2 The core .............................................................................................................2
   3.3 Playlist Generator .............................................................................................3
   3.4 Music Database Server ......................................................................................3
   3.5 Music Player ......................................................................................................3
4. Additional Information ............................................................................................4
Appendix A ..................................................................................................................5
   Overall view of system ............................................................................................5
   User interaction with the system .............................................................................6
1. Introduction

This document will contain the architectural design of Media AI. Media AI is a personalized music player that streams music to individuals’ desktop via a secure music server. Media AI consists of a client-side media player capable of a few actions (play, pause, skip, rate thumbs-up, rate thumbs-down, view song info, and log out) that connects to a music database. In efforts to be a legal product, the music catalog will not be accessible to the user, and the system will never allow more listeners at one time than the number of copies of a song that exist in the database. Media AI is personalized and adaptable because it continual improves each user's playlist the more they use the player. The system takes note of songs that are rated or skipped, as well as their types, and uses that information to adapt to a user's preferences. First, the overall architecture of the system will be described, and then the individual components will be described in further detail.

2. Architecture

The system will be built in an object-oriented fashion. Each module will be as independent of the other modules as possible, since this project will most likely evolve and continually grow in functionality. By keeping each part its own entity, it will make upgrades and additions easier. The backbone of Media AI will be called the “core.” It will be in charge of tracking logged in users, logging them out when they have timed out, checking if a requested song is available, resolving conflicts when two or more playlists request the last copy of a song at the same time, and make the necessary calls to the music database server to stream the music to each user's player. On top of this core sits the user login & user administration module, the playlist generator, and the music database server. The music player will have some of the login module integrated into it, as well as be able to connect to the playlist generator and music server.
3. Component Design

3.1 User login & administration

1. Purpose: stores all user information including userID, passwords, and other user data. Handles the login process, authentication, session tracking, the timeout feature if player is paused or stopped for a given amount of time.

2. Constraints: none found

3. Provided interface:
   a. Int log_in(loginInfo) – takes in login Info from player interface, and provides player and server with userID if successfully logged in; returns error code (probably 0) if login failed (used by Music Player)
   b. Int log_user_out(userID) – logs a given user out of the system; can be called from the core in case of a time out (used by the Core)
   c. Boolean is_logged_in(userID) – returns whether a given user is logged in and active or not (used by the Core)
   d. String get_user_info(userID) – returns the user's personal data (name, profile) (used by the Music Player)
   e. Boolean edit_user_info(userID, String newUserInfo) – allows editing of a user's personal data, returns true if successful, false otherwise (used by the Music Player)

4. Required interface:
   a. none

3.2 The core

1. Purpose: Acts as the mediator between the playlist generator and the music server. Also makes sure a user is logged in and has not timed out. If user has timed out (a song is paused for a given amount of time) then the system will log the user out and that song will again become available for other users. Resolves conflicts when two requests are made for the same song; keeps a queue of who gets what songs next, how many of each song is available. Basically, playlist sends a song to the system, which checks to see if that song is available, if so it sends the request along to the music server, which delivers it to the player.

2. Constraints: none found

3. Provided interface:
   a. SongData get_next_song(int userID) – called from the player, final return will command the server to stream data to user (used by Music Player)

4. Required interface:
   a. get_next_song_potential(userID)
   b. is_logged_in(userID)
   c. is_paused(userID, songID)
   d. log_user_out(userID)
3.3 Playlist Generator

1. Purpose: stores and sorts info on songs that are skipped, given thumbs-up, and given thumbs-down; generates a playlist for the given user, playing thumbs-up songs first and noting what genre and mood those songs are in. Adapts as songs progress; if a song is skipped, play a different genre; if one is liked, continue with songs in same genre; if listened all the way through, but not explicitly given a thumbs-up, play songs in the same mood but not necessarily the same genre. When a song is given the thumbs-down, it is skipped as well, and a new song is given. Songs that are skipped without a thumbs-down rating are negatively weighted less than songs that are explicitly rated with a thumbs-down, thus they have a more likely chance of being played again at some point. Keep totals of genres and moods as well as titles that have been skipped or given feedback.

2. Constraints: in order to be effective the user must actively use the player and ideally give feedback as much as possible, but even just the use of the player would be enough.

3. Provided Interface:
   a. Int get_next_song_potential(userID) – given a userID, uses the given feedback, current genre and tempo, and other rules to select next song to play, this selection (songID) is then sent to the core for verification (used by the Core).
   b. Boolean store_feedback(int userID, boolean thumbsUp) – takes a boolean that indicates whether the song is a thumbs-up or a thumbs-down for user corresponding to userID (used by the Music Player).

4. Required interface:
   a. None

3.4 Music Database Server

5. Purpose: stores all songs and song info, including title, artist, album, year, genre, tempo and mood.

6. Constraints: must reside on corporate-level server with access to the outside Internet.

7. Provided interface:
   a. SongData stream_song(int userID, int songID) – streams the actual sound file to the player for user with userID to player module (used by the Core).
   b. SongInfo get_song_info(int songID) - sends the song info to the player (used by the Music Player).
   c. Boolean pause_song(int userID) – pauses whatever is currently being streamed to the given user (used by the Music Player).

8. Required interface:
   a. None

3.5 Music Player

1. Purpose: Plays the music that is streamed to it from the server; allows user to rate the songs (thumbs-up or thumbs-down), view song information, pause/play songs as well as skip to the next song. The engine to play songs already exists; this module will consist mostly of the user interface and integrating it with the existing engine, the installer, and the ability to rate the songs.

2. Constraints: player must be downloaded to user’s computer and run locally; entire system will not function for a particular user without user being able to correctly download and install the player.
3. Provided interface:
   a. Boolean is_paused(int userID, int songID) – sends info back to server whether or not the song currently being sent to a user's player is paused or not; used in case there is a conflict with two playlists, and for when a user times out *(used by the Core)*

4. Required interface:
   a. log_in(String loginInfo)
   b. get_next_song(userID)
   c. store_feedback(userID, songID)
   d. pause_song(userID)

4. Additional Information

- Attached is a pictorial design of the modules, and their most basic interfaces
- Attached is a flow chart of what options the user has, and how the next song is selected
Appendix A

Overall view of system

The system is comprised of 5 modules. The player is a client-side, installable application which interacts with a server, using the user module. The core is the backbone and sends info back and forth between the user, playlist generator, and music server.
User interaction with the system

User options in blue. Pressing skip or listening to the entire song will enact “get next song” which then goes through a process of finding the next song. This process, while highly computational, should take a fraction of a second on today’s computers, so it can begin when the next song is requested—that way the current number of users listening to the song will be accurate; when a verified song is found, it is streamed to the player.