|  |
| --- |
|  ECSE 321 – Introduction to Software Engineering – Group 1  |
| Use Case Model & Architecture |
| Mission 3: High Level Design |

|  |
| --- |
| Samuel Cormier-Iijima (260174995), Simon Foucher (260223197), Stefanos Koskinas (260211145), Bertin LeBlanc (260191026), Amin Mirzaee (260209556) 4/4/2009 |

# 1. Use case model

## 1.1 Use Case diagrams

15 marks

**Stefanos Koskinas**

1.1.1. Diagrams should be consistent with the rest of the use case model (naming, etc.)

1.1.2. Proper use of <<include>>, <<extend>>, and inheritance relationships

## 1.2 List of ACTORS

*(Listed in order of frequency of interaction with the system)*

**User**: The user represents any preson who will be using the player. When the user uses the player, it is already installed and configured on a computer

**Installer**: This actor represents the person who will be installing the software on a PC. The installer has access to the music player’s “setup.exe” or .deb file and a PC running Windows 32 bits or Linux. The installer’s primary objective is to install the software.

**Hacker**: The hacker has the music player installed on his machine and wishes to steal playlists from other Remote Users without their consent, or regardless of the configuration of their players.

**Lawyer**: The lawyer wants to see if the player conforms to copyrights regulations and if not, sue the software developers for everything they are worth.

## 1.3 List of use cases

|  |  |
| --- | --- |
| **Audio format** | The **user** wants to use files encoded in xxx format. He install the appropriate audio cedecs. Afterwards, the player will recognize the audio files. |
| **Playback control** | The User is listening to a song, and he wants to stop, pause, seek. He presses on the appropriate buttons on the GUI. The song's playback is updated accordingly. |
| **Volume Control** | **User** is listening to a song and wishes to modify the volume of the player's audio output. He slides the volume slide bar or presses the mute button. The volume outputted by the player is updates accordingly. |
| **Equalizer** | **User** wishes to modify the gain of specific audio frequencies outputted by the player. He slides the equalizer buttons representing the band he wishes to amplify. |
| **Metadata Manager** | **User** wishes to create, update or delete any metadata associated with an audio file (i.e. song title, album, cover, etc…). The user enters the "edit metadata" menu and edits the songs he wishes. |
| **Custom Metadata** | **User** wants to create his own personal fields of metadata and name them. He clicks on the appropriate menu item and a dialogue opens. |
| **Rating** | **User** wants to save a ratings he sees fit for particular songs. He chooses "Edit Rating" in the menu function. |
| **Artwork** | **User** wants to change the artwork associated with an audio file. He selects the appropriate menu button and updates the image file. |
| **Auto Metadata Mgment** | The **User** wants the player to automatically manage the metadata associated with the songs. He enters that choice in the configuration option of the menu. The player will extract the metadata it can from the audio file and update it on the player. |
| **Playlist Add** | **User** wishes to add songs to a playlist. He clicks the song from the main view and drags it to the playlist. |
| **Playlist Remove** | **User** wishes to delete a song from a playlist. He selects the playlist such that it is visible in the main window. He selects the undesired song and clicks on it to highlight it, presses "delete" then "save playlist". |
| **Sorting Playlist** | The **user** wants to change the order in which the songs are played within a playlist. He clicks and drags them up and down. If he wants to make those changes permanent, he clicks on the "save playlist" button |
| **Search** | **User** wants to search for a particular song within a playlist. He enters the song name in the search field and presses "search" button. The playlist containing the song will appear in the main view with the song highlighted. |
| **Import Song** | The **User** can import a file/directory on the local hard drive by opening the "import window" from the menu. Importing an audio file will cause that file to be added to the library. Importing a directory will cause all audio files contained recursively within it to be added to the library. |
| **Default Library** | The **user** wants to have a default import library for the songs imported. He enters the configuration panel via the menu and chooses default library. |
| **Network song access** | **User** winches to select which network location will be browsed for songs. This is done via the configuration panel, accessed from the menu. |
| **Remote song import** | Once viewing the "Network songs", the **user** can select specific files and click on the "import" button to import those files to his local song library. |
| **Share songs/playlists** | In the configuration view (accessed from the menu), the **user** can select weather or not he wishes to share particular songs/playlists |
| **Auto-Sync** | The **user** wants one of his playlist to synchronize automatically with a remote repository. He selects the option in playlist configurations. |
| **Help** | The **user** needs help with some of the software's features. He selects the help entry in the menu and a help window pops up. He can then scroll the document and use search function. |
| **QA** | The **user** wants to listen to undistorted music and not have to deal with signal analysis. |
| **Consistent Gain** | The **user** wants to hear songs at a stable level without having to continuously readjust volume and equalizer |
| **Volume Display** | The **user** wants to be able to see what the volume is set to within the main view. |
| **Equalizer Display** | The **user** wants to be able to see the current frequency gain settings before updating them. |
| **Now Playing** | The **user** wants to get intuitive information the track playing. He selects the "now playing" playlist and sees the song being played in highlighted. |
| **Playback State** | The **user** can get information on the current playback state by looking at the "time left" at the bottom of the player. If the song is playing, the time remaining is shrinking. |
| **Active track** | The **user** can see the Title, Album and Artist associated with the current track playing by looking at the top of the player |
| **Artwork** | The **user** can see the Artwork associated with the file currently playing by looking at the pot right corner of the player, where it is displayed. |
| **Meta Sorting** | The **user** wants to be able to brows his playlist when sorted based on a metadata item. He clicks sort by "meta tag" and the list of songs within the visible playlist are sorted by the meta Tag selected. |
| **Track List** | The **user** can see all the tracks within a playlist by selecting that playlist. The list of tracks will appear in the main vies and he will be able to navigate using a standard sidebar. |
| **Aggregate Playlist Information** | The **user** can learn the total number of songs within a playlist as well as its total duration by looking at the bottom of the player once the playlist is within the main window. |
| **Current order** | The **user** can temporarily change the order of songs within a playlist. To view the current order, he can click on "now playing" and will see a clone of the playlist he originally selected to play. |
| **Active library** | The **user** can see all the songs within the active library by clicking on "now playing". This content will be displayed within the main window. |
| **Location information** | The **user** can see in the main window weather the music files present on the playlist are located on his local machine on at a remote location |
| **Song stats** | The **user** can get playback information on any songs present within his library by selecting the view entire library tab and looking at the appropriate column |
| **Remote Locations** | The **user** can see a list of available remote locations from which we may extract songs. This list is accessible from a menu function. |
| **Remote files available.** | The **user** can click on a remote location which is available, and a list of available songs will appear within the main window. |
| **Remote meta Tags** | When viewing a song playlist from a remote location, the **user** will also see all the available metadata in the main view. |
| **Favorite "Friends"**  | The **user** can specify a list of known remote network location for future quick access. This is done in the remote locations panel. |
| **Documentation** | The **user** can consult documentation relevant to the player by selecting the "help" function in the menu. |
| **Crawler** | The **user** wants the player to keep an updated list of available remote resources on the network without having to continuously refresh and track down location manually. |
| **Auto Sync** | The **User** wants the player to synchronize automatically with certain pre determined remote locations. |
| **Auto Update Metadata** | When the **user** changes the metadata associated with a song, the player automatically updates the audio file to reflect that change. Vice versa, whenever a song is refreshed, its metadata is read from the file and displayed on the screen. |
| **Auto Playback Statistics** | The **user** doesn't have to do anything in particular for playback statistics to get updated. The playback statistics are automatically updated and displayed. |
| **Loading from DB** | When the music player is started by the **user,** the default music library's information is automatically fetched from the DB. |
| **Initial set up** | When the **installer** installs the music player on a computer, a Data Base is created to store library information. |
| **Initialization of services** | When the **user** loads the music player, the audio filters, network utilities are set up. The player also caches data relevant to the music library to speed up future operations. |
| **Background** | The **user** does not have to worry about any kind of background operations involving the host environment. |
| **Backup Library** | The **user** can export the contents of his library by selecting this menu item. The backups created will be importable from any other players. |
| **Open Metadata Editor** | The **user** can open the metadata editor window by selecting this menu item. Then clicked, a separate window will open (see metadata Window below for more details) |
| **Exiting the player** | The **user** can close the player using this button. A safe close will ensure that all unsaved data gets saved and that buffers get flushed to the database before terminating the process. |
| **Playlist Picker** | The **user** can toggle between viewing different playlists and the main song library using this area. |
| **Playlist viewer** | The **user** can see all the songs present within a playlist, as well as their respective metadata in this area. |
| **Double Click Play** | The **user** can play a song simply by double clicking on it in the playlist viewer. |
| **Quick Add** | The **user** can add a song to a playlist by selecting it in the playlist viewer and dropping it in the desired playlist within the playlist picker. |
| **Edit Metadata** | The **user** can open the metadata editor window from the menu after highlighting a track. Once open, the user will be able to edit the song title, artist, Album, Year, Track number, Genre, Rating, Custom Tags. After editing, pressing the "save" button will record the new data within the Database. |
| **Intuitive Use** | The **user** should be able to figure out how to use the music player intuitively, based on previous experience with other players. Otherwise, full documentation is available to describe its functions. |
| **Package** | When installing the software, in addition to the music player, the **installer** will install help documentation. |
| **Performance** | The **user** can load a playlist containing 1000 songs within 1 second on a computer meeting the minimum system's requirements of the player. |
| **Reliability** | The **user** can terminate the music player process abruptly without any effect on the audio files and data base content on the local/network drives |
| **Security** | The **Hacker** will not be able to access the user's data if the User did not select the share option for that data. |
| **Portability** | The **Installer** will be able to install the player on a Windows 32 platform using the executable, or on a Linux platform using the .deb package. |
| **Legality** | The **Lawyer** will not be able to sue the software developers on the basis of Canadian computer software regulations. |

## 1.4 Detailed description of 15 use cases

25 marks

**Bertin Leblanc**

1.4.1. Most important use cases are identified

1.4.2. Flow of events is clear and concise with an appropriate level of detail

1.4.3. Entry and exit conditions are clearly defined

# 2. architecture

##  2.1 system architecture and sub-systems

 10 marks

**Stefanos Koskinas**

Explain the general structure of your system and the architecture(s) you decided to use. What is the rationale for your choices? What are the main subsystems and what service do they provide? Add a diagram that describes the main subsystems and the relationship between them.

2.1.1. Clearly explains the service provided by each subsystem

2.1.2. Clarifies relationship between subsystems

## 2.2 rationals for the chosen architecture

5 marks

 **Samuel Cormier-Iijima**

2.2.1. Lists any tradeoffs taken into consideration

2.2.2. Lists advantages and disadvantages of the architecture chosen

# 3. static model

In this part you should describe the preliminary (system-level) static structure of the system. Focus on finding the main classes in the system, their attributes, and their associations with other classes. **Note:** There is no need to identify and specify the methods of the classes at this stage. Please perform domain analysis and identify classes that will be used in the system. Use the requirements and use-case documents as an aid. When discussing GUI classes, keep the level of details manageable. (There is no need to describe scroll bars or label text locations.) For this sub-task you should submit the following:

1. List of main classes. For each class you should include:

a. A meaningful name

b. Role – describe what it does

c. Important attributes and methods

d. Backward traceability – reference to the use case or requirements that were used to come up with the class

2. Class diagrams. Describe the classes used in the system and the relationship between them. Remember, at this point, this is not a detailed design. No need for design patterns or other forms of reuse at this point.

Prepare tidy and clear diagrams. Messy or unreadable diagrams will be penalized. Pay attention to associations between classes, and multiplicity. Try not to have too many classes; the focus at this point is on major classes so you do not need to include smaller classes (e.g., Date). Follow the instructions that were taught in class and in the tutorials.

## 3.1 list of main classes

10 marks

 **Amin Mirzaee**

3.1.1. Only important, meaningful classes are included

3.1.2. Each class is properly named

3.1.3. The role of each class is given

3.1.4. Traceability – references are given to use cases that involve the class

## 3.2 classe diagrams

10 marks

**Amin Mirzaee**

3.2.1. Diagrams are clear, correct, and consistent with the rest of the document

3.2.2. Aggregation, composition, and generalization relationships are used properly.

# 4. requirement tracability matrix

For every requirement listed in your requirement specification document, show how it will be implemented in use cases and in classes. That is, for every requirement (or group of requirements, if convenient), you must ensure that it is implemented by some class, and that it appears in a use-case (for functional requirements). Include a table where each column is a requirement, each row is a use case or class. For each column, indicate with marks in the appropriate cells which use-case(s) address that requirement, which classes are involved in its implementation. This will later be a useful tool for testing use cases, and verifying that all of the requirements have been properly implemented by your system.

## 4.1 tracability matrix

15 marks

**Simon Foucher (Link Use Cases to requirements)**

**Samuel Cormier-Iijima (Link use cases to classes)**

4.1.1. Complete – includes all use cases from part A, all classes from part C, and all requirements as listed in the most recent version of the requirements spec document.

4.1.2. Comprehensive – every requirement and class is illustrated or used in at least one use-case.