Top-Level Design

Contents

1 The XMMS2 Client Project **3**
1.1 Project Description . . . 3
1.2 Target Audience . . . 3
1.3 Goals . . . 4

2 Requirements **4**
2.1 Features . . . 4
2.1.1 Critical Features . . . 4
2.1.2 Important Features . . . 4
2.1.3 Highly Desirable Features . . . 5
2.1.4 Nice to Have Features . . . 5
2.1.5 Future Plans . . . 5
2.2 Other Requirements . . . 6

3 System Components **7**
3.1 Interactions Diagram . . . 7
3.2 Explanation . . . 7
3.3 Component Descriptions . . . 8
3.3.1 Main GUI . . . 8
3.3.2 Search Bar . . . 8
3.3.3 Collection List . . . 8
3.3.4 Song Table . . . 8
3.3.5 Current Playlist . . . 8
3.3.6 Current Song . . . 9
3.3.7 Playlist Manager . . . 9
3.3.8 Collection Manager . . . 9
3.3.9 XMMS2 API . . . 9
3.3.10 Stable Storage . . . 9

4 User Interface **10**
4.1 GUI Mockup . . . 10
4.2 Prominent Features . . . 10
4.2.1 The Search Bar . . . 10
4.2.2 The Collection List . . . 11
4.2.3 The Song Table . . . 11
4.2.4 The Current Playlist . . . 11
4.2.5 Current Song . . . 11
4.3 Discussion . . . 11

5 Risks, Challenges, and Dependencies **12**
5.1 The Learning Curve . . . 12
5.2 Required Installation . . . 12
5.3 External Dependencies: XMMS2 . . . 13
5.4 External Dependencies: QT . . . 13

6 Plan of Action **13**
6.1 Division of Labor . . . 13
6.2 Testing . . . 14
6.3 File Organization . . . 14
6.4 Build Tools . . . 14
6.4.1 Building . . . 14
6.4.2 Version Control . . . 15
6.4.3 GUI Building . . . 15
6.4.4 Documentation . . . 15
6.5 Documentation . . . 15

1 The XMMS2 Client Project

1.1 Project Description

XMMS2 is an open source project aimed at creating the next generation music player/manager. Currently, the project exists as a music database and playback server with an elaborate API for client applications. Essentially, it is a back end to a music player that handles looking up files, decoding various audio formats, and outputting sound under a veriety of system environments. There are several client programs for XMMS2 currently under development, but many of them are incomplete and unusuable, and none of them fully take advantage of the capabilities of the XMMS2 database.

The XMMS2 project was created because the current options in music management and playback software were considered limited and unsatisfactory by advanced users. The project was designed to be very powerful, but also abstract. It is meant as a starting point for a wide variety of music projects that offers much potential, but doesn't apply many limitations to the end product.

What this XMMS2 Client Project aims to create is a GUI client for XMMS2 that will improve on current music management software options in several key ways. This new project will mainly be useful for users with very large collections of music. It will make both finding individual songs and constructing playlists faster, easier, and more natural. It will do this primarily through use of Collections[1].

1.2 Target Audience

This project is aimed at users who meet any of the following criteria:

- Have a large collection of music
- Have difficulty organizing their music
- Have difficulty finding songs
- Have difficulty remembering all the songs they own
- Are dissatisfied with current music management solutions

This project will be only for users of Linux. Although XMMS2 currently supports a few other unix-like operating systems and QT also supports Windows and Mac OSX, the complexities of designing a project to work accross multiple platforms are beyond the scope of this project in CS190. Perhaps if this project is a success, it could be ported to different OSes in the future.

[1]The XMMS2 Project defines collections merely as unordered subsets of a user's music library. Collections can be used to group related songs together arbitrarily and without any hierarchical limitations. New collections can be generated from the intersections and unions of other collections. This is the new key concept that the XMMS2 Developers hope will revolutionize how electronic music is managed.

1.3 Goals

- Play music.
- Be simple enough to accomodate most computer users.
- Provide methods to quickly find songs in the user's library and organize them into playlists.
- Fully implement collections. (No current XMMS2 client does this)

2 Requirements

2.1 Features

The following are the desired features in order of priority.

2.1.1 Critical Features

1. Play music
2. Add/Remove songs to the user's library
3. Browse the user's library
4. Search the user's library by ID3 tag information
5. Construct playlists by hand
6. Save and Load playlists
7. Provide an easy-to-use GUI

2.1.2 Important Features

1. Browse the user's library by collection
2. Provide meta-data collections (most recently played, play frequency, etc.)
3. Allow for user-constructed collections
4. Add and remove songs from user-constructed collections quickly with minimal effort
5. Generate playlists from collections

2.1.3 Highly Desirable Features

1. Generate new collections from set operations on existing collections
2. Allow for logical collections (generated on the fly from a filter or from set operations)
3. Allow for mixed logical and hand-picked collections (Say you take two collections A and B and find their union to generate collection C. If you add song s_1 to collection C and s_2 to collection A, C should then contain both s_1 and s_2.)
4. Enable drag and drop for moving songs or collections into the current playlist, or perhaps even for adding songs to collections

2.1.4 Nice to Have Features

1. Advanced search strings (ie, `"artist:Cake title:*love*"`)
2. Perform Medialib[2] queries through search bar
3. Browse songs that are not in any hand-picked collection (to find songs that haven't been categoried)
4. Recognize when new songs have been added to the library (watch for new files)

2.1.5 Future Plans

1. Use sound-pattern recognition to look up a user's collection online and correct eronious ID3 tag data (perhaps using MusicBrainz[3])
2. Share your collections with other users so they can categorize their music the same way you did without organizing their library by hand
3. Search songs by lyrics
4. Import music categories from tagging services such as Last.fm to automate the process of building collections.
5. Maintain data about frequency and tempo to find similar songs by sound quality (This could be further specialized to searching for songs that begin similarly to how another song ends, etc.)
6. Port to various operating systems

[2]The "Medialib" is the conceptual media library upon which all of XMMS2 is based. It is used to store song metadata and perform music library queries. It is implemented using an SQLite database.

[3]http://musicbrainz.org/

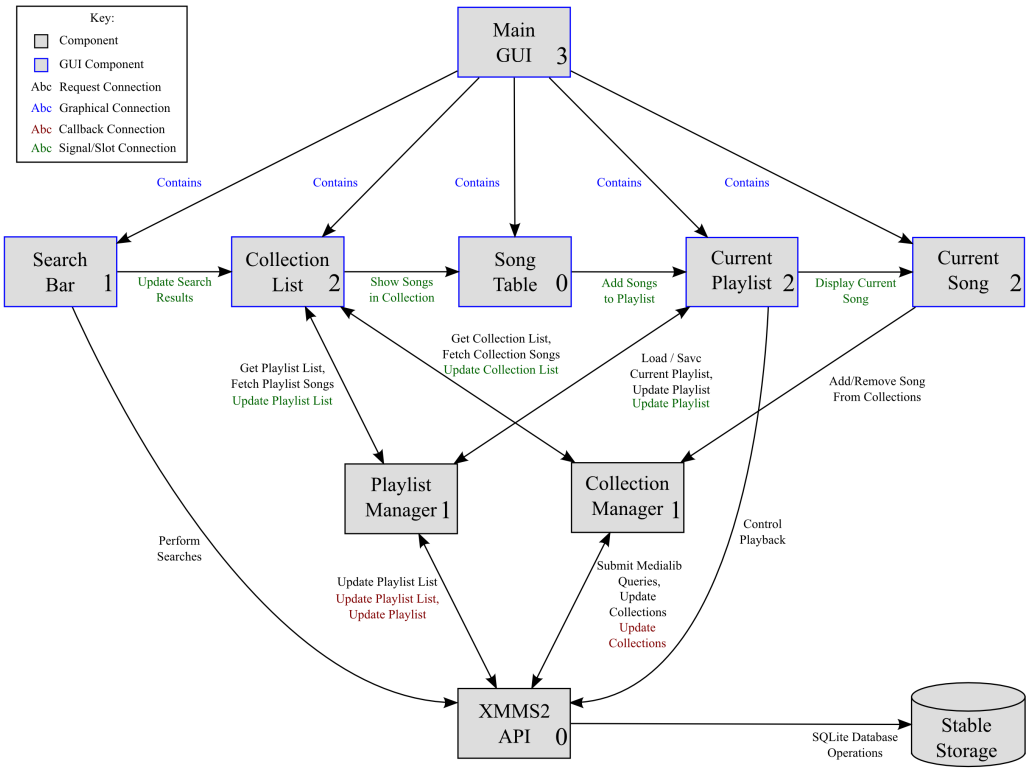
2.2 Other Requirements

These aren't exactly features, but they would be expected by potential users:

- The interface must be quick to load, responsive, and intuitive. This could be verified with user tests, asking them to rate the interface in different categories.
- Data about collections and playlists must be persistant and safe. This should be taken care of by the XMMS2 Medialib system.
- The program must not damage, rearrange, or otherwise interfere with the user's music files (with the possible exception of when authorized by the user).
- Searches must return results quickly, even with a very large music database. Not all results have to come in at once, but waiting more than 10 seconds with no results is unacceptable.
- Fetching all the songs from a collection must be *very* fast (< 5 sec), even if that collection is large or if it is a "special" collection.
- The program must be easy to build and install on any standard Linux system. The source code must be freely available.

3 System Components

3.1 Interactions Diagram



3.2 Explanation

One of the advantages of using the QT library in developing this project is having access to the signal/slot system that QT has developed. This callback like system allows for a class to specify what "signals" it can broadcast and what "slots" it has available to receive signals. When a signal is connected to a slot, then whenever that signal is emitted, the connected slot procedure will be called with the appropriate arguments, like a callback.

The nice thing about this system is signals and slots are part of a class's interface. They are used to connect different classes together, but one class need not concern itself as to which other classes might listen to its signals, or broadcast to its slots. In this system, the main GUI takes advantage of this. It takes on the job of arranging the various other GUI components and it connects their signals and slots together. This way, although data flows from one GUI component to another, none of the GUI components needs to know about the others. The main GUI class is the one that knows all the components and glues them together.

The levelization of the component diagram takes the special qualities of callback functions and signals and slots into account. Only if there is a connection between two components *that would imply an* `include` *relationship* will the level of a component be higher than the one it is connected to.

3.3 Component Descriptions

3.3.1 Main GUI

This will be the actual window that will contain the various GUI components and link them together. The interface will be developed in QT, so connections between GUI components and most callback connections will be implemented using the QT signals and slots paradigm. This way, the Main GUI will be the only component that needs to know about the various GUI components and how they interconnect (The Mediator Pattern).

3.3.2 Search Bar

This component will be used for performing searches on the Medialib based on ID3 tag inoformation. Advanced search patterns or full Medilib database queries will also be accepted here. When a search is performed, the **Collection List** will be notified and it will load the "Search Results" collection.

3.3.3 Collection List

This component will display a list of the user's collections, including hand-picked collections, metadata collections, search results, and playlist contents. It will allow the user to select a single collection or somehow designate unions or intersections of collections (See the discussion of GUI features below). When a selection is chosen, the **Song Table** will be updated to show the contents of the current selection. The **Collection List** will also communicate with the **Playlist Manager** and the **Collection Manager** so that it can fetch data about the available playlists and collections, and so it can be notified if any of the playlists or collections changes.

3.3.4 Song Table

This component will display a list of the songs in the current collection, displaying all relevant metadata. This listing can be sorted by any of the metadata fields, but not edited manually. It is used to view collections and as an intermediary between the **Collection List** and the **Current Playlist**.

3.3.5 Current Playlist

This component displays the songs that have been played, the song that is playing currently, and the songs that will play after the current one. It is responsible for controlling song playback and will need to synchronize itself with XMMS2. Since the concept of a current playlist is built into XMMS2, most of its work will merely be delegated to the **XMMS2 API**

through the **Playlist Manager**. Also, the **Current Playlist** is responsible for updating the **Current Song** component when a new song is selected.

3.3.6 Current Song

This component displays detailed information about the song the user has selected in the **Current Playlist** component and allows the user to change which collections that song is associated with. It will have to communicate with the **Collection Manager** in order to do this.

3.3.7 Playlist Manager

This component serves as an intermediary between the **XMMS2 API** and the rest of the program. It allows for logical management of playlists at a level above XMMS2, but uses XMMS2 to store and retrieve playlists via the Medialib.

3.3.8 Collection Manager

This component serves as an intermediary between the **XMMS2 API** and the rest of the program. It will handle organizing songs into collections in a unified way, and will use song properties in the Medialib to store this data and retrieve it as needed.

3.3.9 XMMS2 API

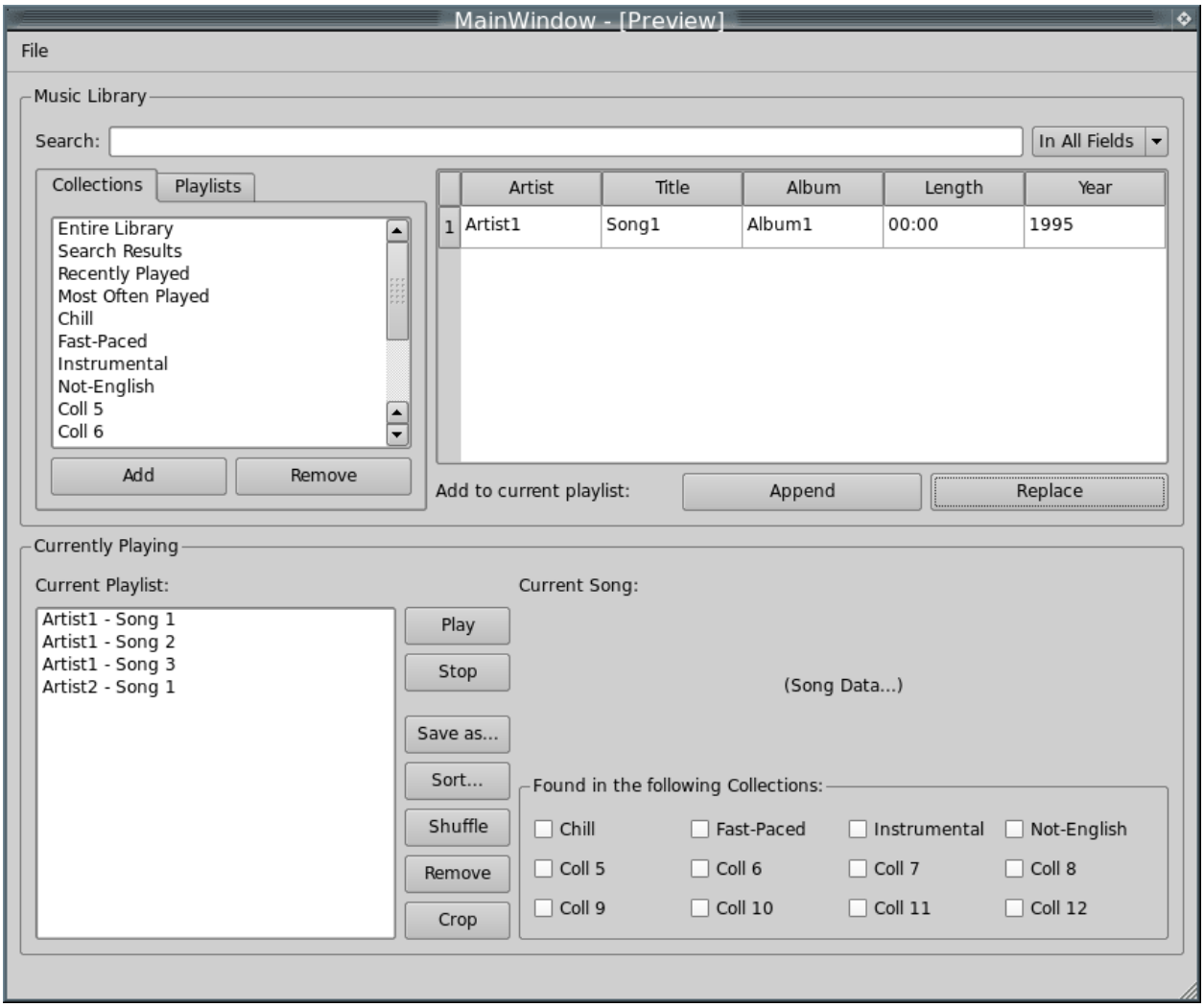
This is the C++ API for XMMS2. It handles all audio decoding and playback features as well as interactions with the Medialib.

3.3.10 Stable Storage

This is where the song metadata, collection information, and playlist information is all stored. This is implemented through the SQLite Database which holds the XMMS2 Medialib.

4 User Interface

4.1 GUI Mockup



4.2 Prominent Features

4.2.1 The Search Bar

At the top of the window will be a text entry field with a combo box designed to fascilitate searching. The user can type a search term into the text box, and use the combo box to decide which fields to search in. This combo box will contain options like “In All Fields”, “In Artist”, “In Title”, “In Lyrics”, etc. Actual database queries could also be performed here, if desired, and when a search is entered the “Search Results” collection will be selected and displayed.

4.2.2 The Collection List

Near the upper left corner of the window is a box with two tabs, one for collections and one for playlists. This is the main way the user will browse through his library. Typically, he will look through one collection at a time (including special collections such as "Entire Library"), but eventually the functionality of selecting multiple collections could be added here. For instance, each item in the collection list could have a "+" button and a "-" button to specifically include or exclude a given collection. The second tab here will list playlists *as if they were collections*. Selecting a given playlist means, *"show me the collection of songs contained in this playlist"* and playlists could be combined using set operations like any other collections.

4.2.3 The Song Table

Near the upper right hand corner of the window is a table containing all the songs in the currently selected collection. These songs can be sorted by field heading, but not manually removed or edited. This is where the user would browse through the songs in a collection, see search results, etc., and not where the user would build a playlist.

4.2.4 The Current Playlist

In the lower left hand corner of the window is the current playlist. This is just used to keep track of what songs have been played recently and what songs will be played next. The user can add songs to the current playlist from the Song Table, and he can rearrange the order of songs appearing in the current playlist.

4.2.5 Current Song

In the lower right hand corner of the window is a listing for the current song, which will include all ID3 tag data, any other desired song meta data, as well as which hand-picked collections the song currently belongs to. The user can change which collections the song belongs to using the checkboxes associated with each collection. If there are more collections than will fit on the screen, the list of collections should be scrollable with the *most common* collections at the top. The current song would be whichever song is selected from the current playlist, not which song is playing. This way, the user can look at details for different songs without interrupting whatever happens to be playing.

4.3 Discussion

This GUI is not completely set in stone, it does not yet accommodate 100% of the likely features, and it will likely need to be revised. Here are some issues regarding the GUI which are still outstanding or should be considered:

- Perhaps the playlists should not be browsable as collections. Perhaps they should be, but shouldn't get their own special tab. It may be awkward to allow the user to select collections to include from multiple tabs at once, since not all the selected collections would be visible at once.

- Currently, to load a playlist, the user would have to switch the playlists tab, find the playlist he wants, and add the songs from that playlist to the current playlist. Not only is this a hastle, but it also technically does not preserve the playlist's order. Perhaps another method should be given (ie, another button to go with the current playlist).
- The number of buttons for operating on the current playlist is starting to get out of hand, and I could easily see more needing to be added. Is there some other convenient way to organize this? Perhaps menus? If so, which features should be immediately accessible (buttons) and which should be accesible only indirectly (through a menu, etc.).
- Perhaps more controls for the collection list should be added. Things like select all, deselect all, invert selection, etc.
- Drag and Drop seems very natural for this interface and would be quite useful. This would perhaps be very complicated to add, though, and it's not entirely clear what should be draggable where.

To address some of these issues, it would be good to brainstorm with the entire group and perhaps run some usability studies with early demos of the program.

5 Risks, Challenges, and Dependencies

5.1 The Learning Curve

We will be using several external libraries for this project, so most if not all of the project team will need to be very familiar with these libraries' APIs. Also, the XMMS2 Medialib is based on an SQLite database. Some of the project team will not only have to get familiar with SQLite but also with the internal structure of the Medialib in order to perform the necessary queries.

5.2 Required Installation

The development environments will have to be configured to suit this project.

- SQLite must be installed as it is a requirement of XMMS2.
- The XMMS2 server must be installed, and a running instance must be available for development.
- The XMMS2 client development libraries must be installed.
- The QT development libraries must be installed.
- Each tester will need access to a large music database.

5.3 External Dependencies: XMMS2

- XMMS2 is extremely powerful and versatile. It will make most of the tricky operations such as playing songs and performing database queries trivial, as all of the hard stuff has been worked out already. It should not limit us much in what we can produce.
- XMMS2 is still actively under development. It is far from bug free, and the functional features change slightly with every new build. The basic functionality does work currently and the API should be set in stone. However, it's dangerous to rely on a work in progress.
- XMMS2 was designed with a server/client model in mind. It has a relatively complicated architecture, and it is NOT safe to assume there will only be one client running at once. We will have to integrate well with the XMMS2 API, make sure we meet all of its requirements, and update the GUI to show external changes to the Medialib.

5.4 External Dependencies: QT

- QT is easy to use, very well documented, and prides itself in being quick to develop with. It provides a robust callback-like communication system called signals and slots which is perfect for dividing a project into several independent but connectable parts. It will be a good toolkit to use for our GUI.
- QT bindings for XMMS2 are planned and in development, but are *not* yet ready for use. This means we will have to interface through the standard C++ XMMS2 API. If we want to use signals and slots for XMMS2 generated callbacks, we'll have to do it on our own.

6 Plan of Action

6.1 Division of Labor

- Project Manager - Organizes meetings, oversees entire project, sets goals and timelines. (Nate)
- Administrator - Organizes internal file organization, build tools, and version control system. Provides support for the other team members in using the system he sets up. (Lincoln)
- Test Czar - Writes system integration tests, periodically checks the full stable build for obvious problems, makes sure people are keeping unit tests up to date, works with the administrator to automate testing processes. (Josh)
- XMMS Expert - Knows every in and out of the XMMS2 API, has studied how other clients interact with it, and can help the other team members make use of it. Will research problems, find example code, and otherwise support the other developers. (Sean)

- QT Expert - Much like the XMMS2 Expert, but for QT. (Nate)
- Distributor - Prepare the project for being released in the open source community. This involves building a web page, maintaining up-to-date usage documentation (man page, web-based docs), creating source and binary packages which are easy to install, etc. (Sean)
- Components:
  - Main GUI (Tara)
  - Search Bar (Kevin)
  - Collection List (Owen)
  - Song Table (Dominic)
  - Current Playlist (Colin)
  - Current Song (Nate)
  - Playlist Manager (Owen)
  - Collection Manager (Brendan)

6.2 Testing

Each developer working on a component will be responsible for writing unit tests for that component. The Test Czar will be responsible for verifying that people are writing decent unit tests, keeping them up to date.

Once the project has reached a stable point, no new code should be submitted without automatically running all applicable unit tests and integration tests. The Test Czar should see that this gets organized and work with the Administrator to ensure that no new code can be committed if it would break the stable build.

6.3 File Organization

I imagine a system where each main component is in its own subdirectory, along with notes on that component, a todo list, and all appropriate unit tests. For GUI components, I would also like to see a stand-alone component demo program that serves only to model that particular component and very vaguely mock up its interactions with the rest of the program.

There would also be separate top-level directories for object files, QT metadata files, documentation, and the like. In the main directory would be files relating to the entire project, such as the primary makefile, the current stable binary, a todo list, etc.

6.4 Build Tools

6.4.1 Building

The QT macros require some preprocessing and generate some special source files to work their magic. Typically, a QT project is managed using a Makefile which is automatically

generated using the qmake tool. Since this is the standard pattern, it has been streamlined nicely and many examples of projects organized this way, so this is how I would propose this project be organized. We would use Makefiles coordinated with qmake for building the entire program, running unit tests, and creating GUI component demos.

6.4.2 Version Control

There are many good solutions for versioning software available. The two most reasonable options are subversion and git. Subversion is very straight forward, well known, and generally understood by most of the students taking this class. Git is the standard versioning software used for the Linux Kernel and for the XMMS2 Project. Git would be a good option just because that is how XMMS2 and most of the current clients are managed. However, there is no reason we could not use subversion instead just because that is what we're most comfortable with.

6.4.3 GUI Building

QT comes with a nice GUI-building application that makes building simple user interfaces easy, using drag and drop. The various GUI components, the Main GUI, and the GUI demos could all be developed using this tool, or they could be written by hand. It would probably make sense to either use the GUI builder for all of the components or none of them for the sake of parallelism, since the process for making a GUI class completely by hand is rather different from that of using the GUI builder.

6.4.4 Documentation

For the sake of the development team, it would be best to keep Doxygen documentation up to date for the entire project. This would not necessarily be helpful to users, but it would be invaluable for developers, especially in working out the details of interactions between components.

6.5 Documentation

For this project, there will need to be at least four distinct types of documentation:

- Linux Man Page - This is expected with any Linux-based application, and there is a very specific format which it should have.
- Doxygen Source Documentation - Not useful for users, but important for developers (both current and future!)
- User Manual - A simple HTML or PDF manual explaining the basics of how to install and use the program.
- Website - Abstract and useful information about what the project is, what it does, how it works, and how to start using it.

The Doxygen documentation will be kept up by the individual programmers as they code, but the other documentation will be the responsibility of the Destributor.