NTNU

Norwegian University of
Science and Technology

A control toolbox for measuring audiovisual quality of experience

Stian Bækkevold

Master of Science in Electronics
Submission date: June 2009
Supervisor: Peter Svensson, IET
Co-supervisor: Ulrich Reiter, Q2S

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

Perceived audiovisual quality is one of the key parameters for the acceptance of multimedia content. User satisfaction is mandatory for new services and schemes to be successful in the field of multimedia presentations. The precise measurement of subjectively perceived quality represents a major problem in the field of audiovisual quality assessment. NTNU will contribute to the field by setting up a test laboratory for the presentation of audiovisual content.

For this, a software-based control toolbox needs to be developed that is able to remote control a state-of-the-art High Definition video clip recorder and a multi-track audio recorder while keeping the audio and video in sync. Typical tasks for the toolbox include the generation and verification of control sequences and status messages, and the randomization of presentation order of audiovisual clips. A JAVA SDK is available for the video clip recorder.

This project will be done in cooperation with Q2S, with supervisor Ulrich Reiter.

Assignment given: 30. January 2009
Supervisor: Peter Svensson, IET

Abstract

Q2S is an organization dedicated to measure perceived quality of multimedia content. In order make such measurements, subjective assessments is held where a test subject gives rating based on the perceived, subjective quality of the presented multimedia content. Subjective quality assessments are important in order to achieve a high rate of user satisfaction when viewing multimedia presentations. Human perception of quality, if quantified, can be used to adjust presented media to maximize the user experience, or even improve compression techniques with respect to human perception.

In this thesis, software for setting up subjective assessments using a state-of-the-art video clip recorder has been developed. The software has been custom made to ensure compatibility with the hardware Q2S has available. Development has been done in Java. To let the test subject give feedback about the presented material, a MIDI device is available. SALT, an application used to log MIDI messages, has been integrated in the software to log user activiy.

This report will outline the main structure of the software that has been developed during the thesis. The important elements of the software structure will be explained in detail. The tools that have been used will be discussed, focusing on the parts that have been used in the thesis. Problems with both hardware and software will be documented, as well as workarounds and limitations for the software developed.

Preface

This thesis is the result of the work completed in a masters thesis, spring 2009. The master thesis has been completed at the Department of Electronics and Telecommunications at NTNU, Trondheim and in cooperation with Q2S.

The goal of the thesis has been to develop software for controlling audio visual experiments for measuring subjective quality of the presented material. Two different applications has been developed to achieve this, with the help of an SDK for controlling a state-of-the-art High Definition video clip recorder.

My biggest thanks goes to Ulrich Reiter at Q2S for invaluable help, guidance and for providing numerous ideas and definitions. I would also like to thank Peter Svensson who got me started with this thesis, as well as fellow students and friends for help with small, everyday problems.

NTNU, Trondheim June 14th 2009
Stian Bækkevold

Contents

List of Figures

List of Tables

Nomenclature

API	Application Programming Interface
AV	Audio Video
DIN	Deutsches Institut fur Normung
DNS	Domain Name System
FPS	Frames per second
GUI	Graphical user interface
HD	High Definition
ID	Identification Document
IDE	Integrated Development
IP	Internet Protocol
LTC	Linear Time Code
MIDI	Musical Instrument Digital Interface
MTC	MIDI Time Code
RAID	Redundant Array of Independent Disks
SALT	Subjective Assesment Logging Tool
SDK	Software Development Kit
SMPTE	Society of Motion Picture and Television Engineers
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
USB	Universal Serial Bus
WYSIWYG	What You See Is What You Get
XML	eXtensible Markup Language

Chapter 1

Introduction

Digital multimedia is a relatively new field. Digital representation of multimedia is an effective form of storing multimedia content with todays computers. Multimedia content has become an important part in peoples lives with the introduction of video services over the world wide web. User satisfaction for these services is a very important criteria for success, and is often highly dependant on the subjectively perceived quality of the content presented.

Measuring subjectively perceived quality and modifying the multimedia content to maximize user satisfaction can therefore be beneficial. This is done with subjective assessments, which is a part of the research area for Q2S. To measure perceived quality, experiments must be set up where the user can, unhindered and without loss of focus, rate the subjective quality of the multimedia presented. To set up such an experiment, software is required to log and control the experiment.

The goal of this thesis is to develop this kind of software, based on the existing hardware at Q2S. This software will be focusing on control of a state-of-the-art clip recorder and logging of message from the person rating the presented material. Logging of messages will be achieved by integrating already existing software into the applications developed. This report will focus on the development of this software and the tools used, as well as how the software can be used to perform subjective assessments of multimedia content using Q2S available hardware.

The report is divided into four main parts. The first part will explain the background of the software, including specifications, problems and limitations of the software. The second part will explain the tools used to create the software and the software affiliated with the hardware used in experiments. This part will also describe SALT which has been integrated into the control application. The third part will focus on the development stage and the problems that have arisen during development. The fourth and last part will have a summary and a discussion of work that can be done to add practical features and fix problems with the software.

Chapter 2

Background

The purpose of the tests is to measure subjective quality of multimedia (AV (Audio Video)) clips. In order to hold experiments, some equipment is naturally necessary. The test itself consists of the test subject and an operator, in addition to the necessary equipment. The operator will set up the equipment and prepare it for use before the test starts. In some cases the test subject will notify that he's prepare to start, which is done with a button on the feedback device. The test subject is considered to have no knowledge of the equipment used in the experiments, therefore ease of use is a factor. The feedback device will be explained in more detail in section 3.4.

The operator will ready the experiment by selecting the clips that are to be rated by the test subject. The list of eligible clips to play will be generated by the clip recorder so the operator can choose the appropriate clips. The operator will also choose on which MIDI (Musical Instrument Digital Interface) channels to communicate with the feedback device. Before the experiment is ready, it needs to be validated. To get the most out of the experiment, the order of clips should be fairly random to ensure that impressions from one clip does not propagate to the next. This function should also be accessible from the software controlled by the operator.

2.1 Setup

The full system overview can be seen in figure 2.1 below. This also shows an audio computer which may be used for to process audio information. This is the control application the operator will use to set up the experiment. In figure 2.1 the communication interfaces are also marked. There are three interfaces used, MTC (MIDI Time Code) , IP (Internet Protocol) and the already mentioned MIDI. These interfaces are explained in more detail below.

2.1.1 MIDI

MIDI is an interface for sending system messages and event messages between MIDI devices. MIDI messages usually specify a tempo and intensity of e.g. a piano event. The MIDI interface have 128 available channels for use through a single connection. The physical interface is a five point DIN (Deutsches Institut fur Normung) connector. Since normal computers don't usually have DIN connections, a MIDI-to-USB-converter has been used in the setup. This is not marked in figure 2.1 as it is only a converter. MIDI is used to send control messages between the feedback device and the control suite, therefore the messages will have another usage than what is usually used in MIDI communication. The

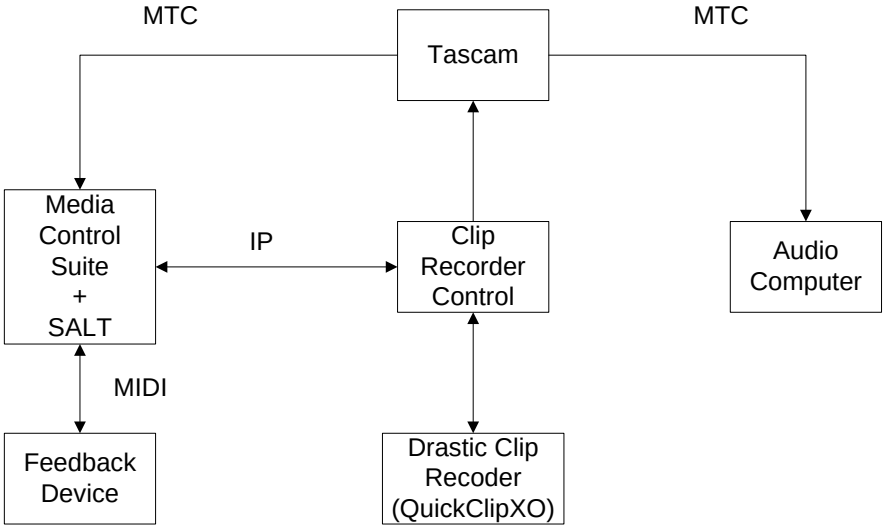

Figure 2.1: System overview

feedback device and what kind of MIDI messages it sends will be described in detail in section 3.4.

2.1.2 MTC

MIDI Time Code is used for synchronization purposes. All it contains is time information, which the receiver can use to synchronize its clock. The clip recorder is the time code master and generate time code as it plays clips. For every quarter frame, time code information is sent. However, it takes eight quarter frames to send a full time code, so for every second frame, a full time code will be sent. The quarter frame information is sent as one byte. The first hex represents the nibble, that is which of the eight quarter frames is sent. The second hex is the value. Since the byte is sent eight times, the time code is divided into eight parts. The first two parts gives one byte to represent the number of frames. The second, third and fourth bytes gives seconds, minutes and hours. See table 2.1 for an overview. The status byte for MIDI Time Code is 0xF1. The status byte simply points to the kind of message that's being sent, see **??**. Time code is used in the control suite to know when clips are finished playing. In string format the time code will be written as "hh:mm:ss:ff", where hh denotes hours, mm denotes minutes, ss denotes seconds and ff denotes the frame count for the each second.

Table 2.1: MTC specification

Nibble	**Value**
0	First hex of frames
1	Second hex of frames
2	First hex of seconds
3	Second hex of seconds
4	First hex of minutes
5	Second hex of minutes
6	First hex of hours
7	Second hex of hours and SMPTE type

2.1.3 IP

To communicate between computers, the widely used Internet Protocol is used. It is available through socket programming in Java. The control suite and the clip recorder will only need to send simple messages to each other, and not a whole lot of data. Therefore there are no speed requirements to speak of, and so there's no need for UDP (User Datagram Protocol) , which is optimized for transport efficiency. TCP (Transmission Control Protocol) is optimized for reliability, so the need for error checking can be kept at a minimum. TCP is therefore the chosen protocol for communicating between the clip recorder and the control suite.

2.2 Definitions

A test consists of a technical specification, and a number of clips. The technical specification is called a design. The list of clips are called a playlist. The playlist has a certain organization as well and consists of trials and items. For each new test subject, name and age is recorded, and possibly more details. The test subject details are called a session and is tied together with the logged data from the test subject. These concepts will be described in more detail below.

There are two different types of tests; multi-stimulus and single-stimulus. In multi-stimulus tests, clips are to be rated against each other, so that the score each clip gets is relative to the other clips. In single-stimulus tests, the clips are played consecutively, although the test subject may still have control of the clips by repeating clips at will.

2.2.1 Design

A design is the technical specifications of an experiment. When creating a new design, MIDI channel numbers are specified for all buttons and faders on the feedback device. The number of faders and its corresponding buttons is also specified. A playlist is attached to the design as experiments are often given for multiple subjects. The design also sets what kind of logging the test should have. SALT (Subjective Assesment Logging Tool) supports three rating modes which are to be used; non-continuous rating, continuous fader rating and continuous button rating.

- The non-continuous rating logs the rating a clip have at the end of the clip. In non-continuous rating the user is free to adjust the fader to set the desired value as long as the clip is playing.
- In continuous fader rating, all fader actions are logged. This might be useful if the clip changes quality during presentation.
- In continuous button rating, a clip is rated by either zero or one. This might be useful where clips are to be rated against each other and the best clip gets a positive rating.

In addition to choice of rating and MIDI channel numbers, a design also holds information about which MIDI devices that are used. MIDI devices are generally split into input and output devices. The control suite have two input devices and one output device. The output device is used to give the user feedback during a presentation. The input devices are used for receiving time code from the Tascam RS422-Midi converter, and for receiving events from the feedback device. In a design, there is also information about where sessions are stored. If extra information about the test subjects are to be included, this is also tied

to the design. A design should be stored as an XML (eXtensible Markup Language) file, with all relevant information including a playlist.

2.2.2 Session

A session stores information about the test subject and the information the test subject gives during the test. A session is stored as an XML file at the location specified in a design. Therefore, a session should not be created without first creating a design.

2.2.3 Playlist

A playlist is simply the list of clips that are to be presented during the test. It has a hierarchy shown in figure 2.2. A playlist is tied to a design and is stored in a design. A playlist contains any number of trials.

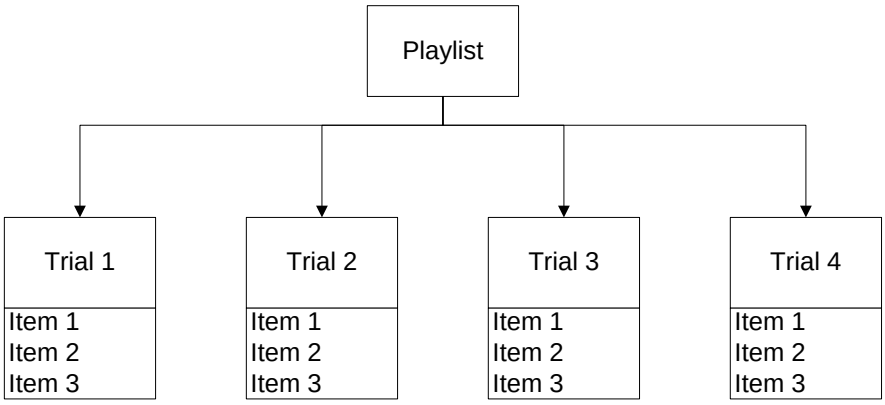

Figure 2.2: Playlist hierachy

2.2.4 Trial

A trial consists of a fixed number of items. It is a placeholder for clips that are to be rated as a multi-stimulus test. A trial may still be used for single-stimulus tests though, e.g. where a number of clips are to be rated with different qualities, so that each trial holds a set of clips for each given quality. Trials should also have a name to distinguish trials from each other. The number of items in each trial is fixed so that all trials have the same number of trials, as shown in figure 2.2. This limitation is justified to be able to set up multi-stimulus tests that can have similar trials.

2.2.5 Item

An item represents a clip. It holds more information than just the filepath of the clip though. In addition to file path, an item should include an item name and also tags to attach extra information the clip. The tags should be used, in addition to providing extra information, to classify clips that are naturally correlated. An obvious example is two equal clips of different quality. As the impressions from the first of these two clips may propagate to the next clip, playing these two clips consecutively should not be allowed.

2.3 Specifications

A list of specifications is necessary to get a clear view of what kind of functions the software should have. Below is a list of necessary specifications for the software.

- A simple graphical user interface
- Integration with SALT to log midi events from the test subject
- Correct organization of playlists, consisting of trials and items
- Ability to save a design and load it
- Ability to save a session and load it
- A function to randomize the playlist with validation to avoid similar clips being played consecutively
- A status panel to get an overview of the status of the software
- A MIDI message log
- Ability to select eligible AV clips from the clip recorder
- Ability to tag items with extra information
- User control of playlist to allow for repetitions and custom clip order.

Chapter 3

Tools

This chapter will describe tools used in the thesis. Both hardware and software will be described, in addtition to the way it is used. The basis for connecting the hardware is figure 2.1. Hardware is equipment that Q2S already have aquired, and so the software developed in the thesis will be custom made to work specifically with this hardware.

3.1 Drastic Clip Recorder

The Drastic clip recorder is a computer designed for working with uncompressed HD (High Definition) video. There is custom software for this machine that is accessible through a SDK (Software Development Kit) . Documentation is available in [11]. This software enables the computer to play uncompressed HD video. The usage of this machine is limited to receiving relevant information about available clips and starting and stopping of playback. Also, creating a playlist compatible with the software is important. The software will be further explained in section 3.1.2.

3.1.1 Hardware

The clip recorder is a computer with an extensive RAID (Redundant Array of Independent Disks) network consisting of a large number of high performance hard drives. This enables the computer to access and write a large amount of data quickly, which is necessary to work with uncompressed HD. The computer also have a high performance cpu. The hardware only serves as a basis for the software, and is therefore not too important for this thesis and will not be described further.

3.1.2 Software

The software used on the clip recorder is called QuickClipXO. As mentioned above, the software is used to play uncompressed HD video. It can also record uncompressed HD video due to its hardware capabilities. QuickClipXO is part of a software package that is installed on the clip recorder. The interface includes a playlist, basic control buttons like 'Play', 'Pause', 'Record', 'Stop' etc, buttons for changing playback modes and controls for time code as well. There are also some configurations that can be done in the application. A screenshot of the software is shown in figure 3.1.

The playlist has two main modes, clip and conform mode. The clip mode is used when the user wants to play single clips consecutively. The single clip is the main focus, and the playlist is simply a list of clips. Each clip restarts the time code. This is very useful

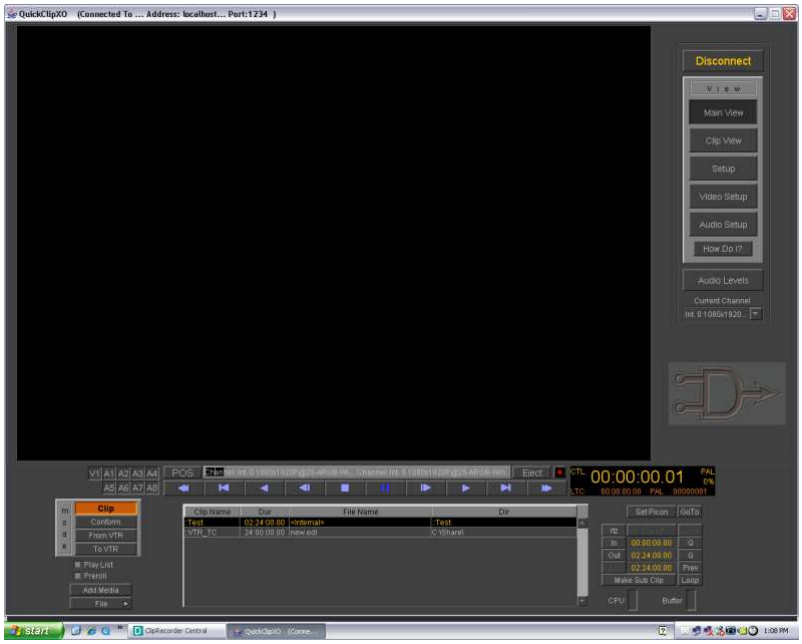

Figure 3.1: Quick Clip XO screenshot

if requirements for complexity is not high.

The other main mode is called conform mode. This mode has the advantage of a single timeline for all the clips, and therefore a single time code. A single time code enables the user to add clips after each other with delay, or even overlapping. Clips are added to the timeline at the users choice. The great advantage of this is that the time code gives information about both the progress for each clip and the progress on the whole timeline. That way time code alone is enough information to know what clip is playing and how long the clip has been played. Because of this advantage, conform mode is the preferred mode of operation for the clip recorder in this thesis.

Quick Clip XO is availabe programmatically by a SDK. This SDK carries the basic functionalities required for the thesis. Since the specifications only asks for playback, the needed functions will be to play a selected clip, pause playback, stop playback and playlist control. Playlist control includes adding items to the playlist and removing them, and adding a conform mode playlist. Connecting the application to Quick Clip XO is of course also necessary. There is also need for knowing the length of each clip to get total control over progress. All these functionalities are available in the SDK, however, playlist control is marked with 'Internal Channels Only' and 'Do not use yet'. Playlist control is however implemented, and as stated not ready for use yet. Some status operations might also be useful, like getting the current video format. Functions that are relevant for this thesis is listed below. The documentation is available online at [11].

- PlayClip - Plays a clip from the current position
- PlayClipFromTo - Plays a clip from the frame A to frame B
- Pause - Pausing playback
- Stop - Stops playback
- Connect - Connects software to QuickClipXO

- Disconnect - Disconnects software from QuickClipXO
- GetClipInfo - Gets information about a specified clip
- Insert - Inserts a clip into the playlist
- Blank - Removes a clip from the playlist
- GetTCTcType - Gets the current type of video, typically this determines the FPS (Frames Per Second)
- TCToFrame - Utility function; Converts a time code in string format to a frame count given the video type
- TCToString - Utility function; Converts a time code in frame count format to a string given the video type

3.2 P2MMC Tascam RS422-Midi Interface

P2MMC Tascam RS422-Midi Machine Control Interface is a hardware device to convert RS422 signals into MIDI signals. This device is externally powered, and takes a RS422 signal as its input, along with a video synchronization signal if necessary. It then outputs to MIDI or as LTC (Linear time code). It can also use MIDI as input, or even LTC, and then output via the RS422 interface. This device is used to give MIDI time code to both the control suite and the audio computer. A connection overview for the Tascam RS422-Midi interface is available at [4].

3.3 MIDI to USB converter

To use MIDI on a standard computer without MIDI connections, there is need for a MIDI to USB (Universal Serial Bus) converter. There is actually need for two converters as there are two inputs required for the thesis. For this thesis, Q2S have supplied two differenet converters. The first one is an E-MU Xmidi 2x2. The second is an Edirol UA-25. Both have drivers ready for download online. The E-MU Xmidi 2x2 can send and receive MIDI from two seperate devices, and is therefore perfect for converting time code from the Tascam in order to send time code both to the audio computer and the control suite.

3.4 Feedback device

The feedback device used in tests are custom made by Q2S. This device has not been available during the thesis work. Instead, another MIDI device was used. This was a Behringer BCN44. It allows for changing and specifying the MIDI channel number for each of its controls. The Behringer can also adjust the range of the slider, which is by default 0 to 127. The Behringer device has only been used for testing purposes, but the general make up of this device resembles that of planned feedback device. Below is an illustration of the planned feedback device and an image of the Behringer device. As the illustration shows, the feedback device has four different sliders. These will be used to grade items in a trial.

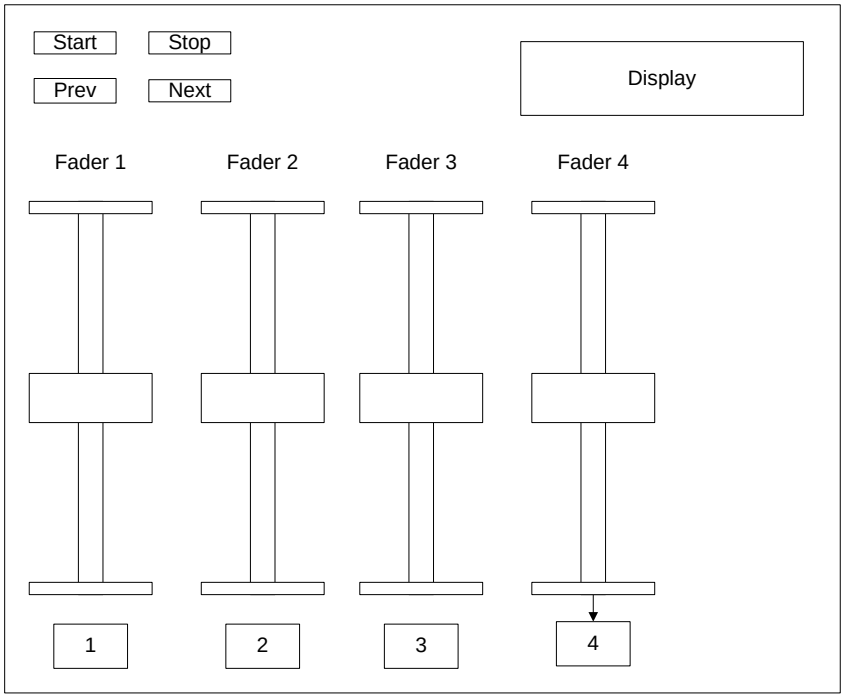

Figure 3.2: Feedback device illustration

Figure 3.3: Behringer BCN44

3.5 SALT

SALT is the Subjective Assessment Logging Tool. It is an application created to log MIDI messages during subjective assessments. SALT already have a structure similar to the structure outlined in the specifications in section 2.3, and so an integration with SALT is preferable. SALT has its own GUI (graphical user interface), shown with a screenshot in figure 3.4. SALT is used by first creating a proper design, and then creating a session. After the session is created, recording of MIDI messages are started immediately. The session is done when a stop button is pressed in the GUI. The source code for SALT was made available to view and change as needed for this thesis. This section will describe the structure of the SALT source code. The code is split in two natural main parts, GUI related classes and application logic.

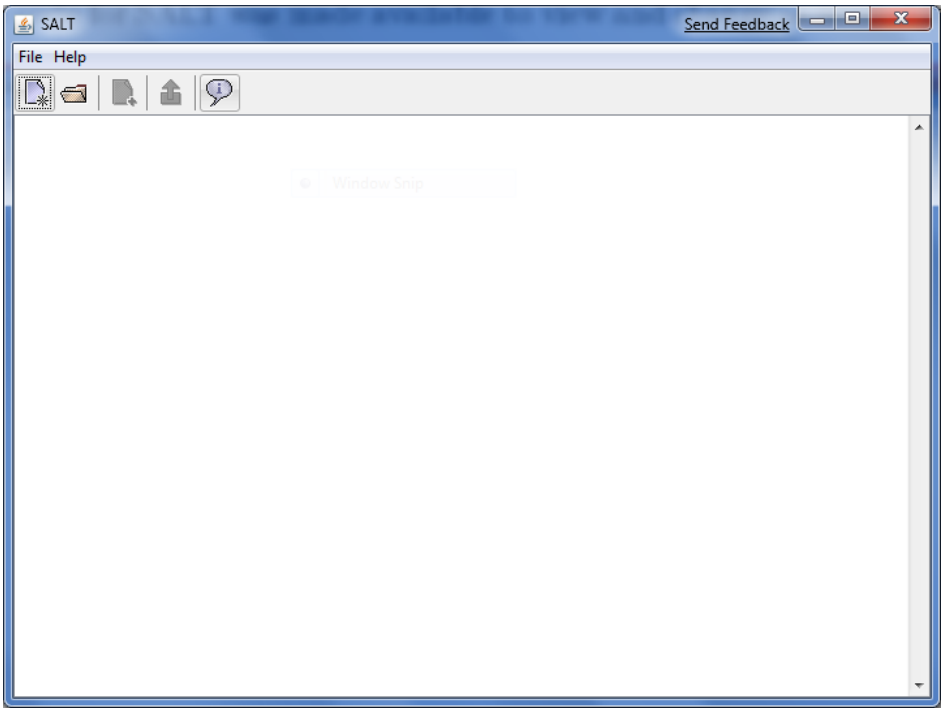

Figure 3.4: SALT screenshot

3.5.1 GUI classes

The GUI classes all have a common ancestor, called Graphics. Graphics controls the layout of each of its instances. The layout used is the GridBagLayout, which is a part of Swing [8]. Two classes inherits from the Graphics class. One is a Layout class, the other is a Dialog class. The Layout class is a class that organizes the layout of the main window. It specifically deals with the toolbar and the menu line, see figure 3.4. The Dialog class is a parent class for all dialogs used in SALT. All dialogs have their own class, governing their appearance and functionality. A GUI class holds the full GUI, including a Layout instance. The GUI class inherits from JFrame and is the main window of SALT.

3.5.2 Application classes

The application specific classes are classes that handle logic and functionality in SALT. This section will outline the main concepts in SALT. The most important classes are ex-

plained in detail, while less important classes usually have a function related to the more important ones.

Salt: The Salt class is the main class of SALT. The main method, the method called when the application starts, is located in this class. This method creates a new instance of Salt, which again creates the GUI. This class also holds a MIDI output and input device, a Design and a Session, a MIDI Receiver (a software implementation of an external MIDI device) and information from the operating system about installed MIDI devices. The MIDI devices are instances of the Java Sound class MidiDevice. This class and how basic MIDI functionality is implemented is explained in section 3.8. The Salt class also implements methods for opening and closing the MidiDevices, as well as outputting a MIDI message. The Design and Session classes are described below. Methods for starting a new session and starting a new trial is included in the Salt class. The MIDI information is a List of MidiDevice.Info, a nested class in MidiDevice. Functionality for retrieving this information is also included in this class. The Salt class handles button clicks in the main window as well.

DumpReceiver: A DumpReceiver is a class implementing the Java Sound interface Receiver. There are two methods required when implementing this interface, a close method and a send method. The close method is obivously called when closing the Receiver. The send method is called internally when the the system receives a MIDI message from a MIDI device. The send method provides access to MIDI messages received in the system. The name might be confusing, but because it is usually called by another MIDI device, not by the programmer, the device calls send. For more information about the Receiver interface, see [6].

When receiving a MIDI message, the DumpReceiver first finds the type of the message. Eligible types of MIDI messages are MetaMessage, ShortMessage or SysexMessage. The DumpReceiver class only handles the ShortMessage and SysexMessage types. These messages are casted into their respective classes and processed in the DumpReceiver. The information from the MIDI device is parsed into information stored into the active session. Since the user might also be able to control the playback of trials and items, control messages are also parsed by the DumpReceiver in the send method.

Design: A Design contains information about the way a subjective assessment is set up. This information consists of four categories. These categories are conveniently reflected in the DesignDialog, where each category has its own tab. The first category is extra information required of the subject, besides name and age.

The second category is general test settings. For different types of tests, additional information might be necessary. This information includes what kind of test it is, and relevant information related to the type of test. For continuous ratings using a fader, a time slice value is necessary to set the sampling rate for the fader. The total time of the file to be played is also necessary for a continuous rating with a fader. For continuous button ratings, the MIDI channel number for the button is necessary. This category also holds information about trails and item. SALT only supports identification and a short description of each trial and item, and this information is stored in the design. SALT also limits the information for each trial by setting the same description for all trials with the same ID (Identification Document, ID is used as a identifier). Therefore it is impossible

to distinguish two items with the same idea, even if they are attached to different trials.

The third category is MIDI channel settings. The MIDI channel settings specify the channel number of every MIDI control, like buttons and faders. How many faders and buttons to be used are also a specified in this category.

The last category stores information about which MIDI devices (recognized by the operating system) to be used. Information about which folder to store sessions in is also stored in this class. The Design class contains a large number of set and get functions, in addition to a clear function used to erase all information stored in a design.

To write a design to disk, a DesignWriter class is used. This is in principle a very simple class, containing a write method. This method takes a design and the requested outputfile as input. The design is written as an XML file, with the structure laid out in figure 3.5. To read a design from the hard drive, a DesignReader class is used. This is also a simple class, containing only a getDesign function that takes the requested file as input.

```
<testdesign>
        <additionalSubjectInfos>
                List of additional subject info
        </additionalSubjectInfos>
        <generalTestSettings>
                List of settings stored in a design like MIDI channel numbers  and other relevant information.
        </generalTestSettings>
        <trials>
                List of trial ids and a short description
        </trials>
        <items>
                List of item ids and a short description
        </items>
        <MIDIInput>Integer value pointing to a value in the list of MIDI  devices installed on the system</MIDIInput>
        <MIDIOutput>Integer value pointing to a value in the list of MIDI  devices installed on the system</MIDIOutput>
        <SessionTargetFolder>Path of folder to store recorded sessions</SessionTargetFolder>
</testdesign>
```

Figure 3.5: Design XML Format

Session: The Session class manages information about a test subject and the rating this subject gives for the relevant assessment. A design might add additional requirements in terms of information about the test subject, but by default the relevant information is only name (split into first name and surname) and age of the test subject. If anything extra information is required, it is stored in an XtraInfo instance. XtraInfo is a simple class with two fields, the info name and info value. Rating information is stored in a list of TrialInfo instances. A TrialInfo is a class that contains the trial ID and a list of ItemInfo instances. An ItemInfo contains information about the item, ID and a value. The ItemInfo class can also hold a list of values and a list of time stamps. XtraInfo, TrialInfo and ItemInfo all contains get and set functions for the relevant fields.

As with designs, a session can be written to the hard drive. This is done with a SessionWriter class. This class is very similar to the DesignWriter, containing the same function, although with a Session as its input. The structure of the XML file is shown in 3.6. There is also a SessionReader class used to read a session from an XML file. This, again, is very similar to the functionality of the DesignReader.

Filefilters: FileFilter is a Java IO class. It is used to filter in only the relevant file end-

```
<testresults>
        <subjectInfo>
                <firstName>First name string</firstName>
                <lastName>Last name string</lastName>
                <age>Age value</age>
                <XtraInfo>List of XtraInfo necessary from design
specifications</XtraInfo>
        </subjectInfo>
      List of items with name, id and given value for the item. Written with
      shown format for each trial.
      <trial name=[trial name] id=[trial id]>
        <item name=[item name] id=[item id]>item value</item>
      </trial>
</testresults>
```

Figure 3.6: Session XML Format

ings. In SALT, FileFilters are used to filter the correct files when selecting files from a file browser. There are three relevant file filters in SALT. They all have their own class, implementing the FileFilter interface. The classes are called FilterDESIGN, FilterLOG and FilterSUM. FilterDESIGN only accepts .design files, which is used when saving a design. FilterLOG is used for .log files, which is the file ending for the session file format. A .sum file is used to combine several .log files, and is filtered by FilterSUM. The file filter only plays a minor role in SALT, but they are useful to filter out irrelevant information for the user.

3.6 Netbeans

Netbeans is a free open source IDE (Integrated Development Enviroment). It can be used to develop applications in Java, C and C++, in addition to several scripting languages. Netbeans runs on several platforms, including the most popular, Windows, Mac OS X and Linux. This thesis will be using Java, and developing on Windows. Developing in Java ensures multi-platform compatibility, which is an advantage. Netbeans documentation is available at [9].

Netbeans has three main focuses. General code development, GUI development and database development. There is no real use for databases in this thesis, so Netbeans is chosen for its excellent GUI builder. Netbeans is also widely used, and has a active community and extensive documentation. Netbeans have several built-in graphical components that can be created using a wizard. The most used are JFrame, JPanel and JDialog. When creating one of these components, a dialog where name and package is specified. After creation, the relevant component shows up. To fill the component with other graphical components like JButtons, JLabels, JLists etc can be added to the component with simple drag-and-drop. In the Netbeans GUI Builder, WYSIWYG (What You See Is What You Get) is a central concept. Most properties of added components can be set with the GUI builder, in addition to custom constructors. The functionality of the Netbeans GUI builder is quite comprehensive. Figure 3.7 shows the GUI builder with an empty JFrame constructed with the GUI builder. There are two modes in the GUI builder, source and design. Source is where source code can be altered by hand, and design is the mode where GUI components can be added, removed or altered. Figure 3.7 shows the design mode as active.

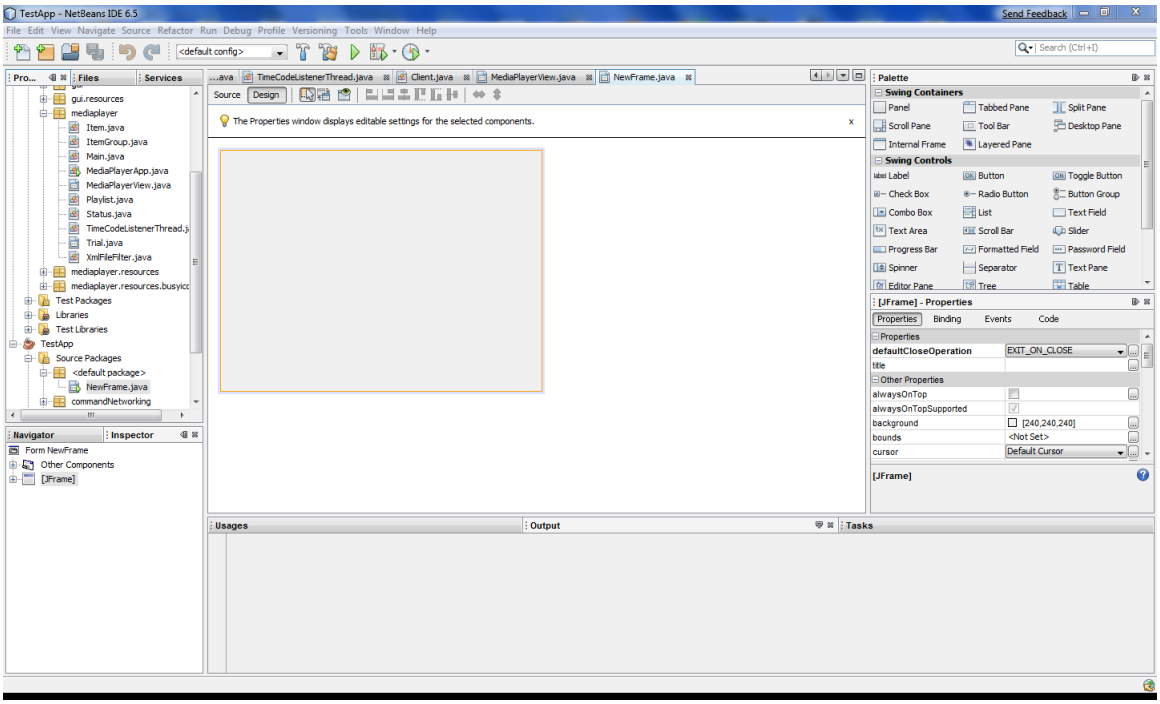

Figure 3.7: Netbeans GUI builder

3.7 Eclipse

Eclipse is another IDE. It is probably the most commonly used IDE for Java. It excels in developing code, and not in a GUI builder as Netbeans. Eclipse do support a GUI builder however, but it is a plug-in and not as simple and not as developed as the GUI builder in Netbeans. Although the two IDE's naturally overlap in many senses, they have different strengths. Netbeans has the best GUI builder, but Eclipse has the best support for efficiently writing code. This thesis uses both IDE's, but for different applications. It's worth noticing that Netbeans have support for Eclipse projects, but not the other way around. Eclipse documentation can be found at [3].

3.8 Java Sound

Java Sound is an API (Application Programming Interface) that provides high level access to low-level operations for audio operations. This includes support for MIDI operations. In Java Sound there is a separate MIDI package. This pacakge contains all relevant functions for operating with MIDI messages. This section will outline use of this package, including a small example.

The most important class in the MIDI package, is MidiDevice. This class is a software implementation of either an external device, like the Behringer BCN44, or a virtual device. An instance of MidiDevice is retrieved by using the static method getMidiDevice in MidiSystem. This method take a MidiDevice.Info as input. Retrieveing MidiDevice.Info is done by simply calling the static method getMidiDeviceInfo in MidiSystem. getMidiDeviceInfo takes no argument and returns a list of MidiDevice.Info instances. This list contains all output and input devices. The MidiDevice class has methods for returning the maximum number of receivers and Transmitters, and this is used to find out whether

the device is an output or input device.

Before the MidiDevice can be used, it needs be opened up by using the open method in MidiDevice. After the device is opened, transmitter and receiver for the device may be set. For input devices, a receiver is created by instantiating a class that implements the Receiver interface. This interface requires a send and close method. The send method gives access to MIDI messages sent from the device. Setting the receiver is done by calling setReceiver on the Transmitter retrieved from the device with the receiver as input. For an output device, the same procedure may be used, however, using the Transmitter interface instead. A small example is included below. It focuses on an input device, and does not cover using an output device. Note that this example do not compile or run by itself, but the code may be used in an existing application framework. Imports are also not covered. All relevant imports are however from javax.sound.midi.

```
//Create Java Sound MidiDevice and connect a Receiver to it
MidiDevice.Info[] infos = MidiSystem.getMidiDeviceInfo();
for(int i=0;i<infos.length;i++) {
    MidiDevice device = MidiSystem.getMidiDevice(infos[i]);
    if(device.getMaxReceivers!=0) {
        //This device is an output device
    }
    if(device.getMaxTransmitters!=0) {
        //This device is an input device

        //Open device
        device.open();

        //Get transmitter from device
        Transmitter trans = device.getTransmitter();

        //Set the transmitters connected receiver
        trans.setReceiver(new CustomReceiver());
    }
}

//A CustomReceiver used to receive MIDI messages
private class CustomReceiver implements Receiver {
    public void send(MidiMessage msg, long lTimeStamp) {
        //Print out msg
        System.out.println(msg);
    }
}
```

Chapter 4

Development

Development has been done exclusively in Java, using both Eclipse and Netbeans as development enviroments. Two applications have been developed, the Media Control Suite and the Clip Recorder Control. In addition to these application, SALT has been integrated into the Media Control Suite to get MIDI functionality. This chapter will explain the structure, functionality and inner workings of the applications developed. The applications are based on the specifications outlined in section 2.3. The flow of information between the applications and hardware is discussed in the next section, 4.1.

The audiovisual experiments are run by the clip recorder, and controlled by the control suite. As figure 2.1 shows, audio should come from another source, an audio computer, however, due to problems with the Tascam RS422-Midi device and lack of proper software for the audio computer, the clip recorder will function as a source for both audio and video. The problems with the Tascam RS422-Midi device will be discussed in 4.6.1. Because of this, the new system overview is simpler. Figure 4.1 shows the new system as it was implemented. Source code will be described with focus on the structural parts of

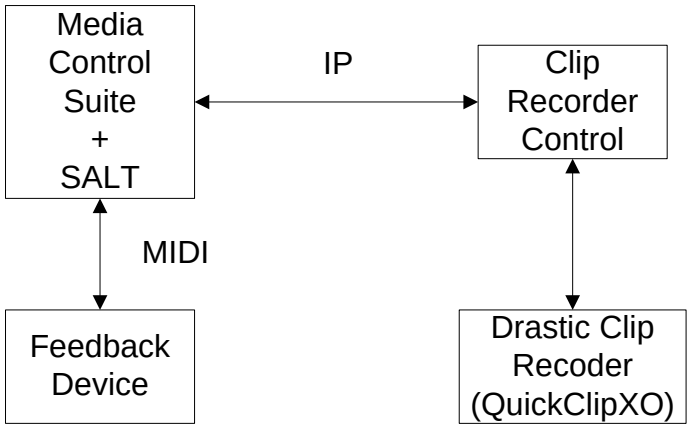

Figure 4.1: Implemented system

the code, including a detailed description of each application and the important properties of the central classes. Details and trivial code will not be discussed, neither will all class properties. The source code is attached for a complete understanding.

4.1 Information flow

The flow of information is varied, using two fundamentally different interfaces. As mention earlier, the Tascam RS422-Midi device was neglected, and so the remaining communication interfaces are IP and MIDI. This section will describe the information flow of the different connections between software and hardware. Communication between the QuickClipXO software and the clip recorder hardware is not described. This information is a part of the package Drastic Technologies deliver and is also proprietary.

4.1.1 Feedback device and control suite - MIDI

Information between the feedback device and the control suite uses the MIDI interface. From the feedback device, this information is limited to information about fader values and button presses. A MIDI message consists of a status byte, and two data bytes.

The status byte is a regular MIDI status byte, although with some extra functionality. The MIDI specification can be found at [1]. SALT only handles the system exclusive messages, which is identified by a status byte value of 0xF0, by throwing them away. They are not a legal message to be logged by SALT. Instead, SALT uses status bytes 0x90 and 0xB0 to distinguish between control buttons and buttons and faders used for evaluation. The control buttons consists solely of the next and previous trial buttons. All other controls uses a status byte of 0xB0. The Behringer BCN44 only outputs with a status byte of 0xB0 however, and so SALT has been modified to use 0xB0 as its status byte for all controls.

The two data bytes are used for identification of the control that is sending the message, and the value it sends. The identification byte can be specified both for the feedback device and the control suite. The feedback device obviously specfies the value in hardware (or rather by changing the firmware), which can be tedious. Therefore it is practical to set the firmware to use a set identifier, and rather change it, if need be, in software using the control suite. The value byte gives the fader value. It is practical for a test subject to set a number between zero and a hundred, simply because those are numbers that a person relates to fairly easily. For that reason, the feedback device should only send out values in that interval. A full byte can give values in the interval [0,255], given its eight bits. The first bit is however used to show that the byte is either a status byte or a data byte, giving an interval of [0,127]. The feedback device should therefore be able to set its range to be a maximum of 100.

The control suite can also send messages to the feedback device. This is done by calling sendShortMessage in SALT. This can be used to, e.g., send messages to the display. This is however not implemented as testing it is not possible without the proposed feedback device. For each new started trial, however, a controller message is sent to the feedback device. This is a part of the original code in SALT. As the sendShortMessage is already implemented, extending the use of communication from the control suite to the feedback device should not be difficult.

4.1.2 Media Control Suite and Clip Recorder Control

Communication between the control suite and the clip recorder control is done over IP. This is a natural choice of communication between software, mostly because of the reliability TCP/IP gives. Sending messages is done by using a library developed for this thesis. This library sends objects over the network, not just low level bytes. The network library is explained in alot more detail in the next section. In addition to pinging each other, as the

network library does by default, there are a few other things the two applications sends to each other. Figure 4.2 shows what types of messages are sent between the two applications. The figure also shows the chronological order of the messages, the first sent messages from the top, the last from the bottom. Messages sent during setup of the experiment and during playback are seperated by a dotted line.

Firstly, the used FPS in QuickClipXO is found using the Drastic SDK. This FPS is sent to Media Control Suite to set the correct FPS for that application. The second package that is being sent is the list of eligible files from the Clip Recorder Control. Only .mov, .tga and .tiff are legal formats, but this is easily changeable by editing the code. Preferably this should be done in a config file. The config file only specifies folders that contain media files that can be played. Clip Recorder Control searches through these folders for eligible files when a connection Media Control Suite has been established. The list is populated with instances of the Clip class. This class also holds information about the length of the clip. The clip also gets named with the filename of the clip. The length of the clip is found by getting the number of frames and dividing by the FPS. This is then converted to time code string format (see section 2.1.2) and stored as a string.

Media Control Suite lets the user choose among the clips received from Clip Recorder Control. The clips chosen is organized by the user, and when the randomizing process starts, a list of trials and items are generated. The randomizing process is described in detail in 4.3.2. Given the length of each clip, a playlist is generated with start and stop points for each selected clip. This information is used to create an .edl file that can used in conjunction with QuickClipXO. This file is simply a playlist of files for QuickClipXO to play, with given clip-start and clip-stop points. This whole list has a common time code that runs through the list and starts and stops clips at the appropriate position. The list of files sent from Media Control Suite is stored in a ListPacket. When Clip Recorder Control receives this packet, it sends a confirmation as a StringPacket. After Media Control Suite has received the confirmation, the rest of the randomizing process will be completed. If no confirmation is received, the randomizing function will assume something went wrong.

During playback, control messages from Media Control Suite are sent. Media Control Suite contains and controls all information about the session, and tells Clip Recorder Control when to start another clip, when to stop and when to pause a clip. To play a clip from and to a given frame, a PlayPacket is used. This contains start and stop time code information, as well as the name of the clip. For stop and pause commands, a StringPacket is used with a string equal to "stop" and "pause". During playback, the current time code in QuickClipXO is also sent to the Media Control Suite. The time code is necessary for Media Control Suite to manage the experiment and give commands at the correct time. This information was originally planned to be sent via the Tascam RS422-Midi Interface device, but as explained earlier, this solution was discarded. Sending time information over IP can be risky though. Time information is obviously sensitive to errors, especially delays in communications. This solution works fairly well, but is not by any means a perfect solution. Using a RS422-Midi interface would be highly recommended as it much less delay prone.

4.2 Networking library

The network library was designed to allow sending of complete object instances. This is a practical way of sending information, as it is easy to pack information in an object. This object has to inherit from a Packet class. The network library consists of a package

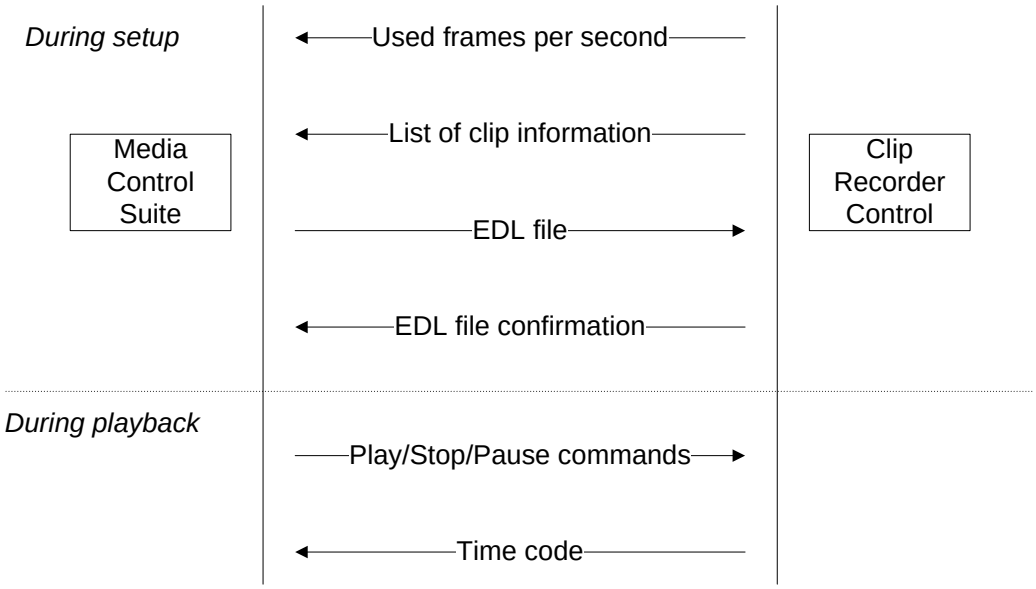

Figure 4.2: Communication between Media Control Suite and Clip Recorder Control

called packetNetworking. The main classes of the library is the Server and Client classes. A server may connect to several clients, but the client may only connect to one server. These classes will be described in more detail in the next subsections. To send packets, an ObjectSender instace is needed for encoding packets. Figure 4.3 shows how the network library works. The MessageListener class is a simple thread that reads bytes from the network stream when data arrives, processes the stream and creates a Packet from the received data. This is then sent to the owner of the thread. The role of the MessageListener is described in more detail in the Server and Client subsections. Note that both the Server and the Client class has its own ObjectSender and its own MessageListenerThread.

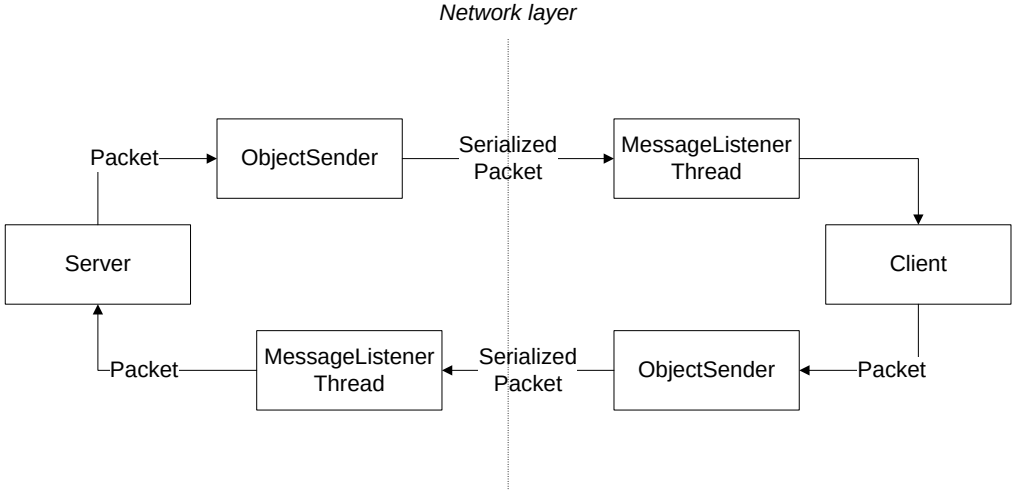

Figure 4.3: Network library overview

4.2.1 Packets

A Packet is an abstract class, and only has a a serialVersionUID to identify the class. Packet implements the Serializable interface, see [7], which is an empty interface, but indicates that the class can be serialized. Serializing is a process where the object instance is parsed into a list of bytes. This list of bytes can be sent via sockets over IP. As

mentioned, the Packet class is abstract, hence it can't be instantiated. Because of this, all other packets must inherit from this class. The network library has two other Packet classes. One is a TestPacket class, which is not used, but serves as an example of how Packets can be used to send information. The other class is a StringPacket, which, as the name indicates, contains a string. This string is its only attribute, making the string the only information this class carries. This Packet is used to maintain connection status for the Server and Client classes. The ping routing is explained in section 4.2.4.

4.2.2 Server

The Server class is a class that governs the functions of a server. The server may connect to several clients, but the number of clients is set when the server is instantiated. The server is instantiated by a list of port numbers the server should use open for connections. Figure 4.4 shows the Server class as shown in a UML (Unified Modeling Language) diagram. This section will discuss the most important methods and fields. The Server class inherits from the Communicator class. The Communicator class only holds the input and output streams and also a Talk method. The Communicator class is abstract, and only serves the purpose of reducing the amount of code. To connect to a client, the server needs access to a ServerSocket and a corresponding ClientSocket. To be able to send objects, an ObjectSender is also required. Before a connection has been established, a thread waiting for a connection is established. When a connection has been established this thread starts a thread listening for messages and dies. The mentioned properties are all per client, so the Server class contains a list of these properties, the list length being equal to the number of supplied ports. The server also contains a StatusListenerThread. This thread starts when the server is instantiated and keeps tabs on which clients are open by using ping information. If three pings fail, the connection to that client is considered broken. The server sets a boolean for each client a ping is sent to, so that the StatusListenerThread has access to that information. The server also have a boolean for whether all clients are up or not.

Methods

- The ProcessPacket method is used to handle packets received from the MessageListener. The server class only accepts ping messages. These are StringPackets and starts with the string "ping".
- Talk(Packet pck) is overridden from Communicator. It is used when the Server needs to send a message to all clients.
- Talk(Packet pck, int index) is the same as above, however, a index to specify which client to send to is mandatory. This method only sends to one client.
- Listen simply starts a MessageListenerThread for each Client.
- UpdateStatus(int index) is a method called when there is need to update connection status for the Server. The input is the index of the Client that has changed its status. If the index is positive, it indicated that the connection is good. If the index is negative, it indicates the connection has problems. The negative index is used by sending the negative index minus one. This is to avoid confusion with index=0.

The MessageListenerThread is a class on its own. It is used for receiving serialized Packets and decoding them with the Deserialize method. After deserialization, it calls the owners ProcessCommand method with the Packet as its input. This way the server or client gets the Packet that was received over IP. The StatusListenerThread sends pings and

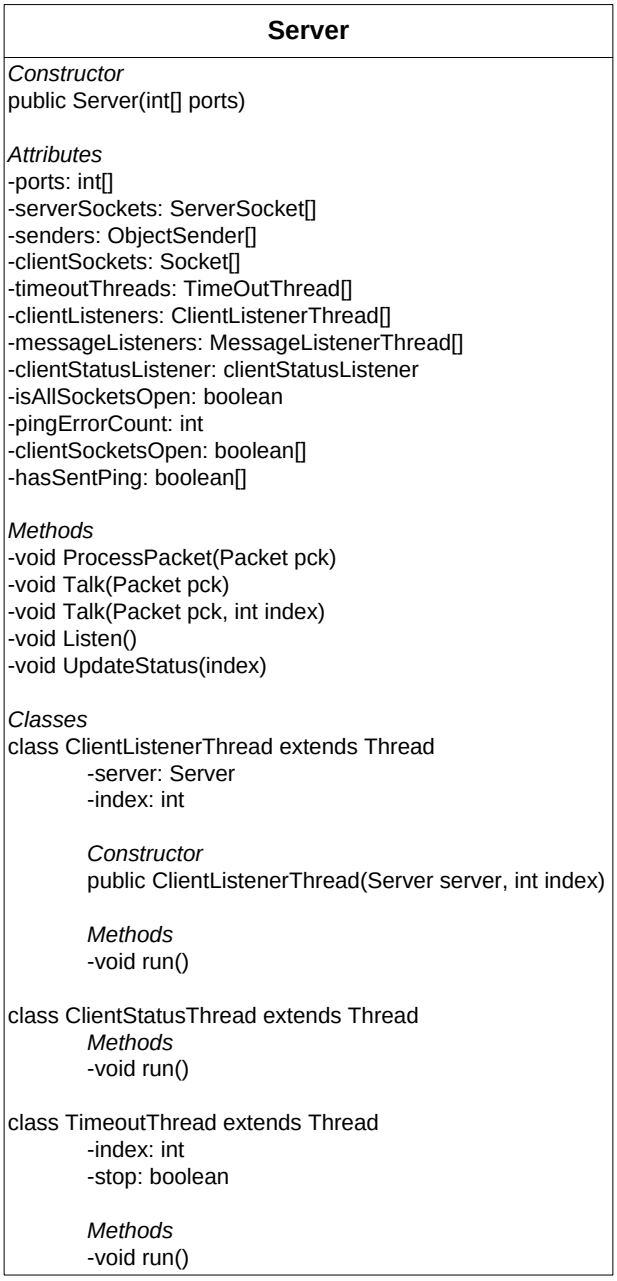

Figure 4.4: Server UML

starts a TimeoutThread for each of the clients. If a client does not respond in time, the UpdateStatus method is called.

4.2.3 Client

The Client class also inherits from the Communicator class. In this way it is no different than the Server class. The Client however, can only connect to one server and is therefore limited to only one input and output stream held by the Communicator class. Figure 4.5 shows the Client class in the form of an UML diagram. The Client class is simpler than the Server class. This is because it has a specific address to connect to. As seen in figure 4.5 the constructor takes both an address and a port number. The address can be in two formats. As DNS (Domain Name System) or as an IP address. The address is therefore an instance of Object since these two formats have little in common. The address is parsed to find out the format in the Connect method. Connect is called only the first time the

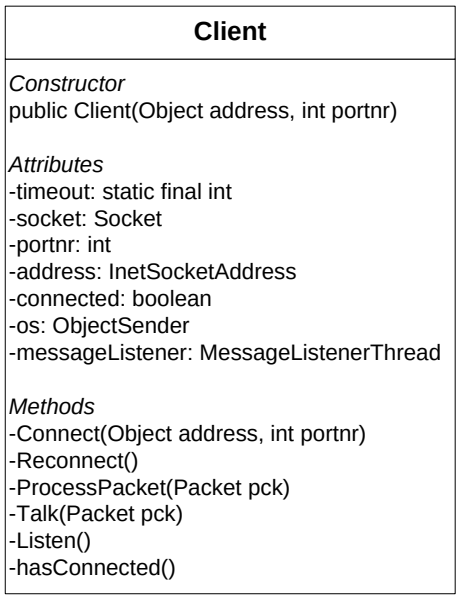

Figure 4.5: Client UML

client connects to the server. Connects calls the method Reconnect at the end to establish a contact with the server. The rest of the Connect method is instantiation of streams and address parsing. The Reconnect method connects to the server with a timeout of 1000ms. After setting up the connection, a MessageListener is started and the method hasConnected is called. The boolean connected is naturally set. As with the Server class, the ProcessPacket method only processes ping commands. To get access to other types of Packets, the Client class should be extended and override the ProcessPacket method. As with the Server class, Talk is overridden. It simply calls on the ObjectSender to send a given Packet. Listen has the same functionality as in the Server class.

4.2.4 Ping routine

To maintain a connection status, the server sends a ping to the client each 1000ms. The interval is hard coded, but easy to change in the source. It is located in a class called ClientStatusListener in the Server class. The ping also gives the index of the client when sending the ping. The client responds when receiving the ping. When it receives the ping, the connection to the server is successfull. When the server receives the response, it knows that it has a valid connection to the client. A ping is a StringPacket, and packs its information as a string followed by the index number of the client.

4.3 Media Control Suite

The Media Control Suite application is the main application. It controls most of the the information available, including the full subjective assessment. There is too much code to go into detail about everything, so only the most important classes and critical methods will be discussed. The source code is attached digitally for further review. Media Control Suite contains of seven packages. Three of them contain SALT in its modified form. There is one GUI package and a mediaplayer package containing the central classes. There are also a package with the network library, and a package with classes representing Packets to be sent using the network library.

4.3.1 GUI

The GUI for the Media Control Suite is based on using the Netbeans GUI Builder. This simplifies building a GUI substantially. The main GUI class is the MediaPlayerView class. This class holds the main GUI, with all buttons and its functionality. There is also two other frames active in the GUI. A MIDI message window and a status window, both are controlled by their own classes. There are also two different dialog classes, the ItemDialog and TrialDialog. The application uses dialogs from SALT as well. Most importantly the DesignDialog and the SessionDialog. The name of the dialog class points to what kind of dialog it brings up. Their functions are in that respect self-explainatory, and therefore these classes won't be discussed in further depth. The focus will be on the main windows, as that is were most of the user actions will be.

Main window

A screenshot of the main window is shown in figure 4.6. The screenshot also includes the MIDI message window and the Status window. It is divided into three main parts. The top part, as seen in the figure, concerns SALT concepts, design and session. These have their own "New", "Edit", "Open" and "Save" buttons. These buttons each have their own ActionPerformed methods in MediaPlayerView. These methods simply bring up the relevant dialogs.

The middle part of the window contains a playlist window and buttons related to trials. The playlist windows is a JList with custom components. The elements in the list are of the Trial class, which inherits from JPanel. The Trial class will be explained in 4.3.3. The list is organized using a DefaultListModel. DefaultListModel is used to add or remove elements from the list. It is attached to the JList by using a custom constructor with *new DefaultListModel()* as its input. Since the list contains JPanels, the JPanels paint method will be used. Trial is also build using Netbeans GUI Builder, making the JPanels in the list based on the design of a Trial. This only paints the Trial as it is built though. Having different colors for selected trials and unselected trials is preferable. To achieve this effect, a custom PanelCellRenderer class is used. This class implements the ListCellRenderer interface [5]. The getListCellRendererComponent method is called during repaint, and is used to paint a blue background for selected trials. In addition to the JList, buttons for trial handling and for randomizing are included in the middle part. The "New", "Edit" and "Delete" buttons also have their own ActionPerformed methods. These methods are just relayed to the Main class however, since that class has easier access to the necessary fields. There is also a drop-down menu for choosing the number of items per trial. If choosing a number higher than the current number, nulled trials are added. If choosing a lower number, all trials over the new number of items per trial are nulled. The "Randomize" button starts the randomizing process explained in 4.3.3. The "Shuffle?" checkbox sets

whether the items should be shuffled in the randomizing process or not. The "Clear" button clears the playlist of trials.

The bottom part of the main window includes buttons for hiding and showing the MIDI message frame and the status frame. Also included are buttons for starting and stopping the experiment. As with the trial buttons, the "Start" and "Stop" buttons are relayed to the Main class. The MIDI message window and the Status window are controlled from the MediaPlayerView class.

Status window

The status window shows, as the name implies, the status of the application. It consists of five status messages and a message window. The message window is used by the application to give feedback to the user when operations fail or something necessary is missing. The status window gives information about the state of five experiment requirements. The first two requirements are the need for specifying a design and a session. The third concerns the connection to the Clip Recorder Control application. If the connection is good, this message will show "Connected" as its status. The forth is whether or not a file list has been received from Clip Recorder Control. These four requirements must be fulfilled before starting the randomization process. The randomizing is the fifth requirement before an experiment can be started. A full explanation of the use of this application and to set up an experiment is included in section 4.5. The status window is governed by the Status class. The Status class contains get and set methods for all the five requirements and a method for adding message to the message window. It also contains a method that returns a boolean that is set when all requirements are fullfilled.

MIDI message window

The MIDI message window simply shows MIDI message received from the feedback device during the experiment. This window is almost equal to the SALT main window and mainly writes out the same messages as in SALT. This windows is used to double check that message from the feedback device are received, and in the right format.

4.3.2 Key classes

The key classes in the Media Control Suite are those connected to experiments and the control of these classes. Design, Session, Playlist, Trial and Item are all essential to having a experiment. The Main class controls these classes. Each of these classes are described below.

Design

This class is a modified version of the class in SALT, described in 3.5.2. There's not alot of changes made, but the changes made are significant. In SALT, trials and their items were a part of the design. This part of the Design class is discarded. The full playlist, including all trials and items are now a governed by the Playlist class. The Design class has static access to this list, but it is not used directly. The DesignWriter and DesignReader classes however, uses the playlist to store it with the design, similar to what was done in SALT. Obviously, trials and items in SALT has also been removed from the DesignWriter and DesignReader classes as well. Another change in the Design class, is that all GUI code is discarded.

Fader buttons are introduced the modified Design class. These buttons are a part of the

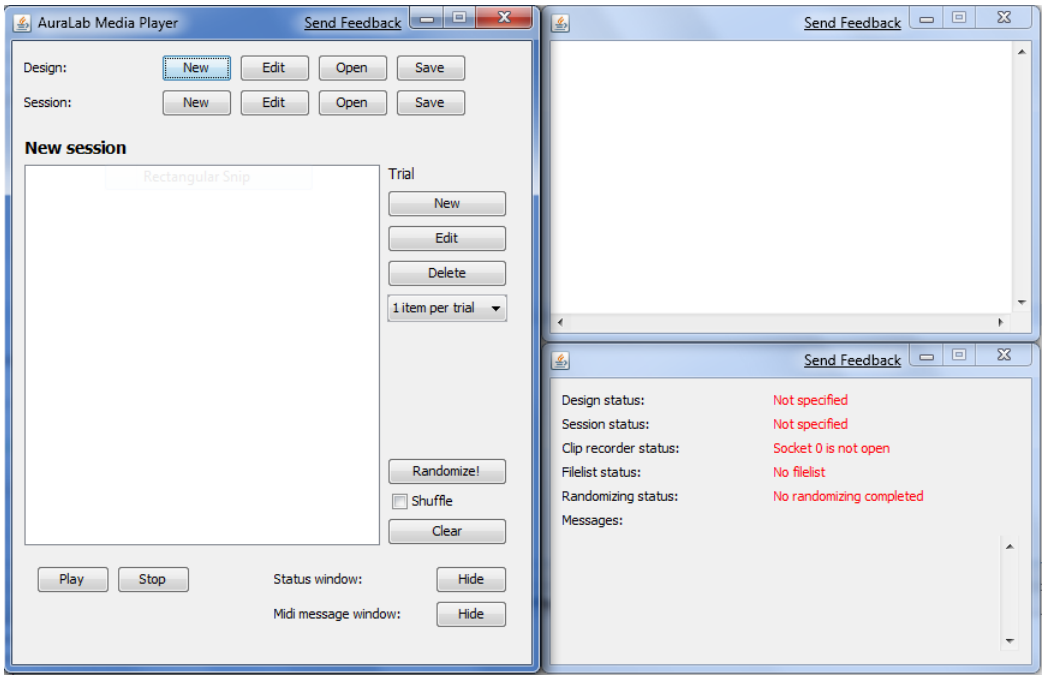

Figure 4.6: Main window

feedback device. There is one button per fader, and pressing the button shifts playback to the item in the active trial. The MIDI controller number for these buttons are stored in the design. This is also modified in the DesignDialog class, where an extra drop down box is used to specify the button controller number. In addition to storing these variables, the Design class now also have get and set methods for these variables. With the addition of possibly using MTC to retrieve time code, there is also added another MIDI device. This device is called timeCodeInputDevice in the source code. This device also have get and set methods. Specifying this device is done along with the other MIDI devices at the MIDI devices tab, explained in 3.5.2.

Session

The Session class is pretty much unchanged from SALT. The only modification done is introducing a List of strings to log MIDI activity during the experiment. Every time the test subject moves a fader, or presses a button this is recorded in this list. The time code position at the moment of the activity is also logged. A major difference from SALT however, is that the experiment does not start automatically after specifying a new session. This is due to the fact that connection with Clip Recorder Control must have been established, and that randomizing must be completed before starting the experiment. Usage is described in more detail in 4.5.

Playlist

A Playlist is mainly the list of trials, stored in an ArrayList. The Playlist class have a String denoting the name of the playlist. There are also two static properties in the Playlist, one is the DefaultListModel (see 4.3.1) for the JList in the main window. The second static property is the number of FPS used in QuickClipXO. Note that this implies that all items in all trials use the same number of frames per second.

The most important part of the Playlist class is the randomize method. This method sets

up the experiment according to the specifiations. The randomizing is done in four parts.

1. Step one is checking conditions for the experiment. This part simply checks the status of Media Control Suite, to see if the first four requirements for starting an experiment is fulfilled. These requirements are specifying a design, specifying a session, establishing contact with Clip Recorder Control and receiving the list of files from Clip Recorder Control. The randomizing method also checks whether all items have been set. When the playlist is created, it is created with nulled items, and all items must be set before randomizing can take place.

2. Step two is the shuffling of trials and items. This is done using the shuffle method in Java Collections [2]. The Playlist class has its own shuffle method, where, after Collections.shuffle has been called, the index of trials and items are set to the new position in the list. Shuffling is optional, and chosen with the "Shuffle" checkbox in the main window. The shuffle method is also validated with the isListOk method. If the list is not valid, it reshuffles until a valid list has been generated. The only criteria for a valid list is that two items with equal first tag is not set to be played consecutively. If no valid list can be found in 15 seconds, randomizing will quit with an error message.

3. Step three is time code and .edl file generation. The QuickClipXO software starts clips from one frame before zero time code. This can sometimes cause problems, since the transition from i.e 23:59:59:24 to 00:00:00:00 can be troublesome. To avoid this, a minute of frames is added before the first item to be played. Between items a 20 second gap is added to ensure that clips are not started consecutively before Media Control Suite has time to start another clip. Since the start and stop times are in the format of time code, conversion methods between frame count and time code are used. The .edl file is generated according to the format shown in [10], where each item gets its own line, pointing to a file path on the Clip Recorder hard drive.

4. Step four is sending the .edl file as an EDLPacket to Clip Recorder Control and wait for a response. If a response has not been received within five seconds, the randomizing process fails and an error is printed at the message field in the status window. To update the JList in the main window, a rearranging of trials is done at the end of the randomizing process, as well as setting the status to "Randomizing completed". The first trial is also set to be active so the experiment starts with the first item of the first trial.

Other than the randomize method, the Playlist class consists of methods used in randomizing and get and set methods. These methods have self-explainatory names and are not discussed in the thesis.

Trial

As explained in the GUI section, the Trial extends JPanel, and is built with Netbans GUI builder. The non-graphical parts of the Trial class, is however fairly simple. A fixed length list of items are stored in a trial, as well as the static value itemsPerTrial, and get and set methods for this value. A trial also has a unique identifier and a counter to generate the trial ID. The Trial class does not contain much code, and have fairly self-explainatory method names. The source code is attached digitally for further reference.

Item

The Item class is a small class, consisting of enough information to play a single clip with QuickClipXO. Each item has its own name, a filename, a list of tags, an instance of the

Clip class and start and stop time code information. There are methods for getting and setting tags using a single String, where each tag is seperated by a line break "\n". An Item instance may be instantiated by providing an existing item (making a copy) or with name, filename and a Clip instance as input.

Main

The Main class controls most aspects of the Media Control Suite application. It contains a Playlist instance, a list of clips received from Clip Recorder Control, an instance of the Status class, the current time code, dialogs and an instance of the Salt classs (with GUI code removed), and a few more attributes. It also contains an instance of the GUI class, MediaPlayerView. The constructor creates a playlist and a status. The Main class UML diagram is shown in figure 4.7. For more information, see the attached source code.

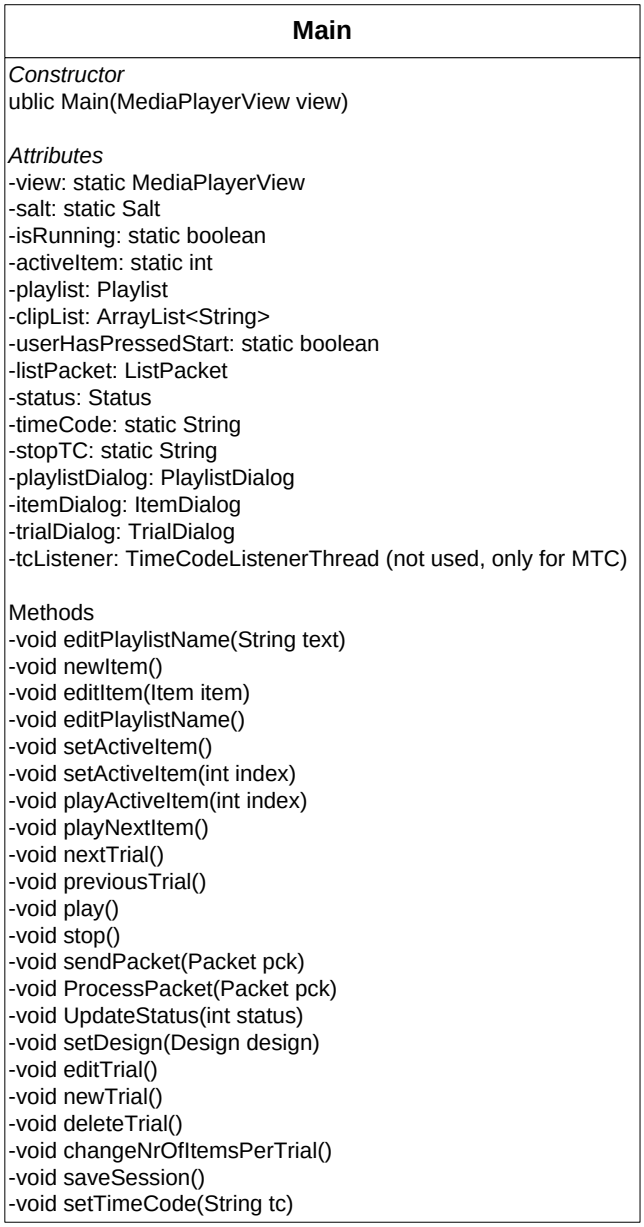

Figure 4.7: Main class

The Main class extends the Server class from the network library, and also overrides the

ProcessPacket method from that class. This method is important as it interprets information received from Clip Recorder Control. Using the *instanceof* keyword, the type of Packet received can be determined. What kind of information Clip Recorder Control sends to Media Control Suite is described in section 4.1. The Main class has its own sendPacket method, but this simply calls the Talk method in its parent class, Server.

Since most actions from the GUI are relayed to the Main class, methods for editing the playlist name, trials and items are in the Main class. These methods include newItem, editItem, editPlaylistName, editTrial, newTrial and deleteTrial. Depending on the action, these methods may bring up the appropriate dialog. There is one dialog class for each of the three levels, PlaylistDialog, TrialDialog and ItemDialog. The PlaylistDialog is brought up by clicking the name of the playlist, by default "New playlist".

Another group of important methods are commands for the Clip Recorder Control applications. These methods are self-explainatory, like play and stop. They work by finding the active trial and active item and sending an appropriate command to Clip Recorder Control. The PlayPacket is used for nextTrial and previousTrial as well. The PlayPacket contains the name of the clip, in addition to a start and stop time code position. The stop time code is a field in the Main class as well, and is set to the stop time position of the clip to be played.

The stopTC field (stop time code) is used when a new time code is received, to check whether or not to stop playback. A playNextItem method is also included in the Main class. It is called by a the setTimeCode method, which is called for every new time code received. The playNextItem method finds the next item to play, by incrementing the current item number. If the new item number is larger than the number of items per trial, it jumps to the next trial. If the current item being played is the last in the playlist, playback stops.

4.3.3 DumpReceiver modifications

The SALT DumpReceiver class was created with SALT in mind. Some adjustments were necessary to work reasonably with Media Control Suite. The most notable change is the removal of all GUI components. Replacing the original GUI code was calls to the MIDI message window. Since all MIDI activity is shown in the MIDI message window, this was a necessary modification. The Behringer BCN44 also outputs only with status byte 0xB0, and this is reflected in the DumpReceiver class, where buttons for next and previous trial are assumed to output status bytes of 0x90. The introduction of a button for each fader is also reflected in the code.

4.4 Clip Recorder Control

Clip Recorder Control is a fairly small and simple application. It comprises of only two classes, in addition to the network library and the Clip Recorder SDK. A Main class and a GUI class. Clip Recorder Control communicates with Media Control Suite as described in section 4.1. Figure 4.8 shows the Main class in a UML diagram. The GUI class will be described shortly below.

The Main class contains most of the functionality for Clip Recorder Control. It extends the Client class from the network library, and overrides ProcessCommand and hasConnected. The ProcessCommand is overridden for transmitting and receiving Packets from

Main

Constructor
public Main()

Attributes
-mci: static MediaCmdIF
-isConnectedToMCR: boolean
-fps: int
-hasGeneratedList: boolean
-configPath: String
-edlPath: String
-edlName: String
-lc: ListCommand
-timeCodeTransmitter: TimeCodeTransmitterThread

Methods
-void disconnectMCR()
-void ProcessPacket(Packet pck)
-String framesToTC(int totalFrames)
-int TCToFrames(String tc)
-void hasConnected()
-void setFPS()
-void sendFPS()
-connectToMCR()
-void generateList()
-String imageInDirectory(File f)
-ArrayList<String> getAllFilesInDir(File f, ArrayList<String> list)
-void getClipLengths()
-ArrayList<String> readTextFile(String path)
-void writeTextFile(String path, ArrayList<String> content)
-void sendList(ListPacket list)
-static void main(String[] args)

Classes
TimeCodeTransmitterThread
Constructor
protected TimeCodeTransmitterThread()

Attributes
-stop: boolean

Methods
-run()

Figure 4.8: Main class Clip Recorder Control

the server. The hasConnected method is called when the client connects to the server. This method sets up most of the functionality for the application. Firstly, it connects to the QuickClipXO software using the Clip Recorder SDK. The connection and access to the SDKs methods is maintained with a MediaCmdIF object. This connection is established with the connectToMCR method.

After connecting to the QuickClipXO software, the currently used FPS is found using static member fields in the TCXlat class. This class gives low level access to most properties for the QuickClipXO software. It is however, poorly documented. There is a flag called TC2_TCTYPE_MASK that, as the name indicates works, like a mask for the time code type. Doing an AND operation with the mask and the time code type returns one of the nine possible video formats. Each of these formats have an *int* with the FPS for the video format. Using a switch/case statement, the correct FPS can be found. The setFPS method in the source code shows how this was implemented.

After the FPS has been found, the list of eligible clips is generated. This is based on folders specified in the config file. The specified folders are searched through for folders containing .tga or .tiff files or single .mov files. These folders and files are added to a list of eligible files. Each of these files are added to QuickClipXO in clip mode. This is to use the SDK to get information about the length of the clips in frames. After information about the clips have beeen found, the clips are removed. The length is then converted to a String and added to each clip. The FPS is sent first, followed by the file list.

The Main class also contains methods that reads and writes a text file. This is used to read the config file, and write the .edl file when it is received. The .edl file is added as a clip to QuickClipXO, and used as the only active clip during an experiment. During the experiment, commands are relayed from the Media Control Suite to QuickClipXO using the MediaCmdIF instance created when connection to QuickClipXO was established. Play commands are relayed using the PlayClipFromTo method in MediaCmdIF. The Main class also contains a thread that sends time code every 30ms. The interval is set this long because error rates for the Packets tend to rise sharply as the interval is shortened. This is the main reason why MTC would be a better solution, it is much more stabile.

The GUI class sets up a very small and simple GUI. The GUI consists of a JLabel, a JTextField and two JButtons. There's a "Start" and a "Stop" button. The JTextField is used for writing the address to the Media Control Suite. When the user presses "Start", the address is parsed as an IP address. If the address does not have the format of an IP address, it is used as a DNS name. If the address is the correct address, connection is established and the hasConnected method gets called.

4.5 Usage

This section will describe and explain, step by step, how to use the applications developed to hold an experiment. To start the applications, a .bat file and a .jar file is included. The .bat file simply starts the .jar file by running *java -jar jarname.jar*. Clip Recorder Control must also include a file called "config.txt". At the end, an experiment held will be briefly explained.

The order of which to start Media Control Suite and Clip Recorder Control doesn't matter. One can be started before the other, although it may be practical to start Clip Recorder Control first, since the experiment is set up in Media Control Suite. When Clip Recorder

Control starts, a GUI will show up. Figure 4.9 shows the simple GUI with an IP address written in. The user then presses start to connect to Media Control Suite. The "Start" button will then be disabled. The "Stop" button disconnects from Media Control Suite and the Clip Recorder and the exits the application. Starting Media Control Suite brings

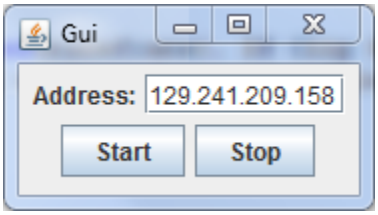

Figure 4.9: Clip Recorder Control GUI

up the windows shown in figure 4.6. If Clip Recorder Control is running on the Clip Recorder, the applications will establish contact, and Clip Recorder Control will send over the file list to Media Control Suite. This may take some time, depending on the number of files in the specified folders. In the mean time, the experiment may be set up. Note that the order of most actions is not mandatory, but is presented here for a logical way to set up an experiment.

The first step is to create a design, or load it from a file. Pressing the "New" button on the design line, brings up the design dialog, as shown in figure 4.10. All tabs should be reviewed. Filling out the used MIDI devices is mandatory. Selecting a rating form and the number of faders is also very important. Setting the correct MIDI channel numbers are also necessary to get MIDI messages from the feedback device. Pressing OK gives a "green light" for specifying a design in the status window.

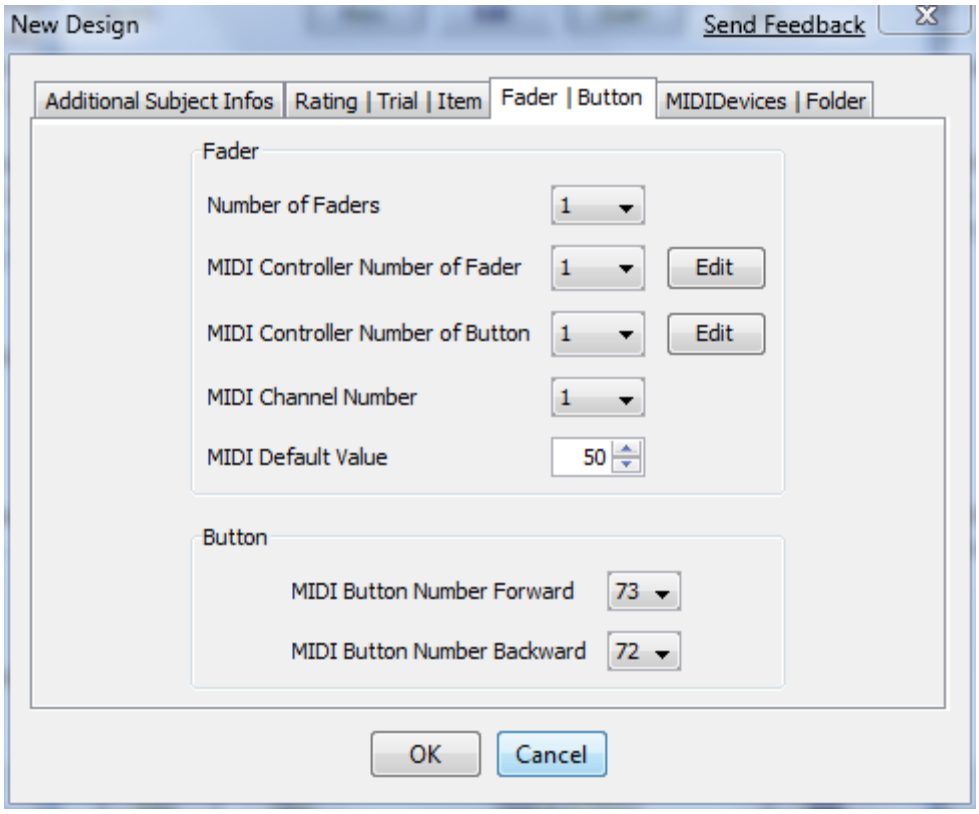

Figure 4.10: Design dialog

Next step is to create a new session. If no extra information is required from the design, this is done by simply filling out the name and age of the person doing the experiment. A simple session dialog is shown in figure 4.11. Pressing OK gives a positive message in the status window.

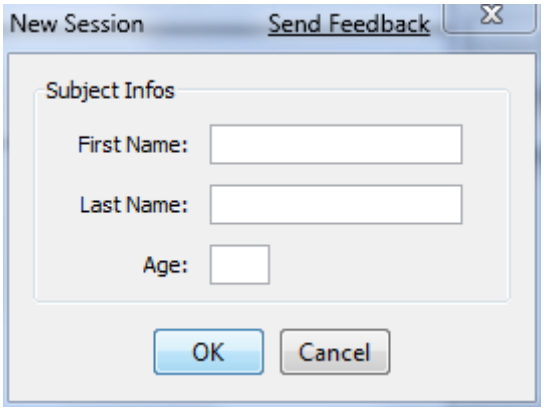

Figure 4.11: Session dialog

The next step is to set the number of items per trial. This should be done before adding trials and items to avoid removing of excessive items. It is recommended that the number of items per trial is equal to the number of faders set in the design. After this, a playlist name should be chosen.

If the file list has not been received in the mean time, the user must wait for this to be completed before starting the next step. The next step is adding trials and items. Adding a new trial brings up the trial dialog, see figure 4.12. This dialog shows the list of items in the trial, the name of the trial, and some buttons to create, edit or delete an item. Note that deleting an item only nulls it, this is a consequence of setting an equal number of items to all trials. All items should be set before pressing the "OK" button. Pressing the "New" button in the trial dialog brings up the item dialog, see figure 4.13. If the

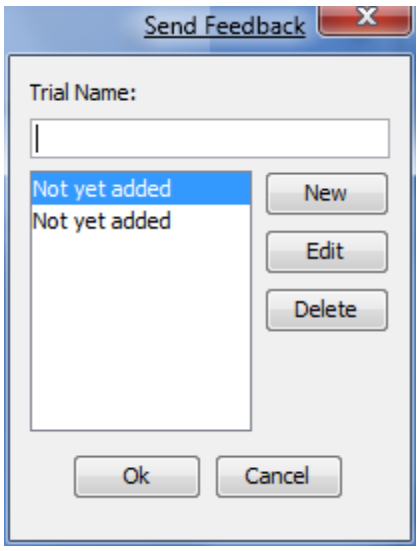

Figure 4.12: Trial dialog

file list has not been received, the list of clips is empty, and the user won't be able to

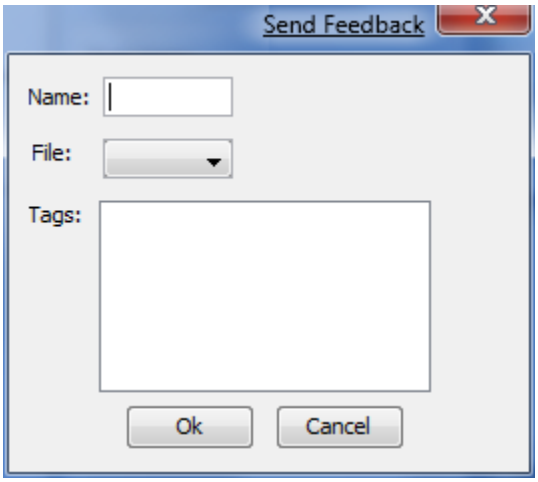

Figure 4.13: Item dialog

create a new item. The item name, as well as a clip must be chosen. To ensure that clips with similar content gets played consecutively, the user may add the same tag for those clips. This tag must be the first tag however. Extra tags may be added to give more information about the clip. Tags are stored in the design file. Pressing the "OK" button returns the user to the trial dialog. When all items are filled, pressing "OK" returns to the main window. When all trials are set, the user may save the design to be able to load the design, including the playlist, later on.

The next step is randomization. If connection the Clip Recorder Control is still good, and all items in all trials are set, the randomization begins. If the user sets the "Shuffle" checkbox, items and trials are shuffled. An .edl file is generated during randomization. This file is then added to the QuickClipXO software and the experiment is ready to start. The user may then press "Play" to start the experiment.

The test subject then controls the rest of the experiment by using the fader buttons and the next and previous trial buttons. When the experiment is over, that is when the last trial is done, the session is automatically stored in the folder specified in the design. The file name is in the format [first name]_[last name]_[age].log. If the operator, the user of Media Control Suite, stops the experiment manually with the "Stop" button, he or she needs to manually save the session, using the "Save" button on the session line.

4.5.1 Experiment

A small experiment was held to make sure the basic functionality works like it should. Three subjects were asked to rate the 'coolness' of clips shown to them. As such, the information they gave have little value, but in terms of functionality, a few bugs were found and fixed. The most crucial bug was that fader activity was not saved in the log for continuous fader rating. This bug required the experiment to be held again. Another problem found was that QuickClipXO freezed after one of the experiments. It was not possible to reproduce this, and the bug was not fixed. Two cosmetic errors were also found and fixed. The test subjects reported that some feedback on the device would be welcomed, but this is not possible with the Behringer BCN44.

4.6 Problems

4.6.1 Tascam RS422-Midi interface

The Tascam RS422-Midi device was proposed to be used to convert time code from RS422 interface to MTC. This was not achieved. It was assumed that this device was able to do this without extra instructions. After contacting the producer of this device, CB Electronics, a reply was received where it was stated that it could be achieved using position requests from the device. This was fairly late into the thesis, and there was not enough time to fix the problem.

4.6.2 QuickClipXO SDK

The SDK that can be used to control the QuickClipXO software was found to have a few bugs. This SDK is mostly used with its basic functions, like adding clips, play, pause and stop commands. However, the SDK also contains methods for converting frames to time code and vice versa. These contained some bugs. It was possible to fix them, but a simpler solution was to create new ones. They are included in the Clip Recorder Control Main class and the Playlist class in Media Control Suite. To convert between the two formats, the used FPS most be known. This was achived using static members of the TCXlat class in the SDK. These members are not documented, but have very low level access to QuickClipXO. This solution should be satisfactory and without bugs.

Another problem with the SDK was the delete method. This method is used for removing clips in clip mode in QuickClipXO. It is, along with a few other methods (blank and insert), used in this thesis, is undocumented and marked as "For internal use only". These methods are however very practical to use, and their usage was tested and their functionality bug free. The delete method however, not only removes clips from QuickClipXO, but also deletes the files they are referring to. Because of this, the use of the delete method was replaced with the blank method for removing clips from QuickClipXO.

4.6.3 QuickClipXO

QuickClipXO is the software used on the Media Clip Recorder. This software has quite a few bugs. The most prominent lack of functionality is that not all legal formats can be played. E.g. the avi format. Files with this format can be added to the clip list, but playback has substantial lag and stops after a few seconds or a few frames. This is true for most formats. A workaround was found to work however. Using MachVideo, a software found to be installed on the Clip Recorder, video files was converted to a series of 24bit *.tiff images. These images are played using the Tiff option in QuickClipXO setup and the ARGB image option. Note that ARGB images are 32bit, and not 24bit as the source images are. The reason why this seems to work well is unknown, but the lack of usable formats proved to be a problem. Using MPEG Streamclip (using a Mac) to convert into .mov files, also seem to work well. The reason for these format problems are unknown.

Another problem with QuickClipXO is that after playing a .edl file, the program does not work correctly. Restarting QuickClipXO always works though. Like the format problems mentioned above, the reason for this problem is unknown. This problem is atleast consistent.

Chapter 5

Summary

In this thesis, software for setting up an audio visual experiment and remotely controlling the experiment has been developed. The audio visual experiments are used to measure perceived quality of the presented material. The software works together with a state-of-the-art High Definition video clip recorder used to play clips. The video clip recorder is used as source for both video and audio. A function for randomizing the order of presented clips is implemented for the experiment.

Development of the software is done exlusively in Java, using Netbeans and Eclipse as development enviroments. The video clip recorder is accessed via an available SDK. Two applications have been developed, one used to set up and control the experiment, and one that relays commands and information to the clip recorder software. A network library has also been developed to enable communication between the two applications.

This document describes the process of development and the usage of the two applications. Some background material is also presented, along with description of tools used during development.

Chapter 6

Further Work

Several additions to the developed software can be beneficial. A natural feature to include in later development is the addition of an audio computer to the setup, as was originally planned for this thesis. This audio computer should receive MTC from the time code master, the clip recorder, and synchronize the audio output with the received time code. The addition of an audio computer would contribute greatly to the flexibility of the test setup.

Using MTC to synchronize the different components in the setup would be another improvement. The Media Control Suite now receives time code using IP, but this is not as reliable as MTC. Using a RS422-Midi interface should be enough to implement this modification.

A feature that could improve the use of the session logs would be a filter for the MIDI log. The MIDI log often gets large, and a filter would help extract the relevant information. Some extra functions in the main window might also be beneficial, like buttons enabling the operator to change trials.

Bibliography

[1] MIDI Manufacturers Association. The complete midi 1.0 detailed specification. http://www.midi.org/techspecs/midispec.php.

[2] Josh Bloch. Trail: Collections. http://java.sun.com/docs/books/tutorial/collections/index.html.

[3] Eclipse. Eclipse documentation. http://www.eclipse.org/documentation/.

[4] CB Electronics. *P2MMC RS422-Midi Machine Control Interface*, February 2004.

[5] Sun Microsystems. Interface listcellrenderer javax.swing. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/ListCellRenderer.html.

[6] Sun Microsystems. Interface receiver javax.sound.midi. http://java.sun.com/j2se/1.5.0/docs/api/javax/sound/midi/Receiver.html.

[7] Sun Microsystems. Interface serializable java.io. http://java.sun.com/j2se/1.4.2/docs/api/java/io/Serializable.html.

[8] Sun Microsystems. Package Summary javax.swing. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/package-summary.html.

[9] Netbeans. Documentation, training & support. http://www.netbeans.org/kb/.

[10] Drastic Technologies. Drastic edl format. http://www.drasticpreview.org/wakka.php?wakka=DrasticEdlFormat.

[11] Drastic Technologies. *VVW Interface Specification*, 2006.