UNIVERSITY OF OSLO
Department of informatics

Music Impro App – challenges in developing a musical network of mobile terminals

Master thesis
60 credits

Kjartan Vestvik

August 01. 2012

Abstract

A lot of research exist on using the mobile phone as a musical instrument. Unfortunately, less research has been done on using mobile devices to create a collaborative musical network that enables a group of people to easily play, improvise and create music collectively.

This thesis is an investigation into how to utilize a mobile platform to develop an application for musical cocreation and collaboration. A development has been conducted based on the first prototype of the Music Impro App. The aim has been to realize the concept demonstrated by the prototype in a fully functional mobile application running on the iOS platform. To accomplish this, several challenges had to be solved. The application must enable communication with other instances in a network, and musical parameters must be coordinated and synchronized to allow a coherent musical experience.

Decisions on network topology, communication protocol and messages have been made. The development has been carried out in light of the term *interconnected musical networks*.

In a musical performance, real-time response is required. Accurate synchronization of playback on all devices has been a goal. A measurement was done on the synchronization to detect timing inaccuracies. The result was used to discuss possible sources of latency and how to make improvements to the synchronization.

The aim of the study is to describe how such a system is developed, to point out the important decisions to make when constructing the system, and to identify challenges in implementing playback synchronization.

Acknowledgements

This thesis is written as a part of the masters degree in *Informatics: programming and networks* at the University of Oslo, department of informatics. I would like to thank my supervisor, Jo Herstad, for valuable support and inspirational ideas. Also, thanks to Harald Holone for help and feedback.

A big thank you goes to my fellow students in the Music Impro App project. This master thesis would not have been realised without the prior work conducted in this project, and I am grateful for being a part of such an exciting project. I would also like to thank Inven2 for giving us faith in developing the Music Impro App one step further, and IT-FUNK for financial support.

Finally, I would like to thank my wife Rita Semira for her unlimited support, kindness and patience throughout the work on this thesis.

Kjartan Vestvik, 31.07.2012

Table of Contents

1. INTRODUCTION 1
1.1 Music Impro App 3
1.2 Motivation 5
1.3 The expanding field of informatics 6
1.4 Problem Space 8
1.5 Definitions and delimitations 9
1.6 Structure 9

2. THEORY 11
2.1 Mobile Operating Systems 11
2.1.1 The major mobile operating systems 11
2.1.1.1 Android 11
2.1.1.2 iOS 12
2.1.1.3 Windows Phone 13
2.1.1.4 BlackBerry 13
2.1.2 The iOS platform 13
2.1.2.1 iOS architecture 14
2.1.2.2 Core MIDI 16
2.1.2.3 Core Audio 18
2.1.2.4 Networking in iOS 19
2.2 Terms in music technology 22
2.3 Coordination in musical performance; why and how? 24
2.3.1 The role of MIDI 25
2.4 Interconnected musical networks 25
2.4.1 Historical overview 26
2.4.2 A theoretical framework of Musical Interconnectivity 27
2.4.2.1 Goals and motivations 27
2.4.2.2 Social perspectives, architectures and topologies 28
2.4.2.3 Interaction and musical coherency 32
2.5 Communication protocols 32
2.5.1 MIDI 33
2.5.1.1 Musical instruments: basics and definitions 34
2.5.1.2 MIDI messages 35
2.5.1.3 MIDI clock 37
2.5.2 OSC 38
2.6 Latency 39
2.6.1 Latency in computer networks 40
2.6.2 Latency in musical performance 40

3. METHOD 43
3.1 Quantitative vs qualitative research 43
3.2 Research paradigms 44

3.2.1 Positivism and post-positivism 44
3.2.2 Interpretive 44
3.2.3 Critical/critical theory 45
3.3 Research in Software Engineering 45
3.3.1 Socio-technologist/Developmentalist paradigm 47
3.4 Conducting the development of MIA 2 48
3.4.1 Analyzing needs and establishing requirements 48
3.4.2 Planning and designing the system 48
3.4.3 Implementing the system 49
3.4.4 Data collection 50

4. BACKGROUND ON MIA 1 51
4.1 Overview of the system 51
4.2 Target users 53
4.3 User tests on MIA 1 53

5. DEFINING REQUIREMENTS 55
5.1 Technical shortcomings of MIA 1 55
5.2 Needs identified from user tests 56
5.3 Predefined guidelines for MIA 2 57
5.3.1 Imagined use 58
5.4 Addressing the shortcomings of MIA 1 59

6. RESULT FROM LATENCY TEST 63

7. INITIAL DISCUSSION 65
7.1 Interdependent Music Performance 65
7.2 Decisions on the functionality 66
7.2.1 Roles in the network 67
7.2.2 Communication 67
7.2.2.1 Messages 68
7.2.2.2 Communication protocol 69
7.2.3 Network topology 70
7.2.3.1 Level of centralization 71
7.2.3.2 Social organisation 72
7.2.3.3 Modes of interaction 73
7.3 Conclusion on network topology 74

8. IMPLEMENTATION 77
8.1 Network connections 77
8.1.1 MIDI Network Session 77
8.1.2 Bonjour 80
8.1.3 Making connections 82
8.2 Communication 82
8.2.1 Sending and receiving MIDI messages 83
8.2.1.1 Individual activity level 84

8.2.1.2 Total activity level 85
8.3 Playback synchronization 85
8.3.1 Preliminary discussion 85
8.3.2 Synchronization in MIA 2 86
8.3.2.1 Generating MIDI clock 86
8.3.2.2 Coordination between clock messages and audio engine 88
8.4 Result from playback synchronization measurement 92

9. DISCUSSION 95
9.1 Topology and network connections 95
9.2 Messages and communication 97
9.3 Synchronization 99
9.4 Timing performance in synchronization 100

10. CONCLUSION 103
10.1 Future research 104

11. BIBLIOGRAPHY 106

Appendix A 111
Appendix B 112
Appendix C 113

Figures and tables

Figure 1: Layers in iOS 14
Figure 2: Structure of Core MIDI 17
Figure 3: MIDI Network Setup in OS X 18
Figure 4: Bonjour: browsing for a service 21
Figure 5: Bonjour: resolving a service 22
Figure 6: Note lengths 23
Figure 7: "Fairlight" software sequencer 23
Figure 8: Process- and structure-centered IMNs 28
Figure 9: Centralized and decentralized topologies 29
Figure 10: Centralized and decentralized topologies with connections 30
Figure 11: Assymetric, centralized topology with weighted gates 31
Figure 12: Sequential, symmetric topology 31
Figure 13: MIDI message 35
Figure 14: MIA 1 systematic overview 52
Figure 15: Results from Ping test 63
Figure 16: Overview of communication in MIA 2 68
Figure 17: Topology and interconnections in MIA 2 (no rhythm) 75
Figure 18: Topology and interconnections in MIA 2 (with rhythm) 76
Figure 19: Endpoints and ports in Core MIDI 78
Figure 20: Calculating timestamps of MIDI clock messages 88
Figure 21: Coordination of sequencer and generated clock messages 90
Figure 22: Calculating time difference between audio callback and sync step 91
Figure 23: Measuring time deviation in Ableton Live 93

Table 1: Research paradigms 46

1. INTRODUCTION

The use of smartphones has exploded the last couple of years. Being a powerful device, the smartphone is capable of running quite advanced programs and demanding calculations. The ability to install and run applications, or *apps* as they are commonly called, transforms the smartphone into a device able to perform all kinds of duties, such as a camera, a GPS device, a gaming console, a media player, or a musical instrument. The marketplace is flooding with apps that can turn the smartphone into any kind of musical instrument, like a flute (Smule´s *Ocarina*[1]), a drum machine (Synthetic Bits´ *Funkbox*[2]), and even a real-time effects processor and recording device (IK Multimedia´s *Amplitube*[3]). Many of these apps can produce really inspiring and high quality results. It is even possible to connect external hardware devices[4] to the phone to extend the way in which you can interact with the app.

However, when it comes to apps that aim toward collective music performance, the possibilities are limited. Of course, it is possible for several people to meet, each with their own smartphone running a musical instrument app, and just play together by using their individual phones. But it might be difficult to coordinate the performance and to make the instruments "fit" together. If several people were using instruments that need to keep track of tempo, for instance a drum machine or a synthesizer playing a rhythmical sound, it would be nearly impossible to have these devices play together with the tempo and the beat matching perfectly. When it comes to game apps, multiplayer games exist that allow each player to use their own phone, and to share and connect to one common game. But as for musical apps, this feature is almost absent. Wouldn´t it be nice to have a music app that several people could participate in at the same time to create music together?

Using mobile phones as musical instruments and processing devices can be seen as an evolution of laptop orchestras, offering extreme mobility and new possibilities for interaction

1. http://www.smule.com/ocarina/
2. http://syntheticbits.com/funkbox.html
3. http://www.ikmultimedia.com/products/amplitubeiphone/
4. A MIDI keyboard, for instance.

through sensors, touch screen, etc (Wang, Essl, & Penttinen, 2008). Research on using the mobile phone as a musical instrument has been conducted by several authors (Wang et al., 2008), (Essl & Rohs, 2007). Networked musical systems is also researched by some people, and one main contributor in this field is Gil Weinberg (Weinberg, 2003), (Weinberg, 2005). But when it comes to the intersection of these fields, i.e. using modern smartphones to create a networked musical system, very little research can be found. There have been certain contributions to exploring how interconnected mobile phones can be used in a musical performance setting (Schiemer & Havryliv, 2007). Although smartphones have been around for about twelve years[5], the field of using these devices as mentioned seems to be in its immaturity.

5. Ericsson R380, the first device marketed as a smartphone, was released in 2000. (Apple´s iPhone was introduced in 2007)

1.1. Music Impro App

In this section I will give an introduction to the Music Impro App project. This project forms the basis for my study.

The Music Impro App is an application developed for smartphones that enables several users to create and improvise music together (Skotterud, Madsen, Sethre, Vestvik, & Bording, 2011). The idea behind the project is to utilize features of modern technology to give users an easy and inspiring way of creating music together, without needing any musical knowledge. Music is often seen as a "language" through which people can express themselves and communicate with others (Gabrielsson & Juslin, 1996), (Holmes, 2011). Some people might have limited possibilities of communicating with others because of disabilities or lack of speech. One important goal of the Music Impro App is to provide people with a way of expressing themselves and communicating with others, and that even children, elderly or people with disabilities should be able to do so. We did not want to target the application to a specific group of people as this might lead to stigmatization (Plos & Buisine, 2006). Instead, we wanted to follow the principles of universal design in that the system will be designed to be usable for all people to the greatest possible extent (Story, Mueller, & Mace, 2011). Modern smartphones comes equipped with motion sensors that can be used to detect how the phone is moved (Essl & Rohs, 2009). We wanted to utilize these sensors so that users should be able to create music by simply moving the phone in various ways instead of interacting with the screen.

The Music Impro App initially started as a student project in a masters level course[6] at the University of Oslo, spring 2011. During this course, a prototype of the application was developed. This prototype will be referred to as *MIA 1* (Music Impro App, version 1). MIA 1 will be explained in detail in chapter 4.

The experiences gained from the work on MIA 1 suggested that the idea had great potential[7] but that the concept needed some refinement to better achieve the goals of the project. A

6. INF5261 - Development of mobile information systems and services

7. Having received the 2. price in the contest *Idéprisen 2011* by *Inven2* was an indicator for the potential of the project.

major contribution to further development of this project has been the financial support from the Research Council of Norway, granted through the IT FUNK[8] programme. These funds gave us the opportunity to start developing an improved version which will be referred to as *MIA 2*. MIA 2 was intended to realise our initial idea in a more complete manner. The challenges that arose when developing this version engaged me to write this thesis on the subject. As such, this thesis is centered around the development of MIA 2.

The Music Impro App is connected to the RHYME research project, which is a collaboration between Institute of Design, Oslo School of Architecture and Design, Institute of Informatics, University of Oslo and Centre for Music and Health, Norwegian Academy of Music. The RHYME project explains its goal in this way:

> *"The goal of the RHYME project is to improve health and life quality for persons with severe disabilities, through use of "co-creative tangibles". These are ICT based, mobile, networked and multimodal things, which communicate following musical, narrative and communicative principles. They are interactive, social, intelligent things that motivate people to play, communicate and co-create, and thereby reduce passivity and isolation, and strengthen health and well-being."* (Cappelen & Herstad,)

By creating the Music Impro App, we intend to come up with a system that is easy to use and that can inspire cooperation and cocreation among people. The intention is that using the application does not require any musical skills, and that it will be accessible for everyone, even children and people with disabilities. In this way we want to support the goals of RHYME.

8. IT for funksjonshemmede - ICT for the disabled

1.2. Motivation

Music plays an important role in many people´s lives. We use music for entertainment and relaxation. Many people talk about the positive effect music can have on health (Solli, 2009). Even Ruud discuss the value of music in everyday life, and the close relationship between music, identity and health. He mention four areas where music may contribute to the quality of life (Ruud, 1997):

- music may increase our feelings of vitality and awareness of feelings
- music gives an opportunity for increased sense of agency
- music-making provides a sense of belonging and communality
- experiencing music creates a sense of meaning and coherency in life

Further, music may also be used in therapy, in a situation involving a therapist and a patient. In these circumstances music may be used as a way of communicating for people with disabilities and people without language (Robertson, 2001).

Learning to play an instrument takes a lot of time and effort, and often requires fine-tuned body control. Persons with disabilities might be excluded from the joys of expressing themselves through music. Utilizing modern technology to provide a way for people to express themselves and communicate, people that otherwise are restricted from expressing themselves, is a major motivation behind this study.

One other motivational factor for me is my background as a musician. I have for many years been playing keyboards and synthesizers in different bands, and I have also been running a recording studio. Performing and creating music together with others is to me a joyful and rewarding experience. I also find the possibilities of new technology very interesting and inspiring. Being able to exploit new technology in such a way that more people get the chance to share a musical experience is a goal that I strive for.

Modern technology and social media make it possible for people to be "connected" all the time while not being together, but do we actually get closer to each other? Sherry Turkle question how technology shape the way we connect to one another in her TED talk *Connected, but alone?* (Turkle, 2012). Howard Rheingold studied the Amish people in USA

and their attitude towards technology. While this people is known for its refusal of modern technology, they have accepted the mobile phone. Rheingold discovered that their conservative approach is not just a plain refusal of new technology; when faced with something new they ask themselves the following simple question: "Does it bring us together, or draw us apart?" (Rheingold, 1999) I think this is a relevant question to ask also when developing new artefacts and new software applications.

1.3. The expanding field of informatics

The field of information technology can be defined in various ways. Broadly speaking, the field is concerned with technology to treat information. A more specific definition can be stated like this:

> "The technology involving the development, maintenance, and use of computer systems, software, and networks for the processing and distribution of data" (Merriam-Webster.com,)

The history of information technology and systems stretches back as far as to the creation of symbols and letters. During the years, input technologies like pen and paper got invented, and then the first books appeared. The next stage of development came with the mechanical age that brought machines for calculating numbers and the first examples of analog computers. The discovery of ways to utilize electricity led to the electromechanical age and the beginnings of telecommunication, the use of punch cards and the founding of IBM. The electronic period started in the 1940s and led to the computer as we know it today (Butler, 1998).

From being devices mainly used for calculations and processing and storing data, computers have taken on new fields of use during the last decades: controlling how a car behaves on the road, clinical care at a hospital or at home, an entertainment medium used to view photos, movies and listen to music and radio stations, and also the computer can be used as a musical instrument. As a consequence of this, constructing information technology that satisfies the needs and demands of today often requires knowledge from many different disciplines. Creating an IT system to be used in a hospital, for instance, relies heavily on knowledge from the medical discipline. Also, computers, and even cell phones, are no longer being used for

only one specific task (performing a complex calculation, or making a phone call). Rather, they have evolved into complex machines that we use for all sorts of duties in our everyday life. This has generated a need for usability; the technological artefacts are being used by everyone, and as such they should not require special skills or knowledge to be handled by the general public. Fields within informatics that specialize in design, user interaction and usability accommodate these needs.

Mark Weiser envisioned a future of what he called *ubiquitous computing*. He imagined that information technology would "weave themselves into the fabric of everyday life until they are indistinguishable from it" (Weiser, 1991). In the western world today, with portable computers, tablets and smartphones becoming increasingly popular, and the internet being accessible whether we are taking a walk in the park, cruising 30.000 feet above ground in a Boeing 737, or sitting at the office, it is not difficult to see similarities with Weiser´s vision.

1.4. Problem Space

This thesis describes an investigation into the development of a musical network using mobile terminals. By a *musical network* I mean a network of wirelessly interconnected devices that allow players to share and interact with a common musical performance in real-time. The investigation originates from previous work in the Music Impro App project. The primary objective is to investigate how a fully functional version of the Music Impro App can be realised on Apple´s iOS platform. The version to be developed is referred to as MIA 2. My main research question can be stated like this:

> *How can the iOS platform be utilized to implement MIA 2, and what are the challenges when it comes to playback synchronization in the system?*

The development is carried out in the light of experiences from the work on MIA 1 combined with research on musical collaboration and *interconnected musical networks* (IMNs).

In order to answer the research question, the following tasks will be performed:

- *Describe the development of the network connectivity and communication functionality.*
- *Describe the choice of communication protocol.*
- *Describe the implementation of playback synchronization across the network.*
- *Identify the main challenges in implementing synchronized playback on several devices.*

This thesis is tightly related to the Music Impro App. Nevertheless, it is my intention that the knowledge developed through this thesis should be applicable to other projects centered around development of interconnected musical networks.

1.5. Definitions and delimitations

Meeting the goals set by the Music Impro App might depend on knowledge within fields such as musicology, music therapy, human-computer interaction, digital signal processing and computer science, among others. To not make this study too wide-range, some delimitations were necessary to make. The goal of this thesis is not to discuss music and health or interaction through motion. There exist lots of interesting research on these topics. Rather, I want to explore the technical possibilities for realising a musical network such as the Music Impro App on the iOS platform, and discuss the challenges of implementing the system.

Topics related to the audio engine, such as sound synthesis and design of the instruments, will be left out of this investigation. Processing of motion data from sensors and design of the user interface are topics also not included in this thesis. Narrowing the scope of the thesis has been crucial to be able to provide more depth on the chosen topics.

Some expressions will occur frequently throughout this thesis, and to avoid confusion I will state the meaning I assign to these expressions.

Synchronization (or sync): When I talk about *synchronization* in this thesis, I refer to the synchronization of real-time processes in different places. This might include either synchronization of playback between devices in a network, or synchronization of different time-critical operations within a program.

Session: The word *session* will be used to refer to the shared musical activity of using the Music Impro App.

1.6. Structure

This thesis is positioned in the intersection of research areas such as music technology, software development, interaction design and music therapy. It is not possible, nor intended, to give a comprehensive insight into all these disciplines. To make the thesis manageable for the reader I have put effort in focusing the study on software development. Some additional knowledge within musicology is needed to get an understanding of all aspects in the study.

The thesis is structured in the following way:

Chapter 1 gives an introduction to the background of the study and my motivations behind the work. The *problem space* is described and *research questions* for the study are stated. Also provided are background information on the Music Impro App project and the associated RHYME project.

Chapter 2 is devoted to theory on relevant areas related to the problem space. An overview is given on *mobile operating systems*, *musical communication and musical expressions*, *communication protocols* and *latency*. Further, the framework of *interconnected musical networks* is explained.

Chapter 3 gives an account of methods in relation to software development. The methodology applied and the methods used during this thesis are described.

Chapter 4 provides background information on the previous work on the prototype of the Music Impro App, *MIA 1*, including intended user group and a description of previously conducted user tests.

Chapter 5 begins with a description of shortcomings identified in MIA 1, and continues with the needs and limitations that have been discovered during user testing. Further, previously defined guidelines for the next version are stated, and from this, the requirements to guide the development of MIA 2 are defined.

Chapter 6 presents results from a test on network latency in a wireless network.

Chapter 7 gives an initial discussion on the design and construction of MIA 2 with regards to network topology, communication and roles in the network.

Chapter 8 describes the central elements of the implementation. The implemented functionality of network connections, communication and playback synchronization is described, and result from a measurement on the synchronization accuracy is presented.

Chapter 9 provides a discussion on the result of the implementation, viewed against the research questions and the results from measurements. This is further discussed in relation to theory from chapter 2.

Chapter 10 presents the conclusion and thoughts on future research.

2. THEORY

In this chapter the theory relevant for the thesis will be presented. The theory provides a basis for the implementation stage and the discussion to follow.

2.1. Mobile Operating Systems

A mobile operating system (OS) is the operating system that runs on a smartphone, tablet, PDA or other mobile devices. Todays smartphones combine the features of a personal computer with cellular technology, wireless networking, touchscreen, GPS navigation, camera, music player, and other features. The mobile OS is controlling all these features and provides the user with ways of accessing and interacting with them. Because of the capabilities of these mobile smartphones, some people refer to them as an ubiquitous computing platform (Ballagas, Borchers, Rohs, & Sheridan, 2006).

2.1.1. The major mobile operating systems

The major mobile platforms today are Android, iOS, BlackBerry and Symbian, based on the market share in USA and UK (Lyons, 2012). Nokia´s Symbian OS has seen a huge decrease in sales the last years, and in february 2011 Nokia announced that they would discontinue Symbian and move over to Windows Phone for their mobile phones (Sorrel, 2011). As a consequence of this, I chose to mention Windows Phone in this section, rather than Symbian.

2.1.1.1. Android

Android was originally developed by Android Inc, a company that was bought by Google in 2005. Android is based on Linux, and Google released the OS code as open-source. The first version of the OS, Android 1.0, was released in 2008.

Third-party applications (*apps*) can be added to extend the functionality of the OS. These apps are available through Google´s online store *Google Play* (formerly *Android Market*) or from third-party sites. The estimated number of apps downloaded from the Android Market

exceeded 10 billion as of december 2011 (Bonnington, 2011). Apps are usually written in a specialized version of Java, although other languages are supported as well.

Android has for a long time suffered from large latency values in audio playback. This has caused music app development to lag far behind the development on Apple´s iOS. Android devices usually have at least 20 times the latency of iOS devices, preventing developers from making musical apps that can compete with those available for iOS (Guillot, 2011). Ross Bencina, the creator of the interactive audio software AudioMulch[9], pointed out the latency issue in a post on his website. He refers to a talk at the developer conference Google I/O 2011, where one of the Android Team members, Dave Sparks, comments on a question regarding the audio latency problem on Android. He answers in the following way (transcribed by Ross Bencina):

> "Latency is a big problem. We're working at, hopefully we hope to be able to do something about it with ICS[10]. As we investigated it it's actually a pretty complex problem. There are a number of different places where latency gets introduced. Most of the latency is introduced below Android. Basically it's happening in the drivers or in the chipsets or somewhere in there, and some of these are really obscene amounts like hundreds of milliseconds of latency in the audio path. So, that's something we're going to push on." (Bencina, 2011)

2.1.1.2. iOS

iOS is Apple´s operating system running on their iPhone and iPad, as well as iPod touch. The first version of iOS[11] was unveiled with the iPhone in 2007. iOS is derived from Apple´s Mac OS X[12] and is hence a Unix operating system.

Developers usually write apps in Objective-C, which is an object-oriented C language largely influenced by *Smalltalk* (Apple, 2011e). The iOS platform features a range of powerful frameworks for developing advanced media applications. When it comes to audio on iOS, the

9. http://www.audiomulch.com/
10. Ice Cream Sandwich, version 4.0 of Android
11. At that time the name was simply *iPhone OS*. The name *iOS* was introduced in 2010.
12. iOS and Mac OS X share the Darwin foundation.

low audio latency[13] makes this an ideal platform for developing musical applications (Guillot, 2011).

2.1.1.3. Windows Phone

Microsoft´s Windows Phone was introduced in 2010, replacing the *Windows Mobile* operating system. By introducing Windows Phone, Microsoft shifted the aim towards the consumer market, instead of the enterprise market, which had been the aim of the predecessor. A new user interface design was introduced in this version, called *Metro*, which divides the home screen into squares called *Live Tiles*. These tiles dynamically update and display information in real time, while they also act as links to applications.

2.1.1.4. BlackBerry

Blackberry is designed and developed by *Research In Motion (RIM)*, a canadian telecommunications equipment company. Blackberry devices are known for their ability to exchange email and instant messages with a high level of security through message encryption. Apps are available through the *BlackBerry App World*, which has over 6 million downloads each day (2012).

Developers can write apps using many different languages, including Java, C/C++ and HTML5.

2.1.2. The iOS platform

As this thesis is centered around the iOS platform, I am in this chapter going to give a brief overview of the architecture of the operating system and features and frameworks that are relevant for my problem space.

13. A buffer size of 256 samples gives an audio rendering latency of 5.8 milliseconds.

2.1.2.1. iOS architecture

The iOS operating system is made up of layers that act as a bridge between the application code and the device hardware. The layers offer programming frameworks for developers to use when developing applications to run on top of the underlying hardware. Each layer provides an increasing level of abstraction away from the complexity of directly interacting with the hardware. This abstraction makes it easy to write applications that will function consistently on hardware with different capabilities. The layers in iOS are illustrated in figure 1.

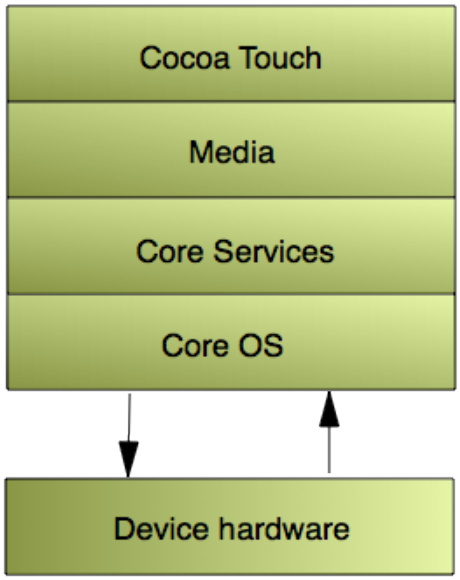

Figure 1: Layers in iOS

Apple suggests that developers should try to use higher-level frameworks over lower-level frameworks if possible, because the abstractions make it easier to write code so that a less amount of code is needed, and it will also encapsulate potentially complex features (Apple, 2011b).

In the following section I will briefly outline the main properties of the different layers.

Cocoa Touch layer

The Cocoa Touch layer contains many of the key frameworks used when building an iOS app. Technologies supported by this layer include storyboards[14], document support, multitasking, printing, data protection, notification services, gesture recognition, file-sharing support, peer-to-peer services, system view controllers and external display support.

Media layer

The media layer contains technologies for using graphics, audio and video in an application. This layer encompasses a big selection of frameworks like: Core Graphics, Core Animation, OpenGL ES, Media Player, AV Foundation, OpenAL, Core Audio, Core MIDI, Core Media, among others. AirPlay[15] is also a part of this layer.

Core Services layer

This layer contains the fundamental system services to be used by all applications. Some of the key technologies in this layer are: iCloud Storage, Automatic Reference Counting (ARC), execution handling, In-App Purchase, and database and XML support. Some of the frameworks in this layer are Core Data, Core Foundation and CFNetwork.

Core OS layer

The Core OS layer provides the lowest-level features that most other frameworks and technologies are built upon. This layer might be necessary to access if you need to "explicitly deal with security or communicating with an external hardware accessory." (Apple, 2011b) Frameworks in this layer include the Accelerate Framework, Core Bluetooth, External Accessory Framework and various security frameworks. This layer also includes the system level and the OS kernel. Certain low-level features of the operating system can be accessed

14. Storyboards is a way to design the application´s user interface by providing an overview of all the views and view controllers and how they are related.

15. AirPlay makes it possible for the application to stream audio to an Apple TV or other AirPlay compatible devices.

through the LibSystem library, like threading, networking (BSD sockets), Bonjour and DNS services, memory allocation and standard I/O.

2.1.2.2. Core MIDI

Because Core MIDI is an essential framework for the implementation of MIA 2, I will provide a description of the important properties of this framework.

Core MIDI is a part of the Media layer in iOS and the framework provides programmers with a way of building MIDI support into applications. This framework was introduced on the iOS platform in version 4.2 of the operating system. Core MIDI provides a set of MIDI system services available to the application. It offers high-performance access to MIDI hardware devices and provides abstractions for communicating with MIDI devices on a MIDI network.

Central to the framework is a MIDI server that handles all MIDI communication and allows for all MIDI devices and endpoints to be shared simultaneously among the applications on the system. The MIDI server also loads any MIDI drivers needed. On the kernel level is IOKit, which handles all the communication over the transport in use (USB, Wireless network, etc). On the top level is the Core MIDI framework that the application links against, and MIDI services for applications are communicated with the Core MIDI server through Mach IPC[16]. A structural overview of Core MIDI is given in figure 2.

To use Core MIDI for communicating MIDI data, the application has to configure certain parameters that will be explained in the following section.

16. Mach: the kernel that Mac OS X and iOS is built on. IPC: Inter-process communication

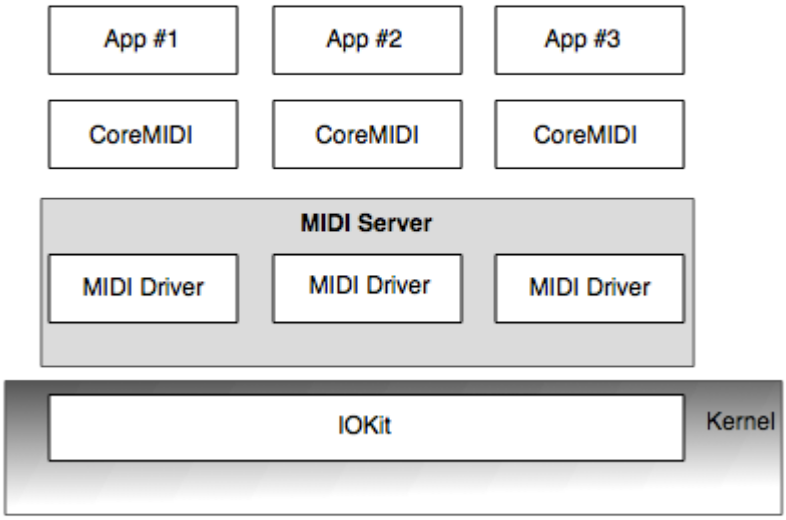

Figure 2: Structure of Core MIDI

The application must create one *MIDIClientRef*, which is used by Core MIDI to refer back to the application. The relationship between client, ports and endpoints in Core MIDI is shown in figure 19.

A client will have only one input and one output port; these are referred to by setting up a *MIDIPortRef* for each port. All MIDI data being received by the client will come through the input port, and likewise, all data being sent from the client will go through the output port.

The *MIDIEndpointRef* is a specific input or output that Core MIDI can send MIDI data to or receive from. This can be for instance a specific USB interface through which data is received, or a virtual port through which an other app running on the same device receives MIDI messages. *MIDIEndpointRef* can be compared to a physical MIDI cable that is able to carry 16 channels of MIDI data in one direction.

Core MIDI also allows MIDI communication over a wireless network. This is called a *MIDI network session* and can be created by using the MIDINetworkSession class in Core MIDI. In Mac OS X, the utility application *Audio MIDI Setup* can be used to access this network session. Clicking the Network icon opens the *MIDI Network Setup* window (shown in figure 3) which gives access to the Mac´s MIDI network session and enables the computer to communicate MIDI data with other MIDI network session-enabled devices on the network.

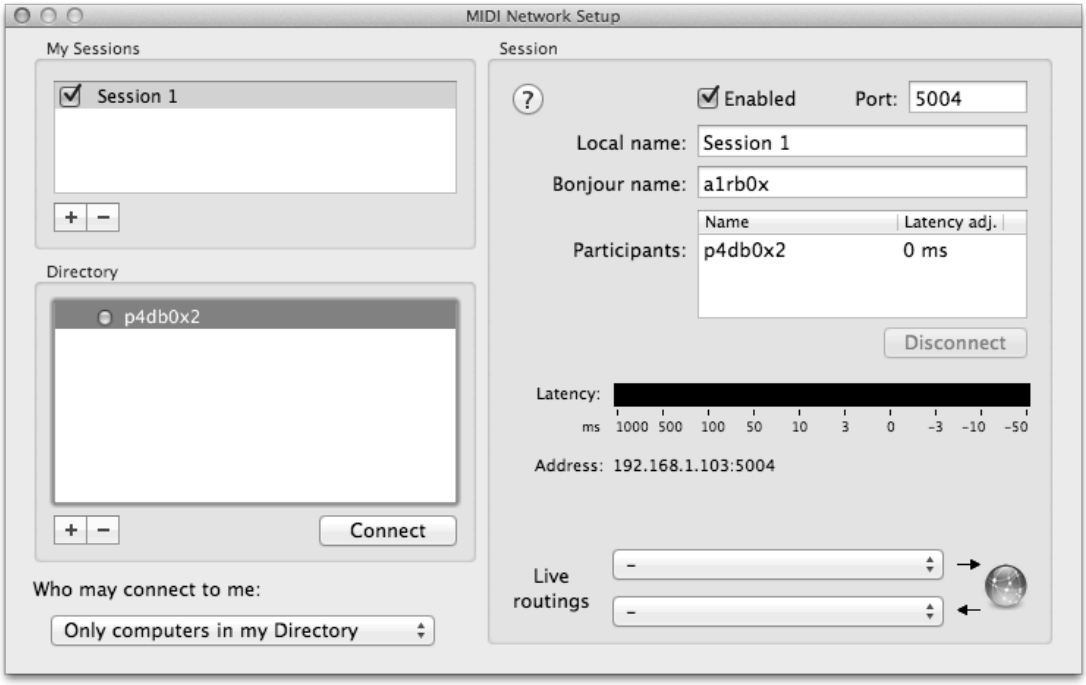

Figure 3: MIDI Network Setup in OS X

2.1.2.3. Core Audio

Core Audio is a collection of frameworks for working with digital audio in Mac OS X or iOS. Any sound played on a Mac or an iOS device is produced by an audio engine based on Core Audio. The frameworks that make up Core Audio can be split into two groups: audio engines, which process streams of audio, and helper APIs, which help getting audio in and out of these engines or working with them in other ways.

The audio engine APIs are:

- Audio Units: a low-level API consisting of small units working on buffers of audio data
- Audio Queues: an abstraction on top of Audio Units that simplifies recording and playing back audio
- OpenAL: an API for creating positional sound, ideally suited for game development

The additional helper APIs are, among others:

- Audio File Services: a framework for reading and writing various audio file formats

- Audio Converter Services: a framework for converting between encoded audio formats and uncompressed audio
- Audio Session Services (iOS specific): a framework that coordinate audio resources on the device

The Audio Unit framework is the lowest-level API in Core Audio. This framework allows programmers to create a chain of audio processing units, often called an *audio processing graph*, or *AUGraph*. In such a graph, output from one unit is connected to the input of the following unit. Core Audio uses the *pull architecture*, which means that when an audio unit needs samples to process, it requests samples from the unit connected to its input. This architecture depends on *callbacks* to request the data, and a programmer can access and modify samples going to an audio unit through a *render callback* function (Adamson & Avila, 2012).

Core Audio is written in C.

2.1.2.4. Networking in iOS

The lowest level networking framework in iOS is the BSD socket library, which is very powerful, and complex, and requires coding in C. Apple decided to introduce a higher level framework to hide some of the complexity of the BSD socket library, and this is called CFNetwork. While this framework also requires coding in C, it has some advantages (run-loop integration). In iOS, anything that starts with "CF" (CoreFoundation) is in C. Classes starting with "NS" (Next Step), on the other hand, is in Objective-C and CocoaTouch. Some components have both "CF" and NS" versions, for instance CFNetService and NSNetService. Usually, the "NS" versions are easier to use since they are on a higher level, but the corresponding "CF" versions offer higher level of complexity and control.

Bonjour

Bonjour is an open protocol for zero configuration networking over IP. It is a standard to simplify the process of configuring services and devices on a local area network that usually doesn´t have address and name servers (DHCP and DNS) to allocate addresses and perform name-to-address translation. The protocol makes it possible for applications or devices to find

each other on the network automatically. It does this by providing a way for applications to tell others which IP address and port they can connect to for communicating. This announcement is called *publishing a service* in Bonjour terminology. Other applications look for services by *browsing*. When an app has found a service that it wants to connect to, it *resolves the service* to find out what IP address and what port to establish a socket connection to.

Bonjour uses a special way of naming services and the transport protocols they are using. The service name identifies the service, and this can be either an existing service type (*ftp*, *http*, *printer*) or a custom service name. The transport protocol can be either *tcp* or *udp*, depending on the transport protocol used by the service. These two put together form the *registration type*, which uses the following format:

```
_ServiceType._TransportProtocolName.
```
[17]

As an example of how the process of publishing, browsing and resolving a service is performed, lets consider a music sharing IP-enabled jukebox. When this device gets plugged into the local network, a couple of things happen. First, the device randomly selects an available IP address and assigns it to itself. Second, it starts up its own Multicast DNS responder and verifies its own host name (in this case *eds-musicbox.local.*). Next, it starts up a music sharing service on TCP port 1010, and then it publishes the service, of type *_music._tcp* under the name *Ed's Party Mix* in the *local.* domain.

To discover a service, an application browses the network by performing a query for matching service type (in this case *_music._tcp*). The Multicast DNS responders on each device receives the request, but only devices that match the service type respond with the record containing the service instance name (*Ed's Party Mix._music._tcp.local.*). This process is illustrated in figure 4.

17. The underscore prefix is used to distinguish registration types from domain names in DNS resource records. (Apple, 2011a)

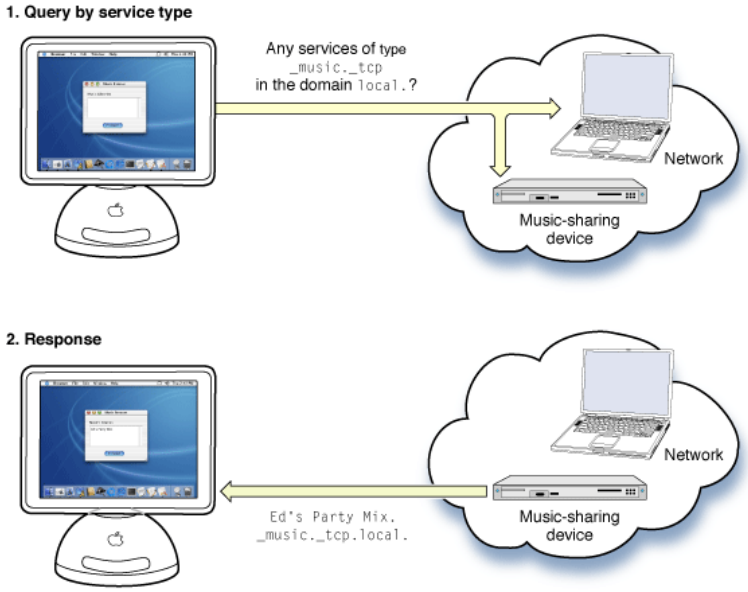

Figure 4: Bonjour: browsing for a service (Apple, 2011a)

Further, resolving a service involves getting the domain name, port number and IP address. This process starts with a DNS query with the instance name received from the discovery process (*Ed´s Party Mix._music._tcp.local.*). The matching device returns the service´s host name and port number. Then the client sends out a request for the IP address, and finally the target device returns its IP address. This process is illustrated in figure 5.

Bonjour can be accessed via the NSNetService and CFNetService APIs in the iOS SDK.

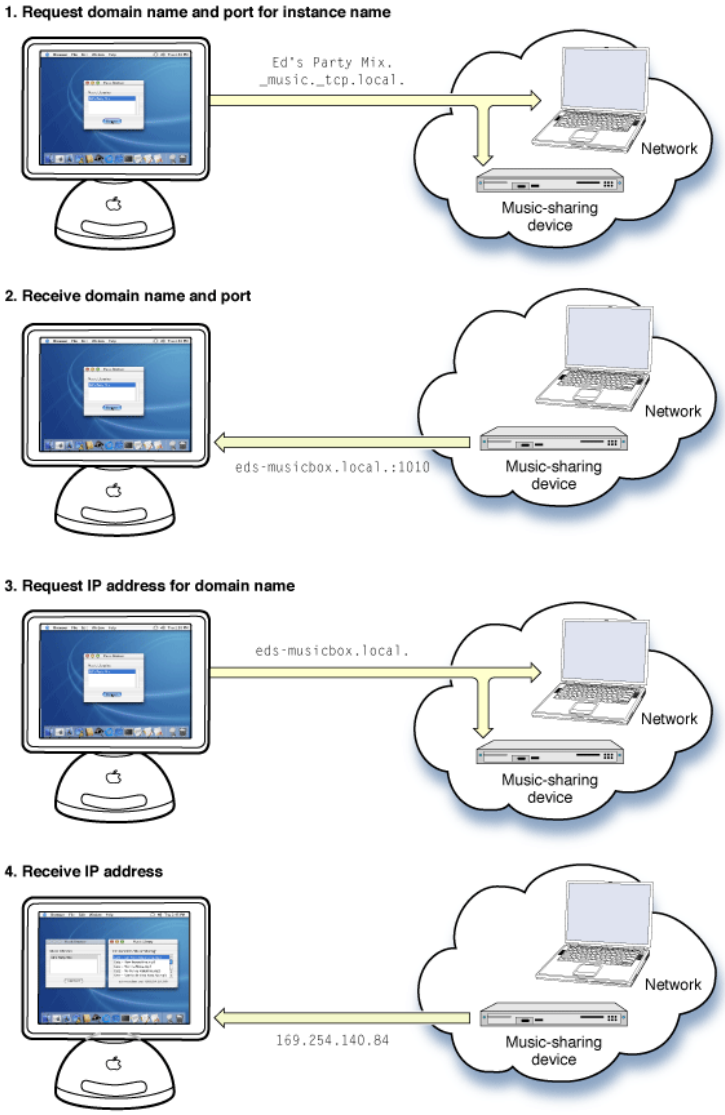

Figure 5: Bonjour: resolving a service(Apple, 2011a)

2.2. Terms in music technology

The work described in this thesis requires an understanding of certain terms related to music technology and music theory. In this chapter I will give a brief explanation of words and terminology used.

Bar (or measure): a segment of time defined by a given number of *beats* according to the time signature of the musical piece. I.e in the most common 4/4 signature, four *beats* make up one *bar*.

Beat: usually a synonym for a rhythm or a drum pattern. A *beat* can also mean a particular division of time in music, the *pulse*. For instance, each quarter note in a bar is often called a beat.

Note length: The note length specifies the relative duration of a note. It indicates how many beats it covers in a measure. For instance, a whole note covers all four beats in a 4/4 measure, and a quarter note covers one of the four beats in the 4/4 measure. Figure 6 illustrates the relationship between note lengths; the notes in each line all fill up one measure.

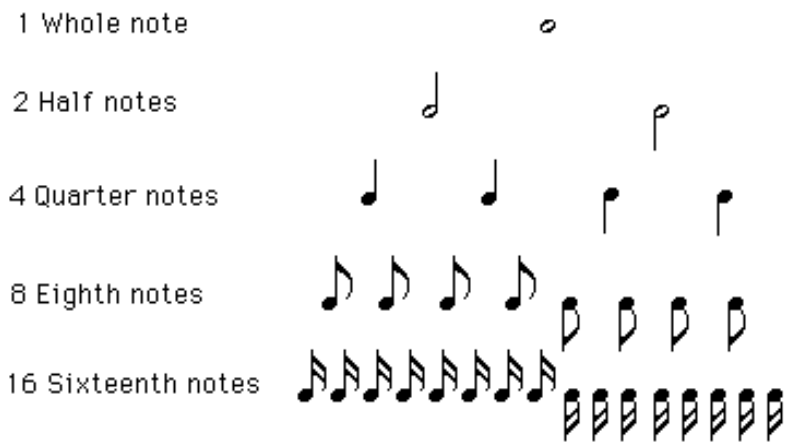

Figure 6: Note lengths

Sequencer: A device (or software application) that can record, edit, or play back music by handling note or performance data in various forms. Sequencers are normally distinguished by the form of data handled; the most common being *MIDI* data or *audio* data. An example of an early software sequencer is shown in figure 7.

PPQ: PPQ (Pulses Per Quarter note) is the smallest unit of time in a sequencer. This determines the resolution of the sequencer, the divisions in time at which events can be placed. Sophisticated hardware sequencers like the AKAI MPC5000 uses a resolution of 960 *ppq* (AKAI, 2012), while a vintage drum machine typically has a *ppq* of 4.

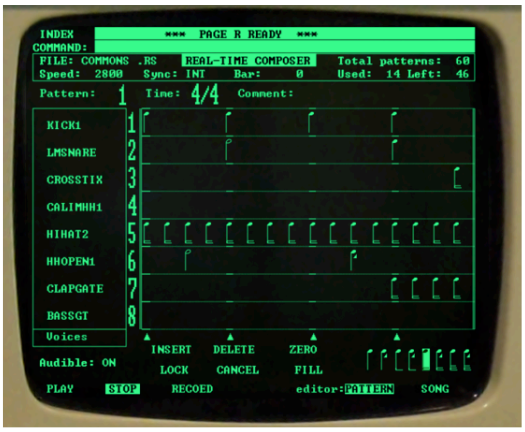

Figure 7: "Fairlight" software sequencer

Sample rate: The term sample rate refers to the number of samples per second that are used to represent a sound digitally. The sample rate for CD-quality audio is 44.100 samples per second (often referred to as Hertz, Hz). Higher sample rates allow for more accurate digital representation of audio, but this comes at the expence of requiring more processing power, as there are more samples to process per second of audio.

2.3. Coordination in musical performance; why and how?

Taking part in certain activities often requires some kind of coordination. Dancing to music usually demands some feeling of, and adaption to, the rhythm of the song. Participating in a marching band requires coordinating the pace of the marching in time with the music, as well as playing the instrument in coordination with the others. Singing a hymn in church involves adjusting to the rest of the community, and the whole community again adapts to the organist playing and leading the song. Social activities involving music thus demands coordination and timing adjustments, probably more than we think of in our daily life. In this chapter I will explain certain aspects of musical communication by looking at situations involving people performing music together.

A musical piece normally consist of a time signature and a tempo[18]. The time signature, stated in the beginning of a musical score, tells the number of beats in a bar and the note value of each beat. This provides the metre that guides the progress of the music and generates the melodic rhythm; the metre is often described as the *timing reference*.

To start with, let´s imagine a duo consisting of a pianist and a violinist playing a musical piece together. The part they are playing requires that they both play together from the very beginning, and that they keep the same tempo. How do they accomplish this? One possible solution is that one of them does a count-in[19]. This signals to both of them the point where the song starts as well as the song´s tempo. When they have started playing, they must adapt to each other´s performance so that they both keep the same tempo, as no other means for

18. The tempo in popular music is normally provided as a number in beats per minute. In classical music the tempo is given as an expression telling the "mood" or "feel" of the piece (Adagio, Andante, Allegro, Vivace, etc)

19. Counting aloud the beats of one bar, usually "one, two, three, four", in the tempo of the song.

indicating the tempo is available. Among more experienced musicians, and especially in classical music, the musicians start without a count-in. This method is based on a common musical understanding of the feel and tempo of the composition, and the start point may be indicated through some bodily gesture like bowing the head.

Moving on to an ensemble of several musicians, it is common to have one person controlling the performance, usually a *conductor*. All the musicians in the ensemble follows the conductor and the musical cues he/she gives with regards to tempo, start/stop, and dynamics. This makes the timing control hierarchic, so that the conductor of an orchestra or the leader of the band provides the timing reference for the rest of the ensemble (Shaffer, 1984). In the case of a band, the drummer will usually fill this role. In a solo performance, the performer is his/her own conductor, controlling the timing reference and all other musical parameters.

2.3.1. The role of MIDI

The MIDI standard (as explained in chapter 2.5.) provides the ability to connect musical devices. This technology has given musicians a way of automating many of the tasks involved in coordinating a musical performance. By communicating musical parameters, MIDI enables coordination between devices like starting, stopping and keeping the same tempo.

2.4. Interconnected musical networks

In this chapter I will outline the history of IMNs and describe a theoretical framework of musical interconnectivity. The term interconnected musical networks has been defined as "live performance systems that allow players to influence, share, and shape each other's music in real-time" (Weinberg, 2003). The description given in this chapter is based on the article *Interconnected Musical Networks: Toward a Theoretical Framework* (Weinberg, 2005).

2.4.1. Historical overview

Some distinct technological innovations can be said to have had an instrumental effect on the development of the field of IMNs:

- analog electronics
- the personal computer
- the Internet
- alternate controllers

These innovations inspired many musicians looking for new ways of expression in social musical settings.

The first electronic interdependent musical network is considered to be John Cage´s 1951 *Imaginary Landscape no 4*, a composition for twelve transistor radios played by twentyfour performers. In the late 1970´s, a group called *The League of Automatic Music Composers* was one of the first to use interconnected personal computers to write interdependend computer compositions. Their networked setup allowed detailed and programmable musical interconnections where parameters from one node were used to control other musical parameters of another node. This group developed into an offspring group in 1986, named *The Hub*, that used the more accurate MIDI protocol for communication in combination with central computers to facilitate the interaction.

Through the invention of the Internet, large-scale IMNs became possible, and this technological advancement provided a reliable way of connecting players spread across huge distances. Different approaches exist when it comes to in what degree the players can interact with each other, and how the computer enhances the interdependent social relations. Taking part in these networks often required special skills and knowledge because of the complexity of the interconnections, and the interactions suffered from the fact that these Internet-based networks were not able to support strict real-time gestural performance cues. In addition, the graphical user interface limited the ability of players to interact with and control the musical performance, as this "graphical user interface cannot replace the personal, unmediated connection provided by tactile interaction with physical instruments in a local space" (Weinberg, 2005).

To remedy these shortcomings and focus on more expressive interconnected musical experiences, researchers simplified the interaction by using physical controllers, sensors, and gestural input, and they created systems in the local space. *Small-scale* local systems thus evolved. Weinberg defines a small-scale collaborative musical network as one that supports three to ten players in close proximity. Examples of such networks are Chris Brown´s *Talking Drum* (Brown, 1999) and the *Squeezables* instrument (Weinberg & Gan, 2001). A *large-scale* musical network, on the other hand, is defined as a system designed for more than ten participants. Examples include Tod Machover´s *Brain Opera*[20], and the work of Feldmeier, Malinowski, and Paradiso (Feldmeier, Malinowski, & Paradiso, 2002)

2.4.2. A theoretical framework of Musical Interconnectivity

From the historical overview and the classification of the local systems mentioned above, the author suggests a taxonomy of musical networks based on theroretical and practical principles of musical interconnectivity. The classification is based on three questions:

- Why? What are the goals and motivations for designing and participating in a musical network?
- How? What are the different social perspectives, architectures and network topologies that can be used to address these goals/motivations?
- What? What are the musical parameters and interdependent algorithms that can be used in the network, filling the form with musical content?

2.4.2.1. Goals and motivations

The motivation behind the musical network can be classified into two major categories based on the relation between the design of and the participation in such a network:

- Process-centered networks, where the emphasis is on the player´s social, creative, or educational experience. The musical outcome is usually of less importance.

20. http://park.org/Events/BrainOpera/

- Structure-centered networks. Here the focus is on the outcome of the interaction, whether it is the music or the performance.

An illustration of the two categories of networks and the corresponding motivations is provided in figure 8.

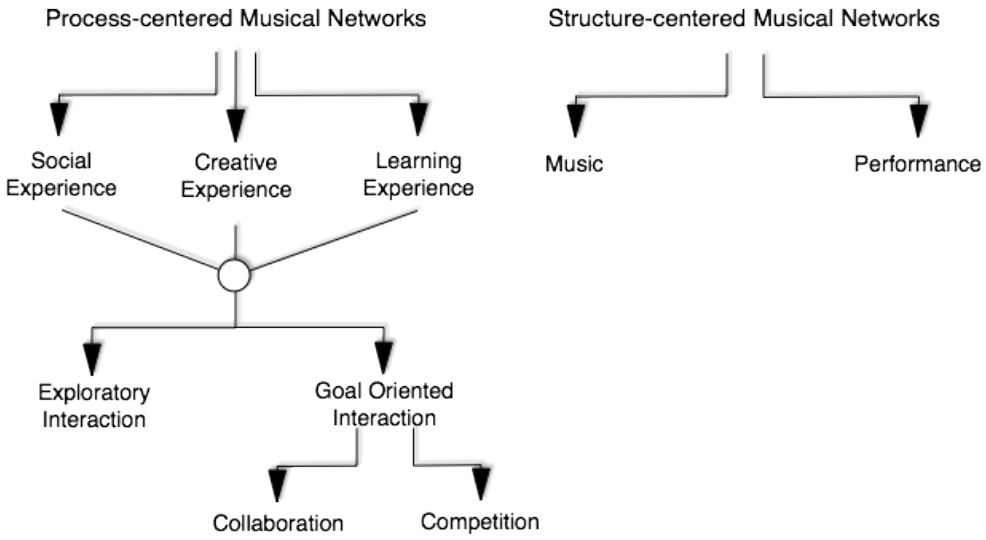

Figure 8: Process- and structure-centered IMNs

2.4.2.2. Social perspectives, architectures and topologies

When it comes to the network´s topology and architecture, a classification into centralized and decentralized networks can be made. A *centralized network* enables the players to interact with a central computerized hub through instruments and controllers, but the players do not have any direct influence on each other. The central hub receives data from the players and analyzes and generates the musical output. In a *decentalized network*, on the other hand, players are able to interact directly with each other through devices that can process and generate audio. These network architectures are illustrated in figure 9.

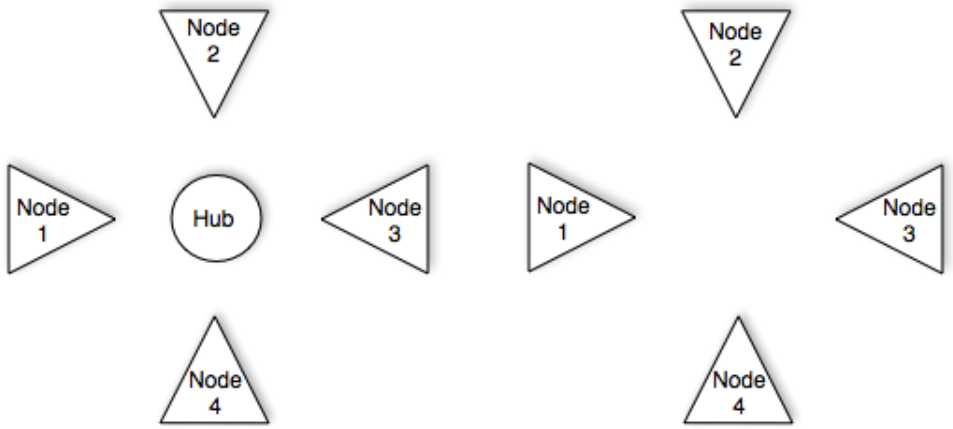

Figure 9: Centralized and decentralized topologies

A range of approaches can be applied under these two categories, distinguished by the musical role of the participants and their level of equality. Networks that provide equal roles to the participants and give them equal opportunities for manipulating the musical content can be classified as a *democratic* network. A system that features one leader (a human or a computer) that directs the other participants´ interaction can be termed as a *monarchic* system.

Centralized and decentralized networks can be further classified into two subcategories, *synchronous* (real-time) and *sequential* (non real-time), depending on the interaction in the network. In synchronous networks, the players can directly manipulate the music of their peers in real-time, while in sequential networks, the musical material of a player is generated with no influence from outside until he/she submits it to the other peers, or to the central hub, for further manipulation.

The classifications described so far do not say anything about the direction of the connections. To provide more detail to these classifications, a layer representing the directionality of the connections between the nodes can be added. To represent a synchronous, centralized network with communication going one way from nodes to the hub, we can extend figure 9 with arrows showing the directionality of the communication. Figure 10 illustrates a centralized topology with data going from the nodes to the hub, and a

decentralized topology where players are connected directly to each other and can interact with each other´s output.

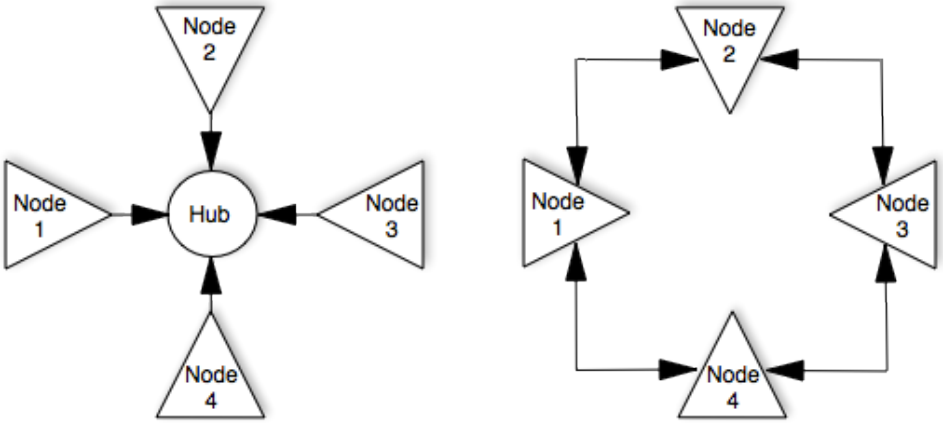

Figure 10: Centralized and decentralized topologies with connections

The topologies presented so far are examples of a symmetric topology. These networks favour equality among the participants; that each player has an equal important social role in the group. However, asymmetric network topologies allow for connections only in specific directions and between specific nodes. An example of such a network can be seen in figure 11, where also the concept of *weighted gates* are introduced. A weighted gate limits the level of influence at a specific intersection in the network. The gate can be open to allow a full level of control and influence, but to restrict the level of functionality the gate can be partly or fully shut. Also, the gates can be manipulated in real time based on input from a player. A number next to the gate indicates the default weight of the gate. An assymetric topology as shown in figure 11 is common in democratic networks, where different levels of control and influence are provided to give different roles to the participants. Taking this assymetric topology to the extreme can lead to a monarchic network where one player has been given full control of the other participants´ interactions by controlling all the weighted gates. An example of such a setup is Golan Levin´s *Dialtones* (Levin,)

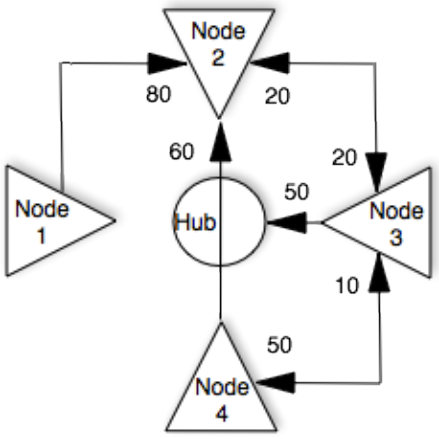

Figure 11: Assymetric, centralized topology with weighted gates

A sequential network can have the same type of interconnections, and a simple example is illustrated in figure 12. This figure shows a symmetric, one-level "stairs" network that connects each node only to the next one, meaning that each node can manipulate the musical output of the previous player. The arrows going forward represent the musical output sent to the next node, and the arrows going backwards represent the manipulation applied to it. More extended versions of this topology can be created by allowing the nodes to interact with not only the previous player´s musical output, but also the output from other players in other nodes of the network, and at different times. The interaction can either be mapped directly between the nodes or via a central hub. Weighted gates can also be added to this topology, similarly to the synchronous networks mentioned earlier.

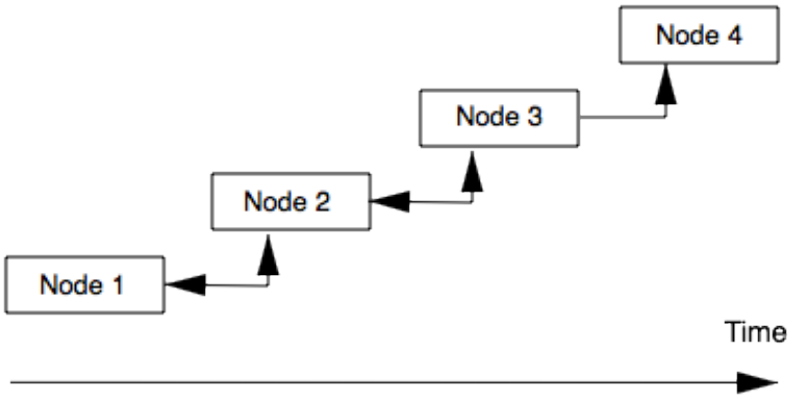

Figure 12: Sequential, symmetric topology

A combination of synchronous and sequential elements is more common in real world musical networks. Typically, each player have full control of their own music before submitting it to the group (sequential interaction), while players can manipulate some parts of the musical output of their peers in real time (synchronous interaction). Adding to this, some parts of the interaction can be centralized through a hub while other parts allow for direct interaction between nodes (decentralized interaction).

2.4.2.3. Interaction and musical coherency

Summing up the article, the author argues that creating an effective musical network is not simply about deciding upon the motivations, social perspectives and choosing the suitable topology for the network. "[...] of great importance for the final musical result are the lower-level decisions regarding the actual musical parameters and transformation algorithms that would infuse such a framework with musical content" (Weinberg, 2005). Choosing which parameters the participants should be able to control in relation to the roles of the other participants is a critical aspect that has to be considered carefully. Giving a player too much control over a key parameter of another peer´s performance might lead to the other player loosing control of his own musical performance. Finding the balance between allowing interaction among participants and keeping the musical coherency is a key challenge when creating such a musical network.

2.5. Communication protocols

Humans need language to be able to communicate through speech. In the same way, devices exchanging digital information are dependant of *protocols*. A protocol defines the rules for sending information from one device to another. Musicians have long searched for ways of connecting their electronic instruments. As these musical instruments have incorporated digital technology, the need for communication protocols has arisen. In this chapter I will describe the most common communication protocols in relation to musical equipment.

In the early days[21] of analog synthesizers, the instruments were comprised of modules that had to be connected in various ways to make sound. The modules were controlled by voltage levels, and the connections allowed the voltage to control the behaviour of the synthesizer, hence their name: *voltage-controlled modular synthesizers*. The connectivity of these modules also made it possible to connect one synthesizer to another to manipulate the sound of the other.

Digital technology made its entry in synthesizers in the 70´s. This quickly led to more advanced features such as the ability to store settings (patches). Synthesizer manufacturers also started to develop interfaces that would enable synthesizers to connect to each other for communication and data exchange. One example is Roland´s DCB[22] interface. Such interfaces were proprietary and didn´t allow synthesizers of different brands to communicate. The first universal standard for connecting digital instruments was introduced in 1983, and it was called MIDI.

2.5.1. MIDI

The MIDI (Musical Instruments Digital Interface) standard was introduced in 1982 as a result of a cooperative effort among several manufacturers (MIDI Manufacturers Association). This new standard provided musicians with an easy and powerful way of connecting synthesizers, drum machines and other musical equipment. MIDI quickly grew to be a standard feature on most digital musical equipment. This new opportunity to connect devices of any brand and easily have them communicate revolutionized the music industry. MIDI was eventually incorporated into computer-based music software and used as communication between the software and external hardware like synthesizers, sound modules and controllers. Even today, MIDI is still the industry standard protocol for communication between musical equipment.

The MIDI standard consists of both a simple hardware standard and a transmission protocol for communication between digital musical instruments and devices. The hardware is an asynchronous serial interface, and the standard connector used is a 5 pin DIN. In the

21. In the 1960´s and 1970´s.

22. Digital Control Bus

following chapters I will explain the transfer protocol, which is the most relevant part for this thesis. But before diving into this, let´s have a look at some basic principles on how musical instruments work.

2.5.1.1. Musical instruments: basics and definitions

All musical instruments make sound. And this sound is controlled by a musician. The musician controlling the instrument can choose when to cause an instrument to start making a sound. A musician can for instance push down a key on a piano to start making a sound, or pluck a string on an electric guitar. This action of starting a sound can be referred to as a *Note on*. On most instruments, the musician can also control when the note stops playing. On the piano, the musician releases the key to stop the note, and on the guitar the finger on the fretboard can be released or the vibrating string muted by a finger. So, we can refer to the action of stopping a note as a *Note off*.

The sound from an instrument can usually be started at different volumes, for example a piano key can be hit with different force. The strenght, or force, of the note being played is usually referred to as *velocity*.

Musicians often want to control several electronic instruments at the same time. It might be that sound from a sound module should be combined with another sound from a synthesizer, and that both sounds should be controlled from one keyboard. This method of playing one musical instrument, and having that instrument controlling one or several other musical instrument(s) is called *remote control*. A device used just to control an other instrument while not generating sound of its own is called a *controller*.

Automatic control is one other way of controlling instruments. This method implies that the musician uses some other device to play an instrument, as if another musician was playing it. This device is typically called a *sequencer* (see chapter 2.2).

From what we have described here, musicians have a need to control instruments remotely and automatically. The MIDI standard allows for this through communication of *MIDI messages*, which will be described in the following chapter. The description is based on MIDI Manufacturers Association´s MIDI tech specs (MIDI Manufacturers Association).

2.5.1.2. MIDI messages

MIDI´s primary function is communicating musical events between two or several devices. This communication is made up of messages that gets sent over a physical connection (MIDI cable, WiFi network). The MIDI standard allows messages that are related to sound generation to be sent on a specific *channel*. These messages are called *Channel Voice Messages*. A total of 16 channels is available on each physical MIDI connection. Devices can be set to receive messages on one specific channel; messages sent on other MIDI channels will be ignored. It is also possible to receive messages on all channels, as if they were received on the same channel. This is called *Omni Mode*. Messages that are not related to sound generation are grouped into *System Common Messages* and *System Real-Time Messages*. Such messages are not related to a specific MID channel but are broadcast to all connected devices.

MIDI messages have a format that consists of a series of bytes, usually 1 to 3 bytes, but some messages can be even longer. Figure 13 shows an example of a 3 byte MIDI message.

Figure 13: MIDI message

Common for all messages is that the first byte is defined as the *Status* byte. This byte has a special signature to distinguish it from the other bytes: it´s the only byte that has bit #7 set to 1, all other bytes will have this byte set to 0. So the start of a MIDI message is always identified by this bit. If we use hexadecimal numbering, a status byte with bit #7 set will be in the range 0x80 - 0xFF. The remaining bytes of the message (if any), the data bytes, is in the range 0x00 - 0x7F. A status byte in the range of 0x80 to 0xEF indicates a Channel Voice Message. This status byte can be broken up into two 4-bit *nibbles*. As in the example given in figure 13 the status byte 0x90 can be broken up into two nibbles: 9 (high nibble), and 0 (low

nibble). The high nibble tells us what type of MIDI message this is. The available types, and the *Voice category* they represent, are:

8 = Note off

9 = Note on

A = Aftertouch

B = Control Change

C = Program (patch) Change

D = Channel pressure

E = Pitch wheel

The low nibble represents the MIDI channel number. In our example, a high nibble of 9 indicates a Note on, and a low nibble of 0 indicates that this Note on happens on channel number 1[23].

The *Control Change* category represents a defined collection of messages that contain a specific *Control Function* parameter, defined by the second byte of the MIDI message. Such messages are used to alter a specific controller on a device usually related to a sound parameter. Examples of Control Functions are *Channel volume*, *Expression controller*, and *Balance*.

Status bytes in the range 0xF0 to 0xFF are for messages that are not locked to a specific channel, and all connected devices will receive these messages. This is used for example to distribute synchronization messages. These bytes are further divided into two groups, 0xF0 - 0xF7 indicating *System Common* messages, and 0xF8 - 0xFF indicating *System Realtime* messages.

System Common messages include MIDI Time Code (MTC) messages, Song Position Pointer and Song Select messages, among others. MIDI Time Code information is used for synchronization of various MIDI and audio/video equipment through a protocol telling the

23. The low nibble represent numbers in the decimal range of 0-15, allowing 16 MIDI channels. Because of the "human" way of counting these channels, a low nibble of 0 actually indicates channel number 1.

time in hours, minutes, seconds and frames, and these messages are transmitted as *System Exclusive* messages. *System Exclusive* (*SysEx*) messages are used to send large amounts of data, such as a dump of the patch memory or some sequencer data. SysEx might also be used to send information that is specific to a manufacturer of a device, as such it might be used to create custom messages to extend the defined standard of messages. A SysEx message starts with a Status byte of 0xF0, followed by any number of data bytes, and terminates with a 0xF7 Status byte.

System Realtime messages consist of only one byte, the status byte. These messages are used for timing and synchronization functions, and because of this, it is critical that the messages are sent and received at specific times without any delay. The MIDI standard allows such messages to be sent at any time, even in the middle of another message, for example a System Realtime message might be sent in between the two data bytes of a Note on message. Devices must be able to handle such situations, processing the System Realtime message as soon as it arrives, and then continue processing the previously interrupted message as if the System Realtime message had never occured.

2.5.1.3. MIDI clock

MIDI clock is a way of synchronizing playback on several MIDI devices. MIDI Time Code, as mentioned in the previous section, is also a way of synchronizing MIDI devices with other equipment. But this type of synchronization is based on time in hours, minutes and seconds, and does not tell anything about the tempo of the song. This protocol is common when syncing the playback of video or tape machines with MIDI equipment[24]. MIDI clock, on the other hand, uses musical beats as a reference, and this gives the connected devices the possibility to determine both the tempo and the playback position in the song. As this way of synchronizing devices is the most appropriate for this thesis I will provide an explanation of this protocol in this chapter, and omit MIDI Time Code for the same reason.

For clock synchronization to work in a midi setup, there should be only one device set to be the *master* clock device and the other midi devices being *slave* clock devices that receive and

24. MIDI Time Code embeds the same timing information as SMPTE timecode, which is a standard developed for synchronization of film, video or audio material (Huntington, 2007).

lock to the master device´s tempo. To begin with, the master device sends a MIDI *start* message that tells all the slave devices to start. Then it sends a stream of MIDI *clock* messages at a rate of 24 messages per quarter note. Through the rate of these messages, the slave devices will be able to tell how the master´s playback advances; when the master clock has sent 24 such messages, the sequencer will have advanced one quarter note worth of the song. The slave devices will in this way be able to tell the average tempo of the song (through the rate of the clock messages received) as well as the song/bar position (through the total number of clock messages received since the start message). When the master clock stops, it sends a MIDI *stop* message telling all the slave devices to stop playing.

If the master clock adjusts it tempo, the rate of the MIDI clock messages will change, and the slave devices will notice this change and adjust their tempo accordingly.

So, to implement a MIDI clock, three messages are neccesary: MIDI start, MIDI clock and MIDI stop. These messages are formatted like this:

- MIDI Start: status byte 0xFA
- MIDI Clock: status byte 0xF8
- MIDI Stop: status byte 0xFC

(MIDI Manufacturers Association)

Compared to other MIDI messages, these messages are very simple, containing only the status byte identifying the message type. MIDI channel number and data bytes are absent, and because of this all devices that are connected to the master device will receive these MIDI messages. Also, a slave device will assume these MIDI clock messages are coming from only one device being the clock master. If two devices are sending MIDI clock messages at the same time there will be no way to distinguish the messages, and the result will be a much higher rate of MIDI clock messages, resulting in the slaves playing at a much higher tempo.

2.5.2. OSC

The MIDI standard is limited in certain ways as a result of the technology of the time the standard was defined. To overcome the limitations of MIDI, a new standard was proposed in

1997: *Open SoundControl*, better known as *OSC*. OSC is a protocol intended for communication among computers, synthesizers and media devices (Wright & Freed, 1997). The protocol was originally developed, and is still a subject of ongoing research, at UC Berkeley Center for New Music and Audio Technology (CNMAT). The OSC protocol features an open-ended, dynamic, URL-style symbolic naming scheme. This allows for human-readable messages that can be created and customized to the user´s needs. Messages can be addressed to a feature of a particular object or a set of objects through this hierarchical namespace. A typical address may look like this:

```
/voices/drone-b/resonators/3/set-Q
```

OSC also features a powerful pattern-matching language to be able to specify multiple recipients of a single message.

One of the major critisisms of MIDI is the limited data resolution as a result of integer representation. OSC offers higher resolution[25] than MIDI, and the data format is flexible. OSC is often used as an alternative to the MIDI standard. Some examples of products supporting the protocol are TouchOSC[26] and EtherSense[27].

2.6. Latency

The term latency is used to describe an elapsed time difference (delay) between two events. A broad definition can be stated like this:

> "Latency of a system is the total time from input stimulus to output response" (Wright, Cassidy, & Zbyszynski, 2004).

In relation to musical equipment, such as an electric piano, this can be understood as the time delay between when a player presses a key and the moment the sound comes out of the speaker. When it comes to computer networks, the term latency is often used to describe the time it takes for a packet of data to be sent from one point to another across the network.

---

25. An OSC argument can be up to 64 bit floating point or integer, MIDI is limited to 7 bit data.
26. http://hexler.net/software/touchosc A control surface for iPhone/iPad that supports OSC and MIDI
27. http://recherche.ircam.fr/equipes/temps-reel/movement/hardware/index.htm

Variations in latency occuring over time is called *jitter*.

2.6.1. Latency in computer networks

There are many factors that influence the latency of a computer network:

- Transport medium and distance. The time it takes for a signal to travel from sender to receiver[28]. Light travels through vacuum at a speed of approximately 300.000 km per second. Signals in fiber or copper cables travel at about 70% of the speed of light.
- Application delay: the time it takes for the application to process incoming data, or prepare data for sending.
- Delay in routers and switches
- Serialization, i.e. the encapsulation of data.

When we consider these different factors that affect network latency, a more specific definition of latency in a computer network is:

> "network latency of packets is the delay from the time of the start of packet transmission at the sender host to the time of the end of packet reception at the receiver host." (Kay, 2009)

The most common method to measure latency in a computer network is by using *Ping*[29]. Ping measures the *round-trip time* of a packet, which indicates the delay between the sending of a ping request packet and the receipt of the corresponding response packet.

2.6.2. Latency in musical performance

In musical performance, latency occurs in various forms. Some musical instruments may be constructed in such a way that a noticeable amount of time occurs from the musician performs a gesture on the instrument to play a tone to the actual sound is produced by the instrument. Another example of latency is a result of the speed of sound through air, which is approximately 344 meters per second. When two musicians playing are placed 5 meters

28. Often referred to as *propagation delay.*
29. Ping is a standard utility in most operating systems used to troubleshoot network connections.

apart, the time it takes for the sound of each musician´s instrument to travel to the other musician can be calculated using the following formula:

```
distance in meters / 344 meters pr second * 1000 ms = time in ms
```

In the above example the calculation will look like this:

```
5 meters / 344 meters pr second * 1000 ms = 14.53 ms
```

In a symphonic orchestra performing on a big stage, the distance between players could be up to 21 meters (Dammerud, 2011). This leads to a maximum delay of 61 ms.

Latency also occurs in computing devices running software instruments. Such an environment introduces latency in many parts involved in the processing of input stimulus and the generation of audio response. If we consider an example of a computing device running an instrument application that responds to movement detected by a sensor, latency might be caused by several parts involved in processing the action:

- Latency of data output from the sensor
- Latency of processing the sensor data (smoothing and other signal conditioning)
- Latency of the operating system initiating a response to the sensor data
- Buffering of audio samples in the operating system
- Buffering in audio hardware
- Phase delay in audio hardware (D/A converters)

(Wright et al., 2004).

3. METHOD

This chapter contains a description of the methodologies of research and an explanation of the methods used during the work on this thesis. First I will give an overview of research methodology and paradigms in relation to information technology. Then I explain the nature of my work related to science and research, and argue for distinctions between different fields within the discipline of Information Systems (IS). Finally, I describe the main parts of the work I have conducted during the course of this thesis and explain the methods used.

3.1. Quantitative vs qualitative research

Research methods are commonly structured into two distinctions: *quantitative* research and *qualitative* research.

Quantitative research methods were developed in the natural sciences, where such methods are used to study natural phenomena. This includes methods like laboratory experiments, measurements, survey methods and numerical methods. Quantitative research is closely linked to the positivist paradigm, where reality is considered to exist objectively and independent of the observer. In this way truth can be represented by measurable properties, usually collected from a representative sample, and these properties are considered unaffected by the researcher and his/her instruments. The ultimate objective is to identify dependent and independent variables, eliminate inadequate variables, and so reduce the complexity of the problem in the purpose of testing a theory, or to draw a generalization (Lázaro & Marcos, 2006). Methods originating from the natural sciences have been adopted by the social sciences as methods for doing quantitative research.

Qualitative research methods originates from the social sciences and are used to study social and cultural phenomena, and to assign meanings to these phenomena. In contrast to quantitative research, multiple realities are considered to exist based on a social reality in constant change. Qualitative methods are used by researchers to help get to understand people and their social and cultural context, and the focus is on understanding phenomena

from the participants´ point of view. The number of samples studied are much more limited than when doing quantitative research, often restricted to a few or even just one sample. Examples of qualitative methods are case study, action research and ethnography. Data sources for qualitative research include participant observation, interviews, documents and text, and even the researcher´s impressions and reactions (Myers, 2011).

3.2. Research paradigms

The philosophical assumptions that guide a research process is usually divided into several different categories of research, often referred to as research *paradigms*. These assumptions are related to the underlying epistemology (the study of knowledge and how it can be obtained) that guides a researcher in his/her reasearch process. Several ways of classifying these paradigms have been proposed. (Guba & Lincoln, 1994) suggest dividing the paradigms into four categories: positivism, post-positivism, critical theory, and constructivism. (Orlikowski & Baroudi, 1991) suggest three categories: positivist, interpretive, and critical.

3.2.1. Positivism and post-positivism

The positivist viewpoint assumes that reality is objectively given and can be accessed and described by measurable properties that are independent of the researcher and of his/her instruments. Also, the observer is ideally not influenced by the observed object, as an influence (in either direction) is considered a threat to validity. Such studies are generally aimed at testing theories to strengthen the predictive understanding of a phenomenon. (Myers, 2011) Post-positivism is an evolution of positivism, and contains the view that the whole truth is never fully seizable, but is progressively approached through the research process. The objectivity of positivism is still intact, but this paradigm opens up for research questions involving complex cultural and social phenomena.

3.2.2. Interpretive

Interpretive studies assume that people create and associate their own subjective and intersubjective meanings to the world around them. Researchers doing interpretive research

thus attempt to understand phenomena through accessing the meanings that participants assign to them (Guba & Lincoln, 1994).

3.2.3. Critical/critical theory

The critical paradigm assumes that social reality is historically constituted, and that people´s ability to alter their social and economic circumstances is limited by different forms of social, cultural and political domination. This paradigm is often applied when studying power issues, inequalities in society, etc. Critical research seeks to be emancipatory (Myers, 2011).

3.3. Research in Software Engineering

The discipline of Information Systems (IS) focus on the development, understanding and use of technology to meet the needs of organizations and individuals, and a major part of this discipline is the engineering of software. Several researchers have asked the question: Can software engineering be considered science? (Gregg, Kulkarni, & Vinzé, 2001), (Marcos, 2005). Gregg et al. (2001) question if software engineering represents a scientific method of inquiry. "While construction of innovative systems can be research, systems development by itself is not necessarily research unless a strong theoretical and methodological grounding provides rigor to the effort." (Gregg et al., 2001) Further, they state that guidelines are missing for the process of cunducting software engineering research, and that what constitutes high quality software engineering research isn´t well defined.

So, what is software engineering in relation to science?

Software engineering is a discipline that deals with the creation of new artefacts (programs, models, objects). We can say that software engineering is concerned with the artificial aspect, the creation of objects that not yet exists. Science, on the other hand, pays attention to the natural aspect and the study of existing objects/phenomena. So we can say that a fundamental difference between science and engineering is that science deals with the study of what things are like, while engineering is concerned with "what they should be like in order to make it possible to construct new objects" (Marcos, 2005). However, science is also involved in action, not just knowledge. And similarly, engineering produces not only action and objects,

but also knowledge. We can say that the difference between science and engineering lies in the object of study, and how knowledge and actions are developed.

Moving on to research paradigms in relation to software engineering research, we easily see that the paradigms mentioned earlier don´t quite fit a research problem of an engineering nature.

The two dominant paradigms for IS researchers have been the Positivist/Postpositivist paradigm and the Interpretive/Constructivist paradigm (Gregg et al., 2001). Researchers working in the positivist/postpositivist paradigm often have the perspective that technology is either present or absent. The interpretive/constructivist paradigm is often used to study how technology affects groups or individuals; then the technology is considered to be available. None of these paradigms take into account the process of creating the information system that is under study. "Both these paradigms do not attend to the creation of unique knowledge associated with the development of information systems from their conception to inception" (Gregg et al., 2001). To address the contribution of software system development to scientific knowledge building, the authors introduce the Socio-technologist/Developmentalist paradigm, as seen in table 1.

Basic beliefs	**Positivist/postpositivist**	**Interpretive/ constructivist**	**Socio-technologist/ developmentalist**
Ontology: What is the nature of reality?	One reality; knowable with probability	Multiple socially constructed realities	Known context with multiple socially and technologically created realities
Epistemology: What is the nature of knowledge?	Objectivity is important; researcher manipulates and observes in dispassionate objective manner	Interactive link between researcher and participants; values are made explicit; created findings	Objective/Interactive; Researcher creates the context and incorporates values that are deemed important
Methodology: What is the approach for obtaining the desired knowledge and understanding?	Quantitative (primarily); interventionist; decontextualized	Qualitative (primarily); hermeneutical; dialectical; contextual factors are described	Developmental (primarily); focus on technological augmentations to social and individual factors

Table 1: Research paradigms

Table 1 is originally adapted from Guba and Lincoln (1994), with added category *Socio-technologist/Developmentalist* from Gregg et al. (2001).

3.3.1. Socio-technologist/Developmentalist paradigm

The following explanation is done on the basis of the article by Gregg et al. (2001). The socio-technologist/developmentalist paradigm considers reality to be technologically created. Multiple, socially created realities co-exist and are influenced by the need, acceptance and comfort level of technology. Knowledge is said to be "coded explicitly and is implicitly experienced through the behaviour of the system as interactions take place" (Gregg et al., 2001). The methodology applied in this paradigm is primarily developmental, including phases starting with generation of idea (concept) and design, to initial implementation and/or formal description. Focus is on the technological innovations/extensions that are intended to positively affect individual and/or organizational experiences. The knowledge building in this paradigm occurs in the process of conceptualizing, designing, prototype building and/or by formal proofs and descriptions (Gregg et al., 2001). We can say that a keyword of this paradigm is "creation", and the result of this process needs to be scientifically evaluated by other research paradigms. In this way the different paradigms are intertwined with the socio-technologist/developmentalist paradigm. First, the interpretive paradigm provides researchers with a context to gain familiarity with a new field and identify needs. Results from this paradigm might be new generated concepts and propositions, which later may be tested as hypotheses, or identified technology needs that may be used in the construction of a new system. The positivist/post-positivist paradigm is mostly used for confirmation/falsification of propositions. Experience gained from stages in this paradigm might lead to a need for further examination and interaction with participants or processes, which calls for the interpretive paradigm. It might also lead to re-design and re-specification of the system, which then feeds into the socio-technologist/developmentalist paradigm.

3.4. Conducting the development of MIA 2

This study is an investigation into the development of the MIA 2 application, and the primary nature of the work can be characterised as software engineering. As we have seen in the previous section, such a study naturally falls under the socio-technologist/developmentalist paradigm. However, the study consists of different stages each requiring a specific approach and where different methods are appropriate to obtain knowledge. In the following I will describe the various stages of the study and the methods used to collect information.

3.4.1. Analyzing needs and establishing requirements

Determining the needs and requirements for the product to be developed is an essential step early in the software development process. The needs for MIA 2 has been identified in various ways. First of all, the initial goals behind the Music Impro App are being used as an overall guidance for the planning of the system. The experience gained during the work on MIA 1 has provided knowledge on which improvements the next version should implement. Combined with the technical limitations of MIA 1, this knowledge has been used to identify the needs for MIA 2. Additionally, users´ opinions on the product are brought in to get a deeper insight into potential limitations. User tests were conducted on MIA 1 in an earlier stage, and results from these user tests are used when specifying the requirements for MIA 2. The user tests are described in chapter 4.3. The resulting requirements established in this stage provide a basis for the following stages in the development of MIA 2.

3.4.2. Planning and designing the system

Before the actual implementation of the system can start, the system must be planned and important decisions regarding the structure of the system must be made. I will in this part discuss choices on the functionality and the design of the system. Deciding on the motivation, social organisation and architecture of the network are essential factors affecting the outcome of such a musical network (Weinberg, 2005). I intend to shed light on characteristics of music performance and discuss this against the framework of interconnected musical networks, as

proposed by Weinberg (2005). This discussion is held up to the requirements for MIA 2, and from this the fundamental structure and functionality of the application is defined.

3.4.3. Implementing the system

The process of implementing the system has been performed in iterations. Each iteration included implementing a certain functionality that could be tested to verify its function before moving on to the next iteration. Various tools have been used for testing and debugging, and these tools provide access to different types of information:

- MIDI Network Setup utility in Mac OS X: Has been used to test wireless connectivity of the application and to provide a MIDI connection with a computer to communicate with other software tools. (see figure 3)
- MIDI Monitor[30]: a free software utility for Mac OS X that displays incoming and outgoing MIDI signals. This software was used to check MIDI communication.
- Xcode´s debug window: Allows messages to be printed from the code by using the NSLog() or printf() commands. This proved necessary for viewing details like timestamp information or checking other timing related issues.
- Other MIDI-capable iOS apps, like MoDrum[31] and FunkBox[32]: These are drum machine apps that feature MIDI clock synchronization. Using these apps gave the possibility of testing network connectivity, MIDI communication and synchronization.

One of the intentions behind this thesis has been to contribute to knowledge within software development related to interconnected musical networks. A considerable part of this thesis is therefore used to describe how certain parts of MIA 2 have been implemented.

30. http://www.snoize.com/MIDIMonitor/
31. http://www.finger-pro.com/modrum.html
32. http://syntheticbits.com/funkbox.html

3.4.4. Data collection

During all these mentioned stages, various documents have been used as important sources of knowledge. Documents like developer references, books, programming tutorials and code samples have been studied to gain an understanding of features in frameworks, how to combine elements of different frameworks, and how to solve other specific programming tasks. Personal communication with another developer through email was also used in a case where the solution to a specific problem could not be found elsewhere.

In addition, the need for numeric data also became evident. Two measurements were done to collect data on the properties of a wireless network and on the performance of the implemented synchronization:

- Network latency measurement: A simple measurement of the average latency of packets in a wireless network was conducted. The Ping utility was used to send a series of packets between two computers on a wireless network and to compute the average network latency. The result from this measurement provides insight into the timing properties of a wireless network, and this will act as a backdrop for the implementation and the discussion of the communication between devices.
- Timing performance of the synchronization: This measurement was done to quantify the deviation in timing between two synchronized devices, in order to evaluate the performance of the implemented synchronization in MIA 2. The audio output of two iPods running MIA 2 was connected to a computer. Then the rhythm was started so that both devices were playing in sync, and the audio signals were recorded on the computer. This recording was then analyzed to detect the timing deviations in the drum beats between the two devices.

These are examples of quantitative methods. However, interpreting how these measured properties affect the user of the system, or what this suggest for the implementation, requires a qualitative approach, and calls for the interpretive paradigm.

As can be seen here, this thesis consists of developmental methods in combination with both quantitative and qualitative methods. This is due to the fact that the conducted software development process involves different stages each requiring a specific approach.

4. BACKGROUND ON MIA 1

The initial work on the Music Impro App involved several iterations that eventually ended up as the prototype referred to as MIA 1. In this section I will explain the functionality and structure of MIA 1, provide a brief description of the target group of the system, and describe the user tests that have been conducted on MIA 1. This material forms the backdrop for the development of MIA 2.

4.1. Overview of the system

MIA 1 is developed for the Android operating system and consists of two apps having different roles in the system. Each device running one of the apps has to be connected to the same wireless network. This will form a musical network and initiate a musical session.

The first app, called *Big Mama*, provides control of the overall musical parameters in the system. The user controlling this app can choose from different musical styles and start and stop playback of drum, bass, guitar and keyboard loops that form the backing music. The overall tempo and the volume of the loops can also be altered. Big Mama was developed for a tablet computer[33], and only one instance of this app is needed in the system. The second app, simply called *Music Impro App*, was made to be used as an instrument. A user controlling this app can choose to play an instrument from a collection of six different instruments. Playing an instrument is done by moving the phone in various ways. The instruments are created so that each one uses a specific type of motion for controlling the instrument; one instrument requires the user shaking the phone, another instrument relies on the user tilting the phone to control the pitch of the notes played, and so on. Several instances of the this app are intended to be used at a time.

All the devices communicate with a computer through the wireless network. The sound of the instruments and the backing music is generated by the computer which is running the sound engine. The tablet and the smartphones communicate sensor data and simple messages to the

33. Samsung Galaxy Tab, in our case.

computer. This layout with a central computerized hub is categorized as a *centralized network*. The sound engine is created in Max/MSP[34] and this engine uses the received messages to generate the music. A pair of speakers connected to the computer play the generated sounds. Running just one of the apps without the computer running the sound engine will of course be meaningless. The sound engine needs to be present for the system to produce sound. A schematic representation of the system is given in figure 14.

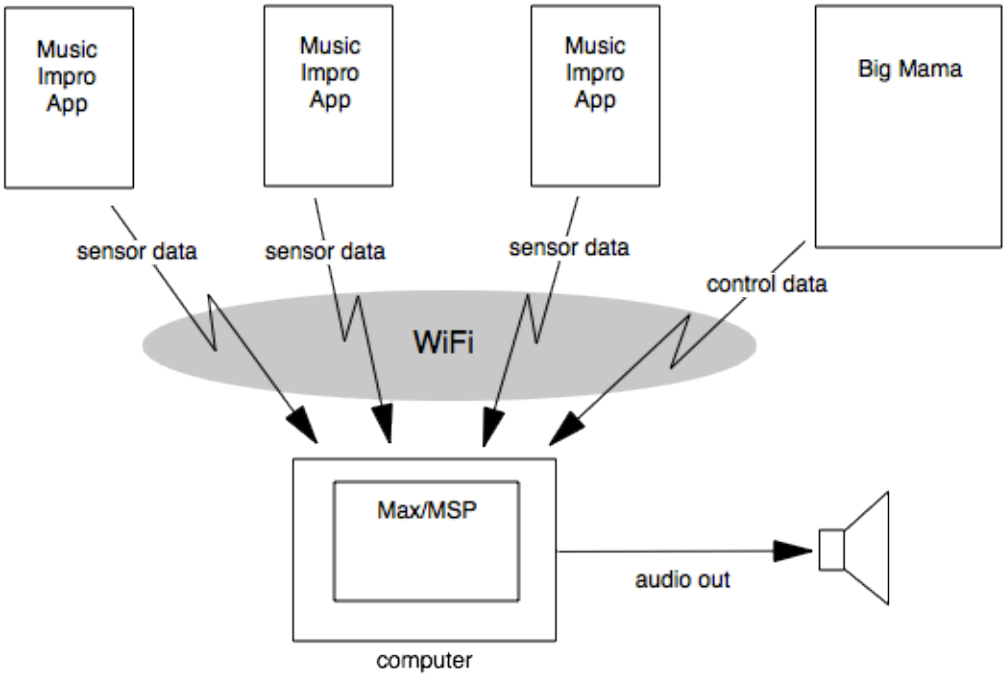

Figure 14: MIA 1 systematic overview

One important element of our system is the ability that instruments can play in time with the rhythm. Some of the instruments generate notes that play a rhythmic figure in time with the backing rhythm, and where the user controls the pitch of the notes by moving the phone. We implemented this feature to make it easier for users to control the instruments and to make the musical output less chaotic and more pleasing. For this feature to work, synchronization between the rhythm and the various instruments is necessary. This synchronization is handled

34. A visual programming language for music and multimedia, created by Cycling ´74. (http://cycling74.com)

by the sound engine since all the instruments and all the sound generating parts of the system are collected in one place.

4.2. Target users

Defining a user group can be valuable for narrowing the scope of a project, but it also might force unwanted limitations on the project before you know your user group well enough. The intention behind MIA 1 has been to reach a wide audience and to also include users that might be restricted from using other commercial products, including children, elderly and disabled people. In this way we wanted our approach towards the user group to be based on the principles of universal design.

In developing MIA 1 we wanted the application to be usable for people with disabilities as well as people without disabilities, for a father as well as his child. Basically, potential users are "only limited to people that can benefit from communicating, experimenting and playing with music, and wish to do so" (Skotterud, Madsen, Sethre, Vestvik, & Bording, 2011).

4.3. User tests on MIA 1

Two user tests were conducted on MIA 1. The goals of the user tests were to see how users responded to using the system in the context of their homes, and to study the effects of having movement as the primary interaction form in a musical interplay situation. Two families were used as target groups. The first family consisted of a mother, a father, and two sons at the age of 7 and 11. The other family consisted of a mother, a father and a 5-year old daughter. The user tests were carried out in the families´ homes.

I have used the most relevant results from these user tests as material when defining the requirements for MIA 2. The test results are presented in chapter 5.2. A thorough description of the user tests and the related findings can be found in the master thesis of Tommy Madsen (Madsen, 2012).

4. BACKGROUND ON MIA 1

5. DEFINING REQUIREMENTS

MIA 1 illustrated our idea of a musical network for musical improvisation and co-creation between several users. Through feedback from user testing, and experience gained during development, this prototype has given valuable insight into how the application can be improved. The development of MIA 2 builds upon these lessons learned from the work on MIA 1.

In this chapter I will first describe the technical shortcomings that have been identified in MIA 1. Then the results from user tests will be described, followed by the predefined guidelines made for MIA 2 before the work on this thesis started. In the concluding section, I will arrive at the requirements for MIA 2.

5.1. Technical shortcomings of MIA 1

The way this prototype is constructed has brought certain limitations on the functionality and the user experience. I will in this section describe the identified limitations that should be addressed in MIA 2.

The most immediate limitation of MIA 1 is the fact that the system relies on a computer to run the sound engine and the relevant processing. Since the intention of the Music Impro App is to utilize mobile terminals without the need for extra hardware, this way of designing the system is far from ideal.

Another limitation of the system´s architecture is that the system is controlled from two different applications with separate functionality. As with the previous point, this distribution of functionality also requires both applications to be present for the musical session to work as intended. This makes the system unnecessarily complex. Additionally, each user´s possibilities are restricted and determined by the type of device the user is controlling. If a user playing an instrument would like to alter the tempo of the whole performance, the user has no ability to do so, other than exchanging the device with the Big Mama, and thus giving away the opportunity to play an instrument.

There are no way users can directly interact with each other, apart from the acoustic interaction while playing. The system doesn´t provide communication between the users´ devices, and in this way the system does not invite to collaboration.

I would also like to add that improvements could be made to the design of the instruments and the mapping of motion to control these instruments. This is a big topic that deserves further research. However, this area is not discussed in my thesis.

5.2. Needs identified from user tests

The user tests conducted on MIA 1 were mainly focused on investigating use of mobile technology in the context of home and the effects of having movement as the interaction form in a musical interplay situation. However, some important findings emerged that are highly relevant for this thesis. These findings are obtained from Madsen (2012), and I will give a summary of the main points in this section.

The limitation of the central computer running the audio engine quickly became evident during the user tests. Because of the sound of all the instruments coming from the same location (the speaker connected to the computer), users reported that it was difficult to distinguish the instrument one was playing from all the other sounds being played. This had a limiting impact on the communication between the participants (Madsen, 2012, page 61).

The lack of dynamic control of the sound level was also an issue that was reported to be limiting. Interplay suffered from the fact that there were no way users could control the intensity of the sound from the instrument. This limitation was observed when two of the users switched to playing acoustic bongo drums. Playing these drums and varying the sound level and the rhythmic intensity led to very dynamic and engaging interplay (Madsen, 2012, page 59).

One user mentioned the importance of responsivity in apps that emulate musical instruments. This user had a musical background and he claimed that if the response from a sound generating gesture is not immediate, the instrument is rather useless (Madsen, 2012, page 66).

5.3. Predefined guidelines for MIA 2

The group behind MIA 1 has made certain overall decisions for the next version. These choices must be catered for, as they act as guidelines for the development of MIA 2. I will in this section briefly state these decisions.

The application will make use of a wireless network (Wi-Fi) to enable the interconnections between devices. Users intending to play together must have their devices connected to the same network. Besides this, no extra device should be needed for the system to operate. It should also be possible to use the app without connecting to anyone else. The app will then act as a solo instrument.

Initially, the users need to connect to each other to create a musical network and initiate a session. Users available for connecting will show up on the graphical user interface as soon as other devices running the app are present on the same wireless network. Once connected, each user can choose an instrument and start playing. When no backing rhythm has been started, the synchronization between the instruments is inactive. The backing rhythm can be started by any user, and the rhythm will also engage the synchronization. The rhythm is distributed to all devices, so that the rhythm will play on all devices simultaneously. The device from where the rhythm is started will distribute synchronization messages to the other connected devices ensuring that the rhythm on all devices are synchronized. In addition, the synchronization will be utilized to provide two types of functionality:

- Some of the instruments will generate notes in a rhythmical pattern, and the user can control the pitch of the notes and the character of the sound by moving the device. This rhythmical pattern is driven by the synchronized rhythm.
- Synchronization messages provide a common musical meter on all devices, and this will be used for *quantization*[35]. Quantization means transforming musical notes, that may occur with imprecise timing, into occuring with precise musical timing that matches the given musical meter. This feature will be implemented on the devices that rely on

35. Quantization is a common feature in MIDI sequencers.

motional gestures to produce sound, to help the timing of notes. It ensures that when playing the instrument, the notes will always play in time with the rhythm.

The backing music introduced in MIA 1 will be slightly simplified for MIA 2 to only include drums. The bass, guitar and keyboard loops are removed.

The application will be developed on Apple´s iOS platform. This choice was made based on the platform´s low audio output latency (Guillot, 2011) and more options for advanced audio programming. And the devices that are intended to run the app will be in the iPhone/iPod touch family, not iPad. This is mostly due to the fact that we want to use motion as the primary mode of interaction, and that tilting, shaking and waving an iPad will be less convenient than using an iPhone/iPod touch in the same way.

The audio engine will be created in the programming language Pure Data (PD). PD is a free graphical programming language for audio synthesis and processing, and this can be embedded into iOS by using a library called *libpd (Brinkmann et al., 2011)*. Libpd makes it possible to write Objective-C code to communicate with a PD patch that can be imported into the project. Various messages can be sent into PD and received from PD through predefined function calls. Libpd handles all the necessary audio setup and configuration of the hardware. The audio engine is out of the scope of this thesis, it is only mentioned here to explain how the code can communicate with the audio engine.

5.3.1. Imagined use

As the application is intended to be further developed in the future, I will describe how we envision possible ways of interaction among users. Having a perspective of features to be added in the future is necessary in the case of MIA 2, as the decisions made during the development must take into account possible extensions of the functionality.

- Modifying another user´s instrument: One user has chosen an instrument to play and while he is playing this instrument, another connected user manipulates the timbre of this instrument.

- Repeating a melody: One user plays a melody on his device. The notes that make up the melody are transmitted to another user for repeated playback on this user´s device and for further modification of the sound or the melody.
- Collaboration between parent and child: The parent controls the pitch of the notes that make up a melody to be played, while the child strikes these notes by shaking his/her device, once for each note. In this way the parent is in control of the notes in the melody, while the child determines when each note is going to play.

Additionally, several ways of interacting should be tested in future user tests.

5.4. Addressing the shortcomings of MIA 1

On the background of the results described in the two previous sections I will in this section establish the requirements for MIA 2.

The structure of the system will be drastically changed in MIA 2 to address the limitations of having a central computer running the audio engine. For MIA 2 the system should not rely on any other devices for the application to operate. The application should handle all the processing and coordination among the devices, and the sound engine must be integrated into the application. This change of structure addresses the limitation of having the audio output from only one location, by making each device in the session play sound through its built-in speakers. In addition to this, the functionality represented by the Big Mama application should also be integrated into the application. In this way, the functionality represented by the various parts of MIA 1 will all be collected into one application, giving all users the same possibilities. This will give every user the possibility of starting and stopping the rhythm, and controlling the tempo. However, once a user has started the rhythm, all the other users should be restricted from interfering with the rhythm until this user has stopped the rhythm. This demands a distribution of roles to be present in the network.

For a musical network to stimulate co-creation and social activity, users should be able to interact with each other´s musical output. Connections among users can lead to an enhanced social experience in musical group playing (Weinberg, 2005). In MIA 2 it should be possible to establish direct connections between devices, and these connections will be used to allow

musical interaction among the users. Communication of musical parameters must be present to enable the interaction. However, the details of how the musical interaction among users will take place are not considered at this stage. This interaction is closely linked to the design of the instruments, which is a topic not discussed in this thesis. Also, future user tests will act as a valuable source of knowledge on which types of interaction to implement.

With this in mind, the implementation of MIA 2 will consist of building an application that can be further developed to support the types of interaction found to be appropriate. This implies that the communication in the system must support many types of musical parameters to facilitate all possible ways of interaction between users. By accurately planning the communication in the network, missing features that are identified at a later stage should be easy to implement by doing simple modifications to the code.

Users should be able to control the intensity, or dynamics, of the sound from the instrument, as well as from the rhythm. This is an effort to address the limited expressivity of MIA 1. Also, users should be given a way of affecting the intensity of the backing rhythm. To accommodate this, a new feature will be introduced in this version, based on the concept of *activity level*. The activity level describes each user´s amount of movement at any given time; when a user barely moves the device the activity level is low, and when he/she makes a lot of movement the activity level is high. The sum of the activity level of all the participants in the session controls the intensity of the backing rhythm. Low overall activity leads to a simple rhythm, and higher activity levels make the rhythm change into a more complex and exciting rhythm. The communication must be able to transmit activity messages.

To sum up, I will list the identified requirements for MIA 2 in brief:

- No central processing unit should be needed. The application must handle all the processing and coordination among the devices, and the application will include its own sound engine.
- The application must be able to connect to other devices running the same application.
- The connections must provide communication of musical parameters to allow interaction among the users.

- The ability to control the backing rhythm must initially be available to everyone. But when a user has started the rhythm, this user will have exclusive control of the rhythm until the rhythm is stopped. The roles of users must reflect this feature.
- The backing rhythm on all connected devices must be synchronized.
- The backing rhythm should be affected by the total activity of the players in the session, so that when the total activity level is low the rhythm is simple, and when the total activity is higher the rhythm becomes more complex and intense.

Additionally, the required musical parameters that the application must be able to communicate to meet the identified needs, are:

- timing messages (to synchronize playback)
- activity level (to alter the backing rhythm)

To be able to facilitate further development of interactions among users, the following parameters must also be supported by the communication. However, these parameters will not be utilized in MIA 2:

- musical notes
- controller data (to alter a specific parameter of an instrument)

6. RESULT FROM LATENCY TEST

The goal of conducting a latency test was to detect how packets are being delayed when sent across a wireless network. The results will provide a backdrop for further discussion of the communication in the musical network.

This measurement was done using a utility program in Mac OS X called *Network Utility*. This program features various tools to diagnose network related problems, and one of these is the Ping tool.

The measurement was performed by having a portable Mac and a portable pc connected to the same wireless network. From the Mac I ran a sequence of ten *pings* to the IP address of the portable pc. The results are shown in figure 15.

```
PING 192.168.0.105 (192.168.0.105): 56 data bytes
64 bytes from 192.168.0.105: icmp_seq=0 ttl=128 time=6.143 ms
64 bytes from 192.168.0.105: icmp_seq=1 ttl=128 time=4.521 ms
64 bytes from 192.168.0.105: icmp_seq=2 ttl=128 time=4.441 ms
64 bytes from 192.168.0.105: icmp_seq=3 ttl=128 time=4.495 ms
64 bytes from 192.168.0.105: icmp_seq=4 ttl=128 time=4.999 ms
64 bytes from 192.168.0.105: icmp_seq=5 ttl=128 time=6.509 ms
64 bytes from 192.168.0.105: icmp_seq=6 ttl=128 time=4.193 ms
64 bytes from 192.168.0.105: icmp_seq=7 ttl=128 time=4.427 ms
64 bytes from 192.168.0.105: icmp_seq=8 ttl=128 time=1.920 ms
64 bytes from 192.168.0.105: icmp_seq=9 ttl=128 time=4.399 ms

--- 192.168.0.105 ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.920/4.605/6.509/1.168 ms
```

Figure 15: Results from Ping test

Because the Ping test measures round-trip time, these numbers indicate the time it takes for a packet to get to the destination and back to the sender. To get the time for a packet to reach the destination (one-way time) we simply divide the round-trip time by two. Data showing minimum, average and maximum round-trip time values are shown on the last line in the figure. The result shows an average round-trip time of 4.6 ms which corresponds to an

approximate one-way time of 2.3 ms. The maximum round-trip time measured is 6.5 ms and this corresponds to a one-way time of 3.25ms.

7. INITIAL DISCUSSION

In this part I will discuss the overall decisions when it comes to planning and designing the musical network. The choices made here act as guidelines for the implementation stage.

7.1. Interdependent Music Performance

Music performance is a constant, evolving real-time process where the output of the instrument is constantly influenced by the musician´s gestures. And the musician is again constantly influenced by the sound coming from the instrument. The performer playing the instrument is part of a delicate interdependency between input and output, between mechanical and acoustic connection. When playing in a group, this interdependency takes on an extra level, as each performer influences, and gets influenced by, the music which is the sum of all performers playing their instruments. Effects such as changes in dynamics and timing are some of the ways the individual player affects the other players and the performance as a whole. Cognitive scientist and jazz trumpeter William Benzon defines music as

> "a medium through which individual brains are coupled together in shared activity" (Benzon, 2002).

This definition suggests a connection among the participants in a musical session, and this is usually formed by the acoustic interdependency. But being acoustically interconnected does not allow for direct manipulation and control of each players´ musical voices. How can modern technology extend this interconnection among musicians in a musical session?

Weinberg states that "Only by constructing electronic (or mechanical) communication channels among players can participants take an active role in determining and influencing not only their own musical output but also that of their peers." (Weinberg, 2005) As an example, consider a player that plays and controls the pitch of his or her own instrument, while simultaneously manipulating the timbre of an other player´s instrument. This manipulation influences the second player so that he/she probably will modify his/her play

gestures as a result of the new timbre. This in turn might lead to a third player getting influenced by this result and adapting his/her musical performance accordingly. And this will again influence the first player; this constant influence among players forms a reciprocal loop. Such an environment can be likened to a musical "ecosystem" that mutually responds to input from individuals (Weinberg, 2005).

One example of the earliest efforts of connecting mobile phones in a musical performance situation is the Pocket Gamelan project that started in 2003 (Schiemer & Havryliv, 2007). In this project, mobile phones are connected via Bluetooth, and messages can be sent from one of the phones to affect the sound of one or several of the other phones. Kaltenbrunner (2005) believe that "playful mobile music creation needs to be an integral part of the digital lifestyle device of the future". The author suggests that mobile devices acting as instruments could be extended to collaborative musical tools if they would provide communication with each other via Wi-Fi or Bluetooth.

Providing connections among the players could also be used to improve/enable coordination between them. In chapter 2.3. the problem of coordinating a musical performance was explained. One of the challenges was related to several musicians performing together; how do they coordinate the point where they should start playing, and how do they all maintain the same tempo? Having all the performers interconnected would provide a channel for distributing messages to synchronize the performance, so that they all listen to the same timing reference.

7.2. Decisions on the functionality

In the following sections follows a discussion on the key parts that constitute MIA 2. Based on the discussion, important decisions have been made for the design of the system based on the following topics:

- The distribution of roles in the session
- The communication among devices in the network
- The topology of the network

7.2.1. Roles in the network

In chapter 5.4 the need for roles related to the control of the rhythm was explained. The user that has taken the role of controlling the rhythm will have exclusive control until he/she chooses to stop the rhythm. As for the playback synchronization, this device becomes the *master* device and provides the timing reference to all the other devices which thereby will become *slave* devices[36]. The role of controlling the master device can be likened to that of a conductor of an orchestra. The conductor signals the timing reference to the musicians in the orchestra so that all musicians are playing in sync. The goal of having the "conductor" role available to everyone is to not give certain users restrained possibilities compared to the others. This is a contribution towards a democratic distribution of roles.

In the same way as the master device provides a common timing reference for the rhythm playback on all the slave devices, the master should also coordinate the changes in rhythmical intensity initiated by the overall activity level. This makes sense because the master device is directing the rhythmical playback on the slave devices, and the rhythmical intensity is also an important attribute of the rhythm. In this way, each device communicates its individual activity level to the master device, which calculates the total activity in the session and communicates the result to the other devices. This feature is strictly not related to the users´ roles in the network, as the handling of activity level is performed automatically by the application without any interaction from users. It is however bound to the device of the user currently holding the conductor role.

7.2.2. Communication

The choice of communication protocol determines how messages are sent between the connected devices, and also which messages that are possible to send. The communication that should be supported by the system is a central feature when constructing a musical network, especially when the focus is on collaboration and co-creation. This is why I have chosen to first discuss the communication in the network before moving on to the network topology.

---

36. This *master-slave* regime is common when synchronizing playback with MIDI clock.

First in this section I will discuss which types of messages that are needed based on the requirements established in chapter 5. Then I will discuss the choice of transfer protocol for communicating these messages on the network. But first, lets have a look at an example of a session involving four devices running MIA 2. Figure 16 illustrates the communication in the network when the device at the bottom is in control of the rhythm. This device communicates synchronization messages to the others, as well as the calculated total activity level. The other devices communicate their activity level to this device. The messages representing total actitivity level is distributed so that each device´s rhythm will be affected by the total activity level in the session.

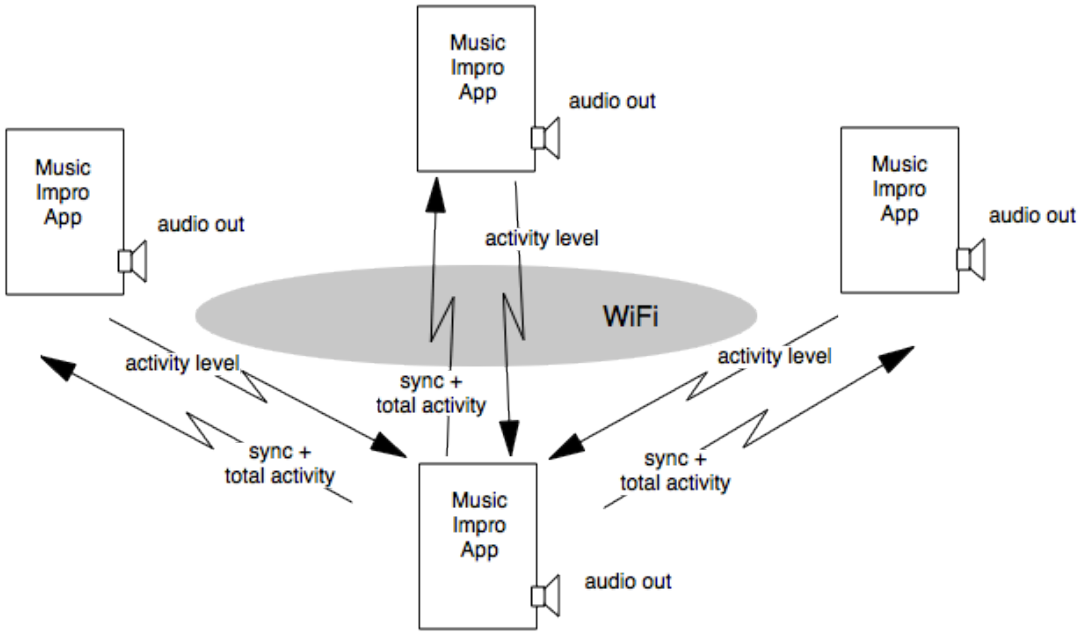

Figure 16: Overview of communication in MIA 2

7.2.2.1. Messages

According to chapter 5.4 messages should be able to transmit many different types of data. For instance, transmitting notes requires data such as note number and velocity. Note number is simply a numbering of the notes in ascending order, like numbering the keys on a piano from the lowest key to the highest key. The velocity value indicates the force the note should be played with. Transmitting controller data requires a value for a certain parameter to be sent

to indicate a change in volume, an instrument´s timbre, a filter´s cutoff frequency, or some other parameter of the sound. Activity level should be represented in the same way as controller data, having a parameter number identifying the activity parameter, and a data field indicating the value of the parameter. In addition to these messages, a clock pulse needs to be transmitted to make the synchronization possible.

Depending on the type of message that is communicated, it should be possible to address messages to a specific recipient. This will be the case when sending note messages from a device to one other specific device, or when sending controller data to alter the timbre of another device´s sound. Situations involving clock messages is not a part of this scenario, as these types of messages are broadcast to everyone. Then it is up to the recipient to take action according to these messages, or not. Recipients must be distinguished in a meaningful way, either by unique names or unique numbers.

7.2.2.2. Communication protocol

The backdrop for the planing and implementation of MIA 2 is the idea that the system should be able to expand as the need for new features are discovered. If we consider the communication protocols mentioned in chapter 2.5, MIDI and OSC, there are some big differences between these two. OSC offers higher data resolution and more flexibility when it comes to data types, and this makes OSC able to communicate a larger range of values. MIDI´s data resolution of 7 bits allows values between 0 and 127. This range might quickly fall short for expressing detailed musical nuances. Nevertheless, as the motivation behind MIA 2 is the collaborative aspect of musical group play rather than the quality of the musical results, the limited expressivity of MIDI is not considered a significant drawback.

OSC offers advanced addressing and messages that can be customized to suit the needs of the application. As such, OSC might be better suited when the system to be constructed is likely to be expanded with new functionality in the future. The MIDI standard features a limited set of defined messages that can be used, and this might limit the possibilities of this standard. But the SysEx messages, mentioned in chapter 2.5.1.2, allows the creation of custom messages. SysEx messages can then be used to overcome this limitation.

The tight integration of MIDI in iOS through the Core MIDI framework makes it hard to consider other communication protocols in the case of the Music Impro App. After all, the MIDI standard has been used for centuries to control musical instruments and to compose and record large compositions, and MIDI is still the dominating protocol in this area. The way that Core MIDI is integrated into iOS also proves to be of great help when implementing synchronisation. This will be explained in detail in chapter 8.3. As a consequence of this, MIDI will be the best suited communication protocol in this case.

7.2.3. Network topology

The topology of a network describes the layout of interconnections among the nodes of the network. This layout will determine in which way interactions among participants are possible, and broadly speaking it will also determine certain characteristics of the network. Deciding the topology is thus a natural starting point when constructing a musical network. Before moving on to the topology of the network, lets sort out what functionality should be facilitated by the network connections.

In chapter 5.4 I summed up with a list of properties that should be supported. Based on the type of messages that should be communicated, the connections should support four types of communication for synchronous interaction:

- Synchronization. This type of communication is used to distribute a common timing reference so that synchronized playback and coordination of tempo and timing is possible.
- Note transfer. To allow a player to play one of the other´s instrument, a communication allowing transfer of note values is needed. This is also useful when recording the notes played onto another device.
- Modification. It should be possible for a player to directly interact with another player´s instrument, modifying a parameter of that instrument. This calls for a connection that can transmit controller data.

- Performance data. Aggregated data regarding the performance, such as the activity level, must be communicated. These data will be used to controll characteristics of the overall performance.

7.2.3.1. Level of centralization

The network should be designed so that it will satisfy the requirements given in chapter 5.4 in the best possible way. The first requirement state that the system should not have to rely on a central unit to coordinate the session. The devices connect directly to each other, without a central hub, and each device contains its own sound engine. This is similar to what Weinberg classifies as a decentralized network topology (Weinberg, 2005). MIA 1 was designed as a centralized network. Making a shift to a decentralized topology is one key choice for MIA 2. Designing the system in this way comes as a result of the fact that each device should play its own sound and contain its own processing capability. Also, this topology is likely to improve the problem users reported in that it was difficult to identify one´s own instrument from all the other sounds in the system.

Before discussing further the choices to be made on the topology of the network, lets step back and consider the overall goal of the system. As mentioned in chapter 1, one of the overall motivations behind the idea that led to the Music Impro App is to create an app for musical interplay that requires no musical skills, that even children and people with disabilities can use, and that make creating and performing music together with others an enjoyable and inspiring activity. The quality of the musical outcome is not the main focus, but rather the interaction and co-creation that occurs during a session. When designing a collaborative interface,

> "[...] the ability to control individual notes, harmonies, melodies and so forth, is not the most important factor to a non-musical person in determining whether or not an interface is engaging. The opportunities for social interaction, communication, and connection with other participants is of paramount importance to the players´ comfort with the interface" (Blaine & Fels, 2003).

This points in the direction of a *process-centered network*, where the focus "is on the player´s experience, whether it is social, creative, or educational" (Weinberg, 2005). Process-centered

networks may also emphasize the interaction taking place among participants, ranging from free, exploratory interaction to goal-oriented interaction like collaboration or competition. If we consider the scenario in chapter 5.3.1 involving two users collaborating on playing a melody, this is an example showing goal-oriented interaction focused on collaboration.

7.2.3.2. Social organisation

When it comes to the roles of the participants in the network, or, the *social organisation* of the network, different choices can be made. Should everyone have equal importance and the same level of control? Or should some users have more control than others and be able to control and restrict the other users ability to interact with others? Weinberg (2005) uses political metaphors to distinguish musical networks based on the levels of equality given to the participants based on their musical role. Systems where all participants have the same opportunities for defining the musical output are defined as *democratic*, whereas a system where a leader controls the other players´ interactions is called *monarchic*.

In MIA 1, the Big Mama application is used to control the backing music and the tempo and style. The rest of the devices do not provide any access to the backing music. This setup is positioned somewhere in between the extremes of *monarchic* and *democratic* networks. The person using the Big Mama is in control of the overall parameters of the musical session, while the rest of the participants are hindered from affecting this aspect. On the other side, no instruments can be controlled from this application, limiting this user´s possibilities. This constitutes the monarchic element of MIA 1. Apart from the one using Big Mama, all the other users are given equal level of control, and this points to the democratic perspective.

For MIA 2 the intention is to provide more equality to the users. A way to contribute in this direction is by minimizing the division of roles, and to make these roles able to change during the session, as discussed in chapter 5.4. The configuration in MIA 1, with functionality separated into two applications, is discarded. Instead, the functionality of these two applications is combined. Controlling the rhythm will in this way be an opportunity available to everyone. In contrast to MIA 1, the user currently in control of the rhythm will not be restricted compared to the rest of the users. Because the sound engine is built into the application, all the available instruments and types of interaction are still available to the user. This means that this user keeps all the possibilities initially provided while getting

preliminary access to an additional layer of exclusive control. So, MIA 2 can be classified as a largely democratic system offering equal level of control to all users, with an additional freely available monarchic element that provides control of the backing music.

We also want to let all participants be able to affect the backing rhythm by the level of motion activity on each device. This is also a contribution towards the democratic organisation of the network. Weinberg (2005) suggests that democratic networks may be more effective when the process is of importance, rather than the musical outcome. This matches well with our intention that the interaction and collaboration among participants is of most importance.

7.2.3.3. Modes of interaction

Musical networks can be grouped into two distinctions depending on how the interaction takes place. In synchronous networks, interactions among participants happens in real-time, whereas in sequential networks each participant first generate their musical material without any direct influence from outside, and then "submit" it to be further developed by the other participants. A synchronous network requires a minimum of network latency for real-time interaction to function as intended, while sequential networks are more tolerant to higher network latency. (Weinberg, 2005) The network latency in the planned network that underpinnes MIA 2 is considered to be low, as the measurement in chapter 6 indicated. We can conclude that the network latency will not be a limiting factor on synchronous interaction.

When we consider the imagined use described in chapter 5.3.1, we easily see the need for synchronous, real-time interaction. For instance, manipulating the sound of another user´s output is an example of real-time, synchronous communication. Playing a melody or some other musical material that gets transmitted to another user is an example of sequential interaction. This type of interaction is illustrated in the example of repeating a melody. In this case the notes of the melody played by the first user will be transmitted to the other user when he/she has finished playing. The System Exclusive category of messages will be appropriate for this, as such messages are "used to send some data that is specific to a MIDI device, such as a dump of its patch memory or sequencer data or waveform data"

(Richardson). Another possible way of solving this may be to record the notes played in real-time. In this case, the use of note messages will be appropriate.

As we can see, the system should support both synchronous interaction and sequential interaction, and MIDI messages provide a way of enabling this interaction. However, further development of the application is needed to implement functionality that utilizes sequential interaction.

7.3. Conclusion on network topology

From the discussion that has been given in this chapter, details on the structure of MIA 2 are starting to emerge. A network of devices running MIA 2 is now used as an example to describe the system. I will provide illustrations to describe the topology of the network and how the interconnections enable communication in the network. The figures are largely based on the terms in the framework created by Weinberg (2005) but I have added some additional features that I considered to be important to clarify the whole functionality of the network. The network is illustrated with four nodes for simplicity.

To start with, figure 17 illustrates the system when the rhytm is not running. Some preliminary connections have been made between the nodes, as shown by the arrows. It should be noted that a connection between two nodes is always bidirectional. This comes as a result of how connections are treated by Core MIDI. A further description of this is provided in chapter 8.1. The weight of the gates is used to illustrate the types of communication that are possible through the connections. These communication types are grouped into four categories:

1) Controlling the rhythm, and consequently, sending sync messages
2) Sending note messages
3) Sending modification messages
4) Sending activity level

In the case when no rhythm is running, only the first three types are available. Activity level is not needed as long as the rhythm is not running. I define the weight of the gate to a value

of 100 when all four types are available, giving each type a weight of 25. In the situation illustrated in figure 17 communication of types 1, 2 and 3 are available to all nodes.

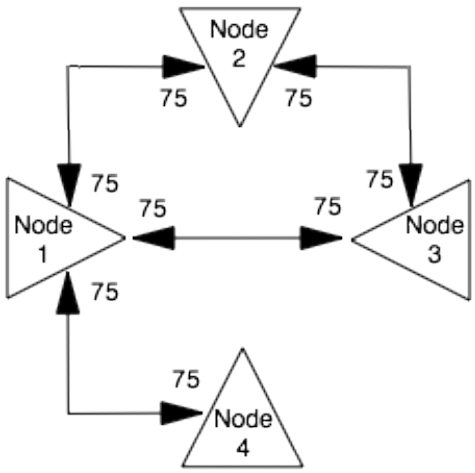

Figure 17: Topology and interconnections in MIA 2 (no rhythm)

As soon as one of the nodes launches the rhythm, the situation changes. The other nodes are immediately restricted from controlling the rhythm, and the node that started the rhythm becomes the master node and starts sending sync messages to all connected nodes. The nodes receiving the sync messages will be sending activity level messages, and the master node will send total activity messages. Now, the master node is able to communicate all four types of messages. The slave nodes are restricted from controlling the rhythm, disabling the first category of communication. The remaining categories 2, 3 and 4 are now available to these nodes for communicating with the master node. It is worth noting that the value of the gate in this case is still 75, the same level as before. However, the ability to control the rhythm has been exchanged with the ability to communicate activity messages. This situation is illustrated in figure 18. In this depiction, node 1 has started the rhythm and thus taken the role as the master node.

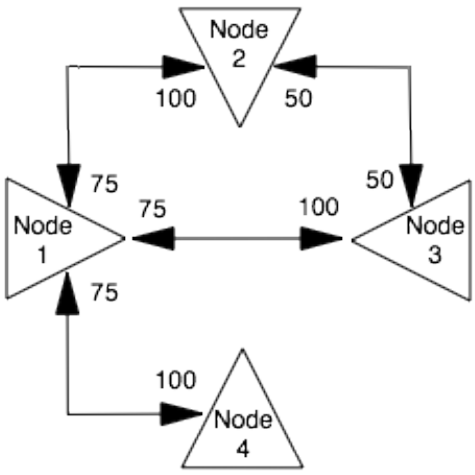

Figure 18: Topology and interconnections in MIA 2 (with rhythm)

Communication between slave nodes, as between node 2 and 3 in the figure, is now limited to a weight of 50. This is the result of activity levels only being communicated to the master node, and not between slave nodes.

8. IMPLEMENTATION

In this chapter I will explain the most important parts of the implementation of MIA 2. To begin with, I will explain the implementation of the networking part of the application. Then, I will describe how the communication in the network is implemented. In this part I will go through the messages that enable interaction between users. The third part is concerned with the playback synchronization, and here I will explain how MIDI clock is implemented in the system.

8.1. Network connections

This chapter concerns the preparation and how the connections between devices are made to form the network that underpinnes MIA 2. The first part explains the configuration of a *MIDI Network Session* to prepare the application for MIDI communication on the network. Then, the process of discovering other devices on the network is explained. How to make connections between devices is described in the last part.

For a developer experienced in using MIDI Network Session and Bonjour, the steps described in this chapter might seem like rudimentary knowledge. But if you haven´t had any experience in using these technologies, figuring out how to implement the networking functionality can be a challenge. The developer references are sparse on this topic, and how to perform certain setup tasks is not explained. Because of the lack of material on this topic, I decided to give a description of how the necessary steps are performed to make the application able to communicate with other devices through the MIDINetworkSession class in combination with Bonjour.

8.1.1. MIDI Network Session

Apple´s Core MIDI Framework Reference in the iOS developer library provides references for classes and services that make up the API of Core MIDI (Apple, 2010a). One of the classes in this reference is the MIDINetworkSession. This is a singleton class that enables

MIDI communication over WiFi and Bluetooth. The session represents one CoreMIDI source-destination pair, known as a MIDI entity (Apple, 2010b). MIDINetworkSession acts as a bridge between the network services and CoreMIDI by providing source and destination enpoints for CoreMIDI to connect to. The endpoints are named from the perspective of the application which means that the source endpoint should be connected to the input port of the client application, acting as a source for incoming MIDI data. This is shown in figure 19.

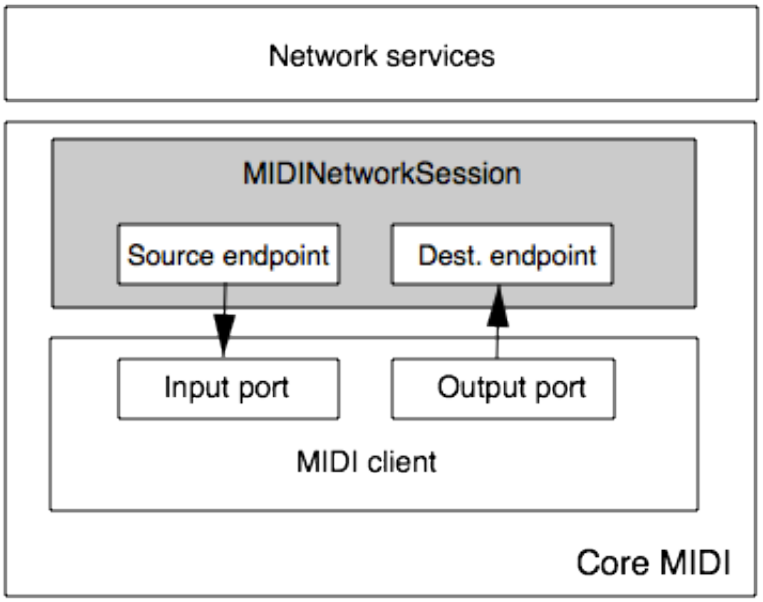

Figure 19: Endpoints and ports in Core MIDI

Using MIDINetworkSession to establish a network MIDI session is quite straightforward. Only three lines of code are needed to enable the network session. First, a pointer to the singleton MIDINetworkSession object must be created. Then, the session must be enabled, and last, the connection policy for the network session must be set. The connection policy defines which types of MIDI network hosts that are allowed to connect to the network session, and this can be set to be either no one, only hosts in the contact list, or anyone. In the case of allowing anyone to connect, the setup can be done like this:

```
MIDINetworkSession *session = [MIDINetworkSession defaultSession];
session.enabled = YES;
session.connectionPolicy = MIDINetworkConnectionPolicy_Anyone;
```

This alone will let the device be visible for other devices that are using the MIDI Network Session, and it will allow others to connect to our app. But further setup is required to be able

to send and receive MIDI data. This involves configuring CoreMIDI by setting up the MIDI client for our application and the input and output ports (and endpoints) mentioned in section 2.1.2.2.

First, the MIDI client must be created by calling CoreMIDI´s MIDIClientCreate(). This function is defined in this way:

```
OSStatus MIDIClientCreate (
    CFStringRef     name,
    MIDINotifyProc  notifyProc,
    void            *notifyRefCon,
    MIDIClientRef   *outClient
);
```

(Apple, 2011c)

The *notifyProc* is an optional callback function through which the client will receive notifications about changes to the system; this can be set to NULL if this is not needed. Then, the *notifyRefCon* is a user data/context pointer that is used by the callback function to access necessary information. In an Objective-C class, this can be set to *self* to access instance variables of that class[37]. The last parameter, *outClient*, is a reference to the MIDI client to be created.

When the MIDI client is created, the next step is to create the input and output ports. This is accomplished by calling MIDIInputPortCreate() and MIDIOutputPortCreate(), with the parameters defined as follows:

```
OSStatus MIDIInputPortCreate (
    MIDIClientRef  client,
    CFStringRef    portName,
    MIDIReadProc   readProc,
    void           *refCon,
    MIDIPortRef    *inPort
);
```

37. When programming Core Audio, the user data pointer is usually a struct.

```
OSStatus MIDIOutputPortCreate (
    MIDIClientRef  client,
    CFStringRef    portName,
    MIDIPortRef    *outPort
);
(Apple, 2011c)
```

These functions both take a reference to the newly created client as the first parameter. Then follows the name of the port to be created. For the input port, *readProc* is a callback function handling incoming MIDI data, with *refCon* as a user info pointer to follow with the callback. This callback function will be further explained in chapter 8.2.1. The last parameter is a reference to the newly created port.

When the input and output ports are configured, the next step is to connect these to the endpoints of the network session. When we consider the case of MIA 2, the MIDI communication will only be through the network, and therefore we only have to configure one source endpoint; the MIDI network session´s source endpoint. References to the enpoint must first be retrieved by calling *MIDIGetSource(index)* with index set to 0. An index of 0 will then retrieve the only endpoint available. Then, the connection between the source endpoint and the input port is performed by calling *MIDIPortConnectSource(inputPort, source, callbackContext)* with the parameters being the created input port, the pointer to the source enpoint, and a pointer to the callback context (*self* is used in this case). Note that the only connection needed is the one between source and input. For the output port and the destination endpoint no connection should be made, as both the port and the endpoint will be provided as parameters when sending MIDI with the *MIDISend()* method (see chapter 8.2.1).

This concludes the initial setup of the MIDI network session. In the following section I will explain how Bonjour provides a way for finding other devices on the network.

8.1.2. Bonjour

MIDINetworkSession alone is not enough to manage connections between devices, because there are no methods to search for and discover other hosts on the network. For this, Bonjour provides delegate methods for handling the discovery of other hosts. I was pointed in this direction through a personal communication with Art Kerns, the writer of the Synthetic

Bits´Core MIDI Brain Dump blog series[38]. As an answer to my question regarding how to initiate a connection to another MIDI Network Session-enabled device, he explained that Bonjour is needed to find other devices, and then Core MIDI is used to make the connection (Art Kerns, personal communication, march 9, 2012). Apple´s documentation on Bonjour states that "Applications can automatically detect services they need or other applications they can interact with, allowing automatic connection, communication, and data exchange, without requiring user intervention." (Apple, 2011a). Through the NSNetServiceBrowser, services on the network can be automatically detected, and the associated delegate methods provide a way to manage contacts that offer these services.

Using Bonjour in conjunction with MIDINetworkSession is not very well documented. No documentation at all could be found on this topic from Apple, but Apple´s sample code project *WiTap* was used to study an example of how to use Bonjour and NSNetServiceBrowser to search for services and to handle when services appear on and disappears from the network (Apple, 2010c). This example used a service of type "_witap._tcp.", where *witap* is a custom name identifying the service type and *tcp* is the transport protocol name. Finding which service type to use when browsing for MIDINetworkSession hosts became evident after studying sample code that used similar setup, specifically Peter Johnson´s MIDIController.m class that is a part of the MIDI setup code in his Molten Drum Machine[39]. This code was published to help other developers sort out the networking setup in CoreMIDI (Johnson, 2011). From this code, the registration type to use when searching for a MIDINetworkSession-enabled host was defined as *MIDINetworkBonjourServiceType*.

Setting up a class to use Bonjour for browsing for MIDI Network session services on the network is done in the following way:

```
browser = [[NSNetServiceBrowser alloc] init];
browser.delegate = self;
[browser searchForServicesOfType:MIDINetworkBonjourServiceType inDomain:@""];
```

38. http://syntheticbits.com/blog/

39. http://www.onereddog.com.au/molten.html

The first line creates a NSNetServiceBrowser object. The second line sets the delegate to *self*, so that messages from the NSNetServiceBrowser will be received by this class[40]. In the last line, a search for the relevant network service is initiated.

When a service is discovered on the network, the delegate method *netServiceBrowser:didFindService:* will be called. In this method, the host providing this service can be resolved by calling *resolveWithTimeout:* A successful resolve will result in the delegate method *netServiceDidResolveAddress:* being called, and then a *MIDINetworkHost* object that represents the resolved host can be created. Hosts available on the network must be added to the network session´s *contacts* list by calling the *addContact:* method. This ensures that the added host is available for making a connection.

The code for handling this procedure is shown in appendix A.

8.1.3. Making connections

Making a connection with a host in the contacts list is a two step process. First, a new *MIDINetworkConnection* object is created by calling *connectionWithHost:* with the selected host as a parameter. Then, this connection is passed as a parameter when calling the *addConnection:* function, which links the connection with the network session.

Disconnecting from a host is done by first getting a reference to the connection from the connections list of the network session, and then calling *removeConnection:* by passing the referenced connection as an argument.

8.2. Communication

The communication of clock messages is a special case of the communication, and this topic is described in chapter 8.3. In this chapter the fundamentals of sending and receiving MIDI messages in MIA 2 is described.

40. In order to respond to the delegate methods, the class must implement the *NSNetServiceDelegate* and the *NSNetServiceBrowserDelegate* protocols.

8.2.1. Sending and receiving MIDI messages

All the information that is communicated between devices running MIA 2 are transmitted in the form of MIDI messages. A MIDI message is structured like the illustration in figure 13. As mentioned earlier in the thesis, various types of messages exist.

In Core MIDI, MIDI messages are sent in the form of a *MIDIPacket* data structure, defined in the following way:

```
struct MIDIPacket {
    MIDITimeStamp timeStamp;
    UInt16        length;
    Byte          data[256];
};
```

(Apple, 2011c)

This structure consists of a *timeStamp* variable, *length* indicating the number of data bytes, and the actual *data* bytes. The timeStamp is used to tell Core MIDI the exact time the packet is to be sent. This is useful when scheduling packets for future delivery, as will be discussed in chapter 8.3. When a packet is to be sent immediately, the timestamp must be set to 0.

MIDIPackets are sent in the form of a *MIDIPacketList*, which may contain a collection of MIDIPackets to be sent to one endpoint. MIDIPackets are added to the MIDIPacketList through the *MIDIPacketListAdd()* function call. The resulting MIDIPacketList is sent by calling the *MIDISend()* method, which is defined like this:

```
OSStatus MIDISend (
   MIDIPortRef           port,
   MIDIEndpointRef       dest,
   const MIDIPacketList  *pktlist
);
```

(Apple, 2011c)

When sending MIDI packets, both the output port to send the data from and the endpoint destination to receive the data must be specified. The endpoint to use in the case of MIA 2 is the MIDINetworkSession´s destination enpoint (see figure 19).

Incoming MIDI messages must be handled in the *MIDIReadProc* that was defined when the input port of the MIDI client was created. This is a callback function that will be called

whenever a MIDI packet is received by Core MIDI. In this callback function the type of message is determined by checking the message´s status byte and then an associated function is called to take the necessary action depending on the received message.

The types of messages used in MIA 2 are:

- clock messages
- individual activity level
- total activity level in the session

Clock messages will be described in chapter 8.3.

8.2.1.1. Individual activity level

Individual activity level is simply a representation of the momentary intensity of motion activity on the device. This value is calculated based on data from the motion sensors on the device, and a low pass filter is applied to smooth abrupt changes. The processed activity level is communicated to the current master device so that the master device can calculate the total activity level in the session. A *Control Change* message has been used to represent the activity level of a device. According to the MIDI standard, no *Control Function* is defined to represent activity level. The *Control Function* that made the closest match was chosen to be the *Expression Controller*, represented by the value 0x0B (MIDI Manufacturers Association). Individual activity level is thus communicated by sending a Control Change message that represents the Expression Controller function. The first two bytes of this message is formatted as follows:

First byte: 0xB0 - indicating Control Change on channel 1

Second byte: 0x0B - indicating Expression Controller

The third byte represents the value of the individual activity level, and this value is within the range of 0 - 127.

8.2.1.2. Total activity level

The individual activity level will be received by the current master device, which is responsible for calculating the total activity in the session and sending these values back to the other devices. This is done in a similar way as the individual activity level, by using a Control Change message. The Control Function used in this case is the *Channel Volume*, represented by the value 0x07. This makes up the following formatting of the first two bytes:

First byte: 0xB0 - indicating Control Change on channel 1

Second byte: 0x07 - indicating Channel Volume

The total activity level is represented by the third byte.

8.3. Playback synchronization

8.3.1. Preliminary discussion

The challenge of having wirelessly connected mobile devices synchronizing their playback is a topic where not much research is to be found. However, one contribution to this area is the work by (Schiemer & Havryliv, 2007). In their Mandala 3 project, a Nokia 7610 mobile phone is connected to three Nokia 6230s via Bluetooth. On initialization, the 7610 "synchronizes each of the three 6230s so that melodic sequences produced by the ensemble sound polyphonically coherent" (Schiemer & Havryliv, 2007). This synchronization process only happens during initialization, as notes on the phones are sequenced by each device´s local clock. These clocks run asynchronously and the authors claim that "There are no discrepancies between the timing of sequences on phones even after a period of twenty minutes" (Schiemer & Havryliv, 2006).

This type of synchronization has some limitations. As the clocks on each device run asynchronously, there are no way for the devices to detect if their clock is running slightly out of sync compared to the others. If the initalization process for some reason causes the startup que for one of the devices to be delayed, there are no way to correct the timing deviation. Another limitation with this approach is that the tempo of the sequences can not be changed while playing. This has to be done before the initialization. When planning the construction of playback synchronization for MIA 2 a similar approach was considered. But as a result of

the limitations with regards to tempo changes and correction of timing deviations, this approach was discarded. In music therapy, one of the ways in which a therapist creates improvisations is by modifying the tempo of the music (Pavlicevic, 2002). A more flexible solution that allows dynamically changing the playback tempo is needed. The best solution to suit these needs is to use MIDI clock.

MIDI clock is widely used, through physical connections like MIDI cables or ethernet, as a method to synchronize playback on MIDI devices like drum machines, sequencers and synthesizers. Because of the way MIDI clock works, where one master device outputs a constant stream of clock messages at a rate corresponding to the musical meter, this method also allows the slave devices to continually adapt to tempo changes initiated by the master device.

The documentation on Core MIDI is sparse, consisting of class references on the various classes that make up the framework. An introduction to the basics of Core MIDI is given in a chapter in the book "*Learning Core Audio: A Hands-On Guide to Audio Programming for Mac and iOS*" by (Adamson & Avila, 2012). The best source for the implementation of MIDI clock with Core MIDI has been a blog series by Synthetic Bits´ Art Kerns (Kerns, 2011). These blog posts provide a detailed description of how Core MIDI can be programmed to perform tasks such as sending and receiving MIDI messages, and generating MIDI clock.

8.3.2. Synchronization in MIA 2

Two main parts are needed to create synchronization based on MIDI clock:

- Generating and sending a stable stream of MIDI clock messages on the master device and receiving them on the slave devices.
- Performing the necessary advancement of the audio engine on both the master and slave devices relative to the clock progression.

8.3.2.1. Generating MIDI clock

According to the MIDI clock standard, the sender of MIDI clock messages should produce a stable stream of clock messages at a rate of 24 per quarter note. If the tempo is set to be 120 BPM, this corresponds to one beat, or quarter note, each 0,5 seconds (60 seconds / 120

BPM). The time interval between each MIDI clock message, lets call it *clock delta*, can then be calculated as follows:

```
clock delta in milliseconds = (60 / BPM) * (1000 / 24)
```

In the example of a tempo set to 120 BPM, the calculation is:

```
clock delta in milliseconds = (60 / 120) * (1000 / 24) = 20,8333 ms
```

So, Core MIDI must send a MIDI clock message each 20,83 milliseconds to generate a stable synchronization clock. MIDI messages are sent as a MIDI packet data structure, as explained in chapter 8.2.1. The timestamp parameter of this data structure is used to tell Core MIDI the exact time the packet is to be sent. This timestamp is a UInt64 data type in host clock time format, the same format as returned by the *mach_absolute_time()*[41] function. Setting the timestamp to 0 will cause the packet to be sent immediately. However, when sending over a network a packet with a timestamp set to 0 will be slightly delayed. This might be due to Core MIDI needing some time to prepare the packet for sending over the network. The key is to use the timestamp parameter to provide a host time into the future, so that packets get scheduled for future delivery. This has been pointed out by several developers (Finger-Pro, 2011), (Kerns, 2011). In this situation, we have a certain time into the future, a buffer time, to add to the current host time retrieved by *mach_absolute_time()*. The amount of time needed might depend on circumstances like the complexity of the application and properties of the wireless network, but in the case of MIA 2 a buffer size of 150 ms was appropriate.

The process of generating the MIDI packets and scheduling them for sending is done by adding MIDI packets to a MIDI packet list, via the function *MIDIPacketListAdd()*, and sending that list to Core MIDI, via the function *MIDISend()*. In MIA 2, six packets are added to the packet list before sending it to Core MIDI. The reason for handling packets six by six is explained in chapter 8.3.2.2. The timestamp for the first of these packets is calculated by adding the buffer to the current host time. Then, the timestamp values for the next five packets are incremented by one clock delta per package. An illustration of this calculation is given in figure 20.

41. Working with host time on iOS is explained in Technical Q&A QA1643 in iOS Developer Library (Apple, 2009).

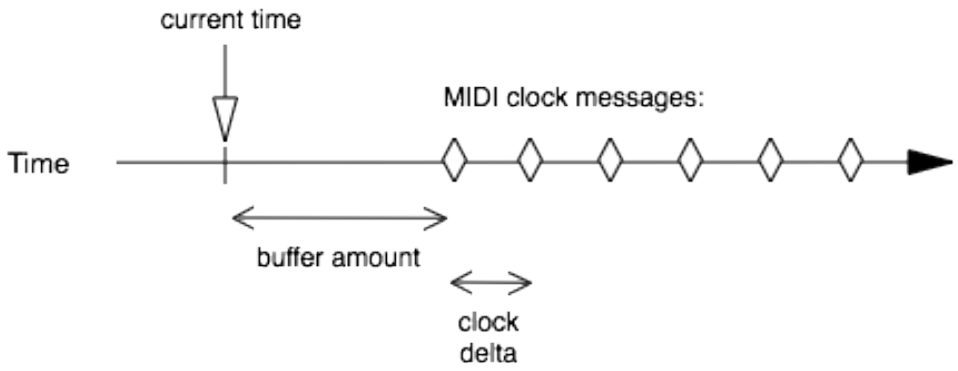

Figure 20: Calculating timestamps of MIDI clock messages

The created packet list is handed to Core MIDI by calling *MIDISend()* with pointers to the output port and the destination endpoint (the MIDI network session). The function for preparing and sending 6 clock messages, based on the current host time, is shown in appendix B.

As MIDI clock messages belong to the *System Real-Time* category they are broadcast to all connected devices. On the devices receiving the clock messages, Core MIDI will handle all incoming MIDI data in the MIDI input callback function declared when setting up the input port, as mentioned in chapter 8.1.1. A MIDI clock message is identified by checking the status byte of the incoming message. When these messages are detected, the next step is to map this synchronization pulse to the audio engine. This leads to the next chapter.

8.3.2.2. Coordination between clock messages and audio engine

As both the master and slave devices incorporate an audio engine that is to synchronize their playback through MIDI clock, the playback of the audio engine must be tied to the stream of clock messages. The relationship between audio engine and clock messages is vital, and to understand this, an understanding of the basic features of the audio engine in MIA 2 is needed.

The rhythm generated by the audio engine is driven by a sequencer. This sequencer is divided into steps, and the drum sounds are triggered on some of these steps. A sequencer´s resolution is often measured in ppq, indicating the number of steps between each quarter note. The

relationship between the resolution of the sequencer and the rate of the clock messages is vital. If the sequencer´s resolution had been exactly the same as the resolution of the MIDI clock ticks, each step forward on the sequencer would generate one clock tick to be sent. And on the receiving side, a received clock tick would simply lead to advancing the sequencer one step. But this is usually not the case. As we remember from chapter 2.5.1, MIDI clock is separated in 24 ticks per quarter note, equivalent to 24 ppq. In MIA 2 the sequencer is divided into four steps per quarter note, corresponding to 4 ppq. This leads to the following situation:

- The sender of MIDI clock must generate 6 clock ticks for each step forward of the sequencer.
- The receiver must advance the sequencer one step for each sixth received clock tick.

The first point is the reason for adding 6 clock messages to a packet list before delivering it to Core MIDI. This means that the operation of preparing these 6 clock messages is performed once per step of the sequencer, making it easier to coordinate these two operations. Determining when this is supposed to happen in time is critical for generating a stable MIDI clock, as well as for running the sequencer precisely in time with the generated clock. We can actually say that the synchronization takes on an extra level, in that the stream of clock messages must be synchronized to the sequencer in the audio engine, both on the sender side and on the receiver side. An overall mechanism for keeping track of time is needed to manage the timing of clock generation and of signalling the sequencer.

On the device receiving MIDI clock, handling the incoming messages is straightforward. The messages are counted, and each time six messages have been received the sequencer is instructed to advance one step. A function performs this processing on a dedicated thread, to ensure that the processing is not delayed by other tasks in the system. This is important as the rate at which these messages will be received is rather high. It is worth noting that the initial step of the sequencer (the first point at the start, the *down-beat*) should occur when the first clock message is received.

When it comes to the master device, where the MIDI clock is generated, the situation is far more complicated. As the MIDI clock packets delivered to Core MIDI contain a timestamp that tells Core MIDI when they should be sent, a similar approach is followed on the master device´s communication with its sequencer. A delay value corresponding to the buffer value

used for the clock messages is used in the messages sent to the sequencer. This causes the sequencer to advance one step when the time indicated by the delay value has elapsed. Both the sequencer and Core MIDI are signalled ahead of time of the event to happen. In this way, the sequencer advances one step at the same time as the first of the six MIDI clock messages are sent. To sum up:

- 6 MIDI clock messages are scheduled for future delivery. The first of these messages will be sent after a time indicated by the applied buffer value (150 ms)
- The sequencer is given a time delay value corresponding to the buffer value (150 ms). After this amount of time has elapsed, the sequencer advances one step.

This situation is illustrated in figure 21.

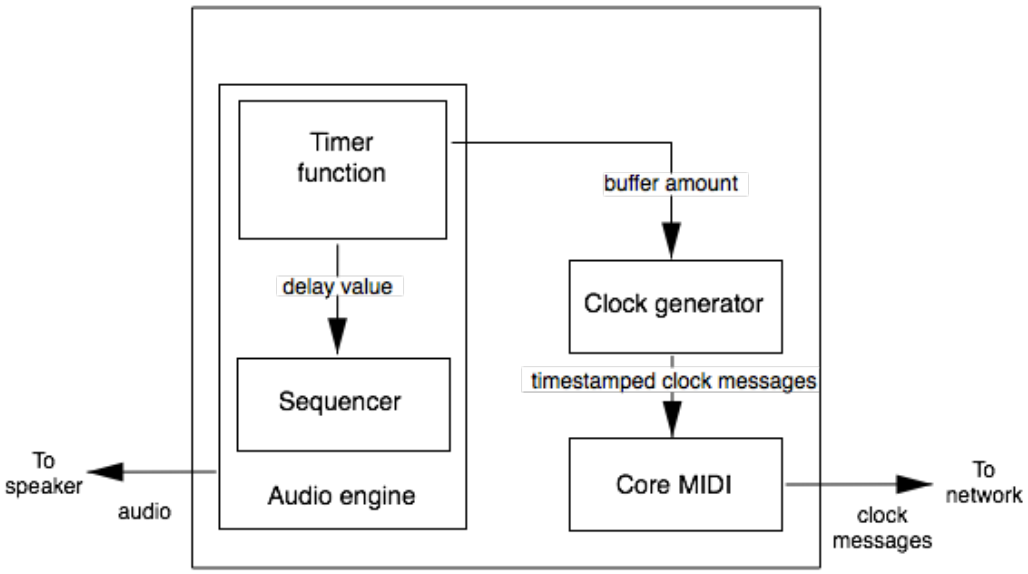

Figure 21: Coordination of sequencer and generated clock messages

The timing mechanism is responsible for triggering the function that generates six new clock messages and the function that advances the sequencer. The time difference between each call to these functions can be calculated by multiplying the clock delta by six, as this will be the time difference between six clock ticks (or one step of the sequencer). In the case of the example used earlier, a tempo of 120 BPM leads to a clock delta of 20.83 ms. Multiplying this value by six gives a value of 125 ms. This means that when time has elapsed exactly 125

ms, the sequencer should advance one step, and six new clock messages should be handed to Core MIDI. Consequently, a precise time-checking function is needed.

A timer object exists in the foundation framework in iOS, namely the *NSTimer*. This might be expected to solve the problem. But according to the NSTimer Class Reference, the effective resolution of the time interval of an NSTimer is limited to on the order of 50-100 ms (Apple, 2011d), and such deviations are not tolerable in this situation.

When coding Core Audio, an audio render callback provides access down to sample level. The callback function gets called every time a connected audio unit needs samples to play (Adamson & Avila, 2012). In MIA 2, the render callback function is not available in our code. The libpd library is responsible for initialising the actual audio engine, and this includes setting up the needed audio units and a render callback. So to get access to the render callback, the libpd code has to be modified.

When libpd initializes its audio engine, a variable called *ticksPerBuffer* is set. This variable determines the buffer size that again influences the rate at which the callback function gets called. I will not go into details on Core Audio and render callbacks, as this is a very complex issue. But it should be pointed out that a small buffer result in shorter time between each call of the callback function, and a larger buffer leads to longer time between the calls. A buffer too small might cause audio dropouts. The lowest value that was useful in the case of MIA 2 was 8 ticksPerBuffer. This resulted in the callback function being called each 23.21 ms. When we consider the time lapse we intend to detect, 125 ms, the rate at which the callback function gets called does not align with this value. To correct this misalignment, the deviation in time is calculated. Figure 22 illustrates this situation.

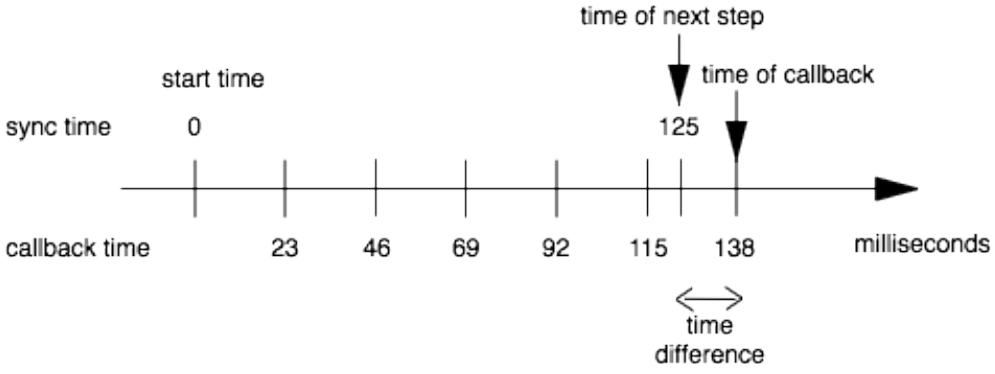

Figure 22: Calculating time difference between audio callback and sync step

A simple comparison is done in the callback function to check the amount of time elapsed since previous call to the clock generating function. If time elapsed is larger than the calculated step time (125 ms in our example), the difference between elapsed time and step time is computed. Then this value is used to adjust the timestamps of the clock messages and the delay value given to the sequencer, to correct for the timing deviations introduced by the render callback function. This procedure is shown in appendix C, which shows an excerpt of the render callback function in libpd and the added code handling the timing of clock .

8.4. Result from playback synchronization measurement

The playback synchronization in MIA 2 has been tested several times by just listening to the sound produced by synchronized devices. This gives an overall impression of how the synchronization performs. A more detailed indication of the performance is obtained by measuring the time differences between events that should occur simultaneously. The goal of conducting this measurement is to detect how precisely the playback is synchronized between two devices, and to use the measured values to compare the synchronization in MIA 2 to similar work by others.

The measurement was conducted by connecting the audio output of two iPod touches to two audio inputs of an audio interface connected to a computer. The audio from the iPods was recorded in the audio recording and production software Ableton Live 8[42]. Both iPods were connected to the same wireless network, and a connection between the two iPods was initiated in MIA 2. Then, the rhythm was started on one of the devices, making this the master device and the other device the slave device. A duration of approximately 6 seconds of playback was recorded by the software. Once recorded, the resulting audio file was analyzed in the same program. By zooming in on the displayed waveform, markers were placed at the start of each recorded drum sound, and the duration in time between two corresponding markers could be found. A screen capture of this process is provided in figure 23. In this figure, *A* points to the start point of a drum sound from the first device, *B* points to the start

42. http://www.ableton.com/live-8

point of the corresponding sound from the second device, and *C* points to a section displaying the duration between these two markers in milliseconds.

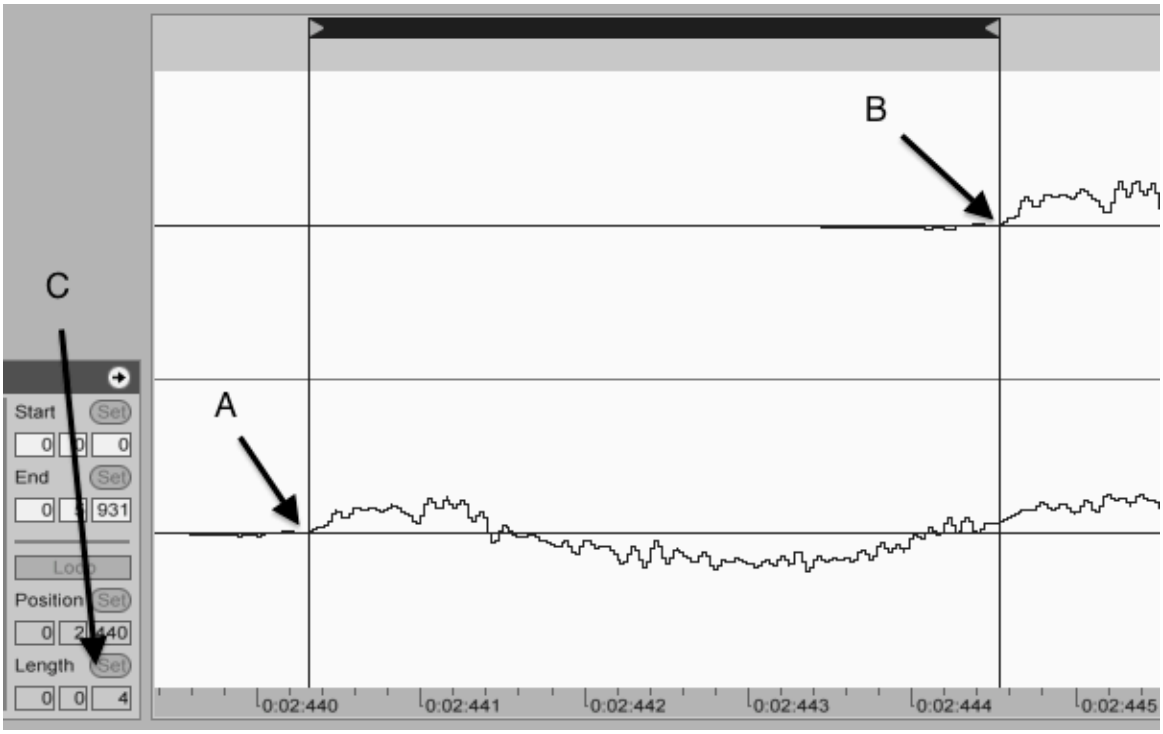

Figure 23: Measuring time deviation in Ableton Live

Results from these measurements revealed a time deviation that varied over time. Even the ordering of the events, which device whose events occured first, was not consistent. Some of the master device´s events occured first, with the slave device´s events slightly delayed from these. But occasional events actually occured first on the slave device. The timing deviation between associated events was measured to be in the range of 4 to 16 milliseconds.

9. DISCUSSION

In this chapter the implementation will be discussed in the light of related work and concepts from the theory. Decisions made during the development will be further argued for, and areas where improvements could be made will be pointed out. Also, the challenges that appeared during the development will be emphasized. Finally, the performance of the playback synchronization will be evaluated.

9.1. Topology and network connections

The choices on the functionality and topology of MIA 2 has been made based on the goal that the application should be easily mastered by anyone. No prior musical knowledge should be needed to be able to produce musical results. An underlying principle has been that of universal design. Not attempting to meet the needs of a specific group, but rather strive for a solution that is accessible for people of all ages and abilities, is a key point of universal design (Story et al., 2011). (Blaine & Fels, 2003) state the importance of restricting the musical control, to make it possible for users to easily learn and participate in the collective experience. To achieve this, the system has been designed to "help" the users play the instruments. Quantization of notes played and generation of notes based on the rhythm are two mechanisms that contribute in this direction. Weinberg (2003) also claims that the musical interaction must be simplified and the possibilities constrained for a musical network to be usable for less experienced users. The author draws a distinction between systems aimed at novice users and systems created for expert users. Systems for novices are likely to suffer under the simplification of the possible interactions, and as a consequence users are prevented from making meaningful and creative contributions to the composition. Expert users might find the system too "shallow", and may quickly loose interest. I see this point as a potential problem of MIA 2, especially since the possibilities for affecting the musical result are limited to manipulating the user´s own instrument or collectively affecting the rhythm through the overall level of activity. This is the reason that I have pointed out that the system should be created to allow extending the features in the future. The MIDI communication

protocol implemented in MIA 2 easily allows adding new types of messages to create additional forms of musical interaction. Creating these new interaction forms and deciding on the musical parameters and how they are modified across the network is of great importance for the final musical result (Weinberg, 2005). Balancing these opposing needs so that novice users don´t get hindered by complexity, and expert users don´t loose interest because of lack of depth, is a challenging endeavor (Blaine & Fels, 2003). As such, this is an area worth of further investigation. User tests on specific user groups will provide insight into possible new interaction forms.

The construction of MIA 2 is done with the intention that it should be a *closed* system, meaning that the application is designed to connect to other instances of the same application only, and not to other applications supporting the MIDI protocol. This can be likened to a multiplayer game app, where users running the same app can connect to each other to participate in the same game. The MIDI protocol was created to allow communication between devices of any brand through a standard protocol. Since the MIDI protocol is implemented in MIA 2, the application speaks a language that other MIDI-enabled applications understand. In this way it may even communicate with MIDI software on a computer[43]. Though this capability has not been emphasized in MIA 2, it makes for the possibility of extending the musical network by allowing other software and hardware devices to interact. For instance, one could imagine using a drum machine in connection with several MIA 2 devices and have the drum machine being synchronized to MIA 2. This would allow manipulating the drum sounds and creating new rhythmical figures to be used in the session. One other possibility is to connect mobile devices running other MIDI-enabled apps. In this way, new and interesting constellations can emerge that provide new kinds of sounds and interactivity. This way of making apps capable of connecting and communicating with other apps through MIDI is pointed out in a document addressing the need for compatibility between musical apps. In this document, called *Open Music App Collaboration Manifesto*, a set of features is listed that apps on iOS should support to be able to work in conjunction with other apps. The features are especially addressed towards using a sequencer application for

43. During initial testing of MIA 2, MIDI communication was tested towards MIDI software on a Mac through the MIDI Network Setup in OS X (see figure 3).

recording and playing back MIDI data from other musical apps. Some of the suggested features are:

- ability to send and receive MIDI
- dynamic discovery of other Core MIDI devices
- a way to map MIDI data to different channels or devices
- support for MIDI control change messages
- MIDI clock support

(Wöhrmann, 2011).

If other apps should be able to interact in a meaningful way with a musical network such as MIA 2, distribution of roles among the devices will have to be determined. In this way, a device acting as input source for user interactions might have the role as a *controller*, a device featuring a sequencer might be designated to be the *timing reference* for other devices, and devices acting as instruments could be given the role *instrument/sound source*. Further modifications of MIA 2 will of course be necessary to facilitate this.

The decision of giving any user the initial ability to start the rhythm was made to contribute towards a democratic distribution of roles in the network. This choice might not be appropriate in the case of a therapeutic situation involving a therapist and one or several patients. Such a situation would require the therapist to direct the overall musical progression, by for instance starting the rhythm. Then, the patients can use the rhythm as a foundation to improvise upon. A different approach to the division of roles might be needed in such a scenario.

9.2. Messages and communication

The communication currently implemented in MIA 2 shows how MIDI messages can be used to communicate musical parameters among devices in the session. One of these musical parameters is the activity level. The devices communicate their individual activity level to the current master device, and the master device communicates the aggregated total activity level back to the other devices. As a result of these messages, the rhythm adapts to the total activity

level in the session. In this way, all participants are given the possibility to collectively affect the music produced in the session. This way of interacting is intended to contribute to the collaborative aspect of the system. Users should feel that their effort makes a contribution to the whole musical output, and not just to the user´s current instrument.

MIDI messages are also used to communicate clock messages to enable synchronization of playback. This feature is fundamental for the functionality of the system, ensuring synchronization of playback and quantization of notes.

The current implementation of MIA 2 realises only these two types of communication, but there are significant opportunities for creating new and more interesting communication. One guideline for the development of MIA 2 was to come up with a system that facilitates many types of direct interaction between participants, as discussed in chapter 5.3. The MIDI protocol supports communication of a vast number of musical parameters, and if a specific parameter is not found in the MIDI standard, SysEx messages can be used to communicate custom parameters. This ensures that the system can be further extended as needs for new functionality are discovered. Implementing a new MIDI message to be communicated in MIA 2 is easily done by setting up the desired message to be sent and handling received messages in the *MIDIReadProc* function.

The need for distinguishing each device in the network was presented in chapter 7.2.2. As for the implementation of MIA 2, this feature has not been implemented. Identifying each device is necessary when messages are to be directed to one specific recipient. In MIA 2, the communication of individual activity level is directed to the master device for calculation of total activity level, and this communication would normally require addressing the messages to this recipient. However, this challenge was solved by making distinctions based on the *master* or *slave* status of each device. The *slave* devices will broadcast their individual activity messages, but only the device being *master* is set to handle these incoming messages. In this way further identification of devices was not needed. Extending the system with additional communication between devices will nonetheless require this feature to be implemented. Dedicating a unique MIDI channel to each device is a possible solution in this case, but assigning the numbers might not be straightforward as no central unit is coordinating the connection of devices. This is a challenge that needs to be solved for future development.

9.3. Synchronization

As has been shown in chapter 8.3, creating synchronization in a networked musical application involves synchronization on two levels; synchronization between the devices in the network and synchronization between the audio engine and the generated clock pulse. On the master device, the audio engine and the generation of clock messages are both controlled by a timer function, and this timer function performs the time-critical coordination between these two. It has been identified that this timer function needs a precise way of measuring elapsed time. This proved to be the most challenging part when implementing clock synchronization in MIA 2. Solving this challenge involved using a timer function that makes use of the audio render callback in the audio engine for calculating elapsed time. However, the time resolution of the callback introduced a deviation in the timer function that had to be calculated and corrected in the resulting function calls. The correction ensured that the deviation introduced by the timer function does not affect the synchronization.

An alternative to tampering with the code in libpd would be to create an audio engine from scratch with Core Audio, and use this instead of libpd. Setting up an audio processing graph with Core Audio is not a trivial excercise[44], but it gives access to the buffer used to store samples. Changing the size of this buffer could lead to an increase in the time resolution of the callback, resulting in smaller time deviation to be corrected by the timer function.

An other alternative would be to create a dedicated timer function. Michael Tyson, developer of the music recording app Loopy[45], has done some experiments with precise timing in iOS to meet his needs for having a timer object when not having access to the render callback. His solution consists of setting up a high-priority thread to schedule events, and then use the system call *mach_wait_until* and a spin lock to minimize time deviation for the scheduled events. The results from these experiments gave an average time discrepancy of just below 3 milliseconds and a maximum time discrepancy of 74 milliseconds (Tyson, 2011). Although this solution gained good results on average, the occasional large time discrepancy makes this solution a poorer alternative compared to using a render callback.

---

44. "*Easy* and *CoreAudio* can´t be used in the same sentence." (Alfke, 2009)
45. http://loopyapp.com/

The reception of clock messages is implemented assuming that the incoming stream of clock packets is as stable as it was generated on the sender side. However, packets may occasionally arrive delayed, due to network traffic or delay caused by the network router. This situation is not handled in the case of MIA 2. Initial tests have shown that the playback synchronization works satisfactorily under most conditions. To address the situation of delayed clock packets, a function predicting the time of the next incoming clock packet could be implemented. Based on the average rate of incoming packets, this function must calculate the assumed arrival of the next packet. Whether this packet arrives on time or not, the predicted value is used to increment the clock counter and, if necessary, advance the sequencer. In this way, the device receiving MIDI clock will have its own timer function that drives the sequencer based on the average rate of incoming clock messages. This implementation will be less susceptible of occasional late packets than the current implementation where the sequencer is directly driven by the incoming clock messages.

9.4. Timing performance in synchronization

The result from the network latency measurement described in chapter 5.6 showed an average one-way latency of 2.3 milliseconds. Conducting a measurement on the timing deviation between audio events on synchronized devices, as described in chapter 8.4, revealed an actual timing deviation ranging from 4 to 16 milliseconds. This result indicates that the network is not the main contributor to the overall latency of MIA 2. Also the inconsistency in these results, in that the events from the slave device did not always occur later in time than the corresponding event from the master device, suggests that some delay is introduced on the devices when handling the clock messages. A source of this delay might be the processing of incoming clock messages on the slave device.

An optimization of the thread processing the MIDI clock messages could possibly improve the handling of clock messages on the recipient side, and consequently result in more consistent timing. The topic of threading is not treated in this thesis.

Another possible contribution to the delay is the preparation of clock messages on the device sending these messages. The timing regime implemented in MIA 2 deals with very accurate time units down to host time level. Time deviations might be introduced from the processing

that occurs between the moment the current host time is retrieved to the necessary action is taken as a result of this retrieved host time. This could happen if the thread performing the mentioned processing got preempted[46] in the middle of the process because some other thread required processing time. Improving this situation also requires optimization of how threads are executed in the system.

These problems aside, how does the measured timing deviations relate to the real world? If we convert the timing deviation to the corresponding distance travelled by sound during that time, the largest deviation of 16 ms corresponds to a distance of 5.5 m. So we can compare this timing deviation to the time it takes for sound to travel between two persons placed 5.5 meters apart. MIA 2 is intended to be used by a small group of people playing music together in the same room. In this situation it is not unlikely that some of the participants might be placed 5.5 meters apart, resulting in such a latency in the audio travelling between the participants. Consequently, the timing deviations caused by the synchronization mechanism in MIA 2 can not be considered to be vital for the functionality of the application.

An example of a related work involving synchronization of networked devices is the Princeton Laptop Orchestra, PLOrk. This system consists of 15 laptops connected through a wireless LAN. With all 15 laptops synchronized, the authors report an approximately measured latency of 30-40 ms or better (Trueman, Cook, Smallwood, & Wang, 2006). Given the size of this orchestra, ranging 40 feet (12.2 meters) in width, the latency corresponds to the acoustic latency that exists in the orchestra. In connection with this, the authors claim that "from a musical performance perspective, the latency feels more-or-less normal, given the familiar separation within larger ensembles" (Trueman et al., 2006). This does not say anything about any jitter in latency values. Given the measurements indicated, the synchronization in MIA 2 performs considerably better than this example.

46. In a multitasking operating system, *preemtion* occurs when a process/thread is suspended to allow another process/thread to run on the CPU.

10. CONCLUSION

This thesis has been centered around the development of a mobile application for musical collaboration. The development originated from previous work on MIA 1, and based on the prototype, a fully functional iOS application for iPhone/iPod touch has been implemented: the MIA 2 application. In this thesis the development of MIA 2 has been described, and challenges related to the implementation have been highlighted. The study has been focused on the network connectivity and communication functionality.

First, the initial decisions on the planning of the networking properties, as well as the choice of communication protocol, have been discussed and argued for based on works within musical collaboration and IMNs. Following this discussion, the topology of a network of interconnected MIA 2 devices was described, and the required functionality of the application when it comes to communication was estabilised. The MIDI protocol was chosen as the communication protocol in the network, largely based on the possibilities of the Core MIDI framework in iOS.

Then, the implementation of the main functionality of the application was described. Implementing the functionality based on Core MIDI posed difficulties as a result of insufficient documentation, especially in the case of using Bonjour in connection with Core MIDI. The implementation of synchronization revealed that the task of having synchronized playback involves synchronization on two levels: synchronization between interconnected devices, and synchronization between clock messages and audio engine. This led to the most critical challenge of this functionality: having precise timing between generated clock messages and audio engine. An audio render callback in the audio engine was used for handling time, and this approach showed a limitation on the detection of time intervals. A solution that overcame this limitation was presented.

The resulting implementation of MIA 2 demonstrated a musical application that enables several users to connect, to interact with each other through messages carrying musical parameters, and to have synhronized playback on all connected devices.

Measuring the accuracy of the synchronization in MIA 2, time deviations of 4 - 16 milliseconds were detected. When comparing the results with a related work, the synchronization in MIA 2 performs favourably. Suggestions on how to further improve the accuracy of the synchronization have been described.

10.1. Future research

A prerequisite behind the development of MIA 2 has been that the system should be expanded with further functionality in the future. The implemented communication easily allows creating new messages to enable new interaction forms. Decisions on the forms of interaction must be done guided by considerations on the motivations behind the system, the target user group, and the social organisation of the system. The development of instruments in the application and how to interact with them is closely related to this area. A holistic view on the application´s functionality and the users´ experiences in using the application is required to conduct this development. As such, further development will rely on knowledge within interaction design and musicology. Testing various forms of interaction on potential users is also a good way of gaining insight into how the application can be extended. User testing on MIA 2 is already planned, and these tests will likely provide feedback on the current implementation as well as identify needs for further functionality.

Although the performance of the playback synchronization in MIA 2 proved to give fully usable results for the intended use of the application, improvements can be made to the timing performance. Identifying the source of jitter in the synchronization is necessary to be able to improve the inaccuracy. On the receiver side, handling delayed packets is a way to improve the accuracy. Also, making improvements to the thread handling incoming MIDI data is suggested to improve this situation. A more thorough investigation into how threads are handled by iOS in correlation with Core MIDI is necessary to identify in which way the timing can be improved.

To be able to fully exploit interaction between devices, it must be possible to address messages to a specific device. This requires that each device is distinguished in a meaningful way, and using MIDI channels is a common way of doing this in a MIDI setup. To

accomplish the distribution of channel numbers in a network like that of MIA 2, a way of assigning these channels must be implemented. Further work must be done to solve this.

The *Open Music App Collaboration Manifesto* suggests ways music apps can increase their ability to connect and communicate with each other through MIDI. Taking this idea one step further, we can envision a standard that tells developers how music apps can be created to enable connecting and participating in a musical network that allows users to interact and collaborate on a shared musical performance. This is an exciting topic for future research and might contribute to opening up new possibillities for collaborative music on mobile devices.

11. BIBLIOGRAPHY

Adamson, C., & Avila, K. (2012). *Learning Core Audio: A Hands-On Guide to Audio Programming for Mac and iOS.* Pearson Education Inc.

AKAI. (2012). AKAI MPC5000. Retrieved 11.07, 2012, from http://www.akaipro.com/mpc5000.

Alfke, J. (2009). Re: Simple Audio Player. *Apple Mailing Lists* Retrieved 12.12, 2011, from http://lists.apple.com/archives/coreaudio-api/2009/Feb//msg00103.html.

Apple. (2009). Technical Q&A QA1643: Audio Host Time On iPhone OS. *iOS Developer Library* Retrieved 09.03, 2012, from http://developer.apple.com/library/ios/#qa/qa1643/_index.html.

Apple. (2010a). Core MIDI Framework Reference. *iOS Developer Library* Retrieved 22.01, 2012, from http://developer.apple.com/library/ios/#documentation/MusicAudio/Reference/CACoreMIDIRef/_index.html#//apple_ref/doc/uid/TP40002091.

Apple. (2010b). MIDINetworkSession Class Reference. *iOS Developer Library* Retrieved 16.03, 2012.

Apple. (2010c). WiTap. *iOS Developer Library* Retrieved 09.03, 2012, from http://developer.apple.com/library/ios/#samplecode/WiTap/Introduction/Intro.html.

Apple. (2011a). Bonjour Overview. *iOS Developer Library* Retrieved 21.05, 2012, from https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/NetServices/Articles/about.html#//apple_ref/doc/uid/TP40002458-SW1.

Apple. (2011b). iOS Technology Overview. *iOS Developer Library* Retrieved 15.06, 2012, from http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/IPhoneOSOverview/IPhoneOSOverview.html#//apple_ref/doc/uid/TP40007898-CH4-SW1.

Apple. (2011c). MIDI Services Reference. *iOS Developer Library* Retrieved 19.04, 2012, from http://developer.apple.com/library/ios/#DOCUMENTATION/CoreMidi/Reference/MIDIServices_Reference/Reference/reference.html.

Apple. (2011d). NSTimer Class Reference. *OS X Developer Library* Retrieved 19.05, 2012, from https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/nstimer_Class/Reference/NSTimer.html.

Apple. (2011e). The Objective-C Programming Language. *Mac Developer Library* Retrieved 21.01, 2012, from http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html.

Ballagas, R., Borchers, J., Rohs, M., & Sheridan, J. G. (2006). The Smart Phone: A Ubiquitous Input Device. *Pervasive Computing*, *5*(1), 70-77.

Bencina, R. (2011). Dave Sparks on Android audio latency at Google I/O 2011. Retrieved 11.07, 2012, from http://www.rossbencina.com/code/dave-sparks-on-android-audio-latency-at-google-io-2011.

Benzon, W. (2002). *Beethoven´s Anvil: Music In Mind And Culture.* Oxford University Press.

Blaine, T., & Fels, S. (2003). *Contexts of Collaborative Musical Experiences*. Proceedings from Conference on New Interfaces for Musical Expression (NIME-03).

Bonnington, C. (2011). Google's 10 Billion Android App Downloads: By the Numbers. Retrieved 11.06, 2012, from http://www.wired.com/gadgetlab/2011/12/10-billion-apps-detailed/.

Brinkmann, P., Kirn, P., Lawler, R., McCormick, C., Roth, M., & Steiner, H.-C. (2011). *Embedding Pure Data with libpd.* Proceedings from Proceeding of the Fourth International Pure Data Convention.

Brown, C. (1999). Talking Drum: A Local Area Network Music Installation. *Leonardo Music Journal*, *9*, 23-28.

Butler, J. G. (1998). A History of Information Technology and Systems. Retrieved 01.03, 2012, from http://www.tcf.ua.edu/AZ/ITHistoryOutline.htm.

Cappelen, B., & Herstad, J. RHYME. Retrieved 30.03, 2012, from http://rhyme.no/.

Dammerud, J. J. (2011). Stage Acoustics for Symphony Orchestras - Just Black Magic? Part I. Retrieved 09.03, 2012, from http://www.polyphonic.org/article.php?id=247&page=8.

Essl, G., & Rohs, M. (2007). *ShaMus - A Sensor-Based Integrated Mobile Phone Instrument*. Proceedings from International Computer Music Conference, Copenhagen.

Essl, G., & Rohs, M. (2009). Interactivity for Mobile Music-Making. *Organised Sound*, *14*(2), 197-207.

Feldmeier, M., Malinowski, M., & Paradiso, J. A. (2002). *Large Group Musical Interaction using Disposable Wireless motion sensors*. Proceedings from 2002 International Computer Music Conference, Gothenburg.

Finger-Pro. (2011). Avoiding the Pitfalls of CoreMIDI Programming (for Developers). Retrieved 20.01, 2012, from http://www.finger-pro.com/docs/avoiding-the-pitfalls-of-coremidi-programming-for-developers.html.

Gabrielsson, A., & Juslin, P. N. (1996). Emotional Expression in Music Performance: Between the Performer´s Intention and the Listener´s Experience. *Psychology of Music*, *24*, 68-91.

Gregg, D. G., Kulkarni, U. R., & Vinzé, A. S. (2001). Understanding the Philosophical Underpinnings of Software Engineering Research in Information Systems. *Information Systems Frontiers*, *3*(2), 169-183.

Guba, E. G., & Lincoln, Y. S. (1994). Competing Paradigms in Qualitative Research. In Y. S. Lincoln & N. K. Denzin (Eds.), *Handbook of qualitative research* (pp. 105-117). Sage.

Guillot, J. (2011). Android is far behind iOS. *Musique tactile* Retrieved 10.07, 2012, from http://www.musiquetactile.fr/android-is-far-behind-ios/.

Holmes, P. A. (2011). An exploration of musical communication through expressive use of timbre: The performer´s perspective. *Psychology of Music*, *40*(3), 301-323.

Huntington, J. (2007). *Control Systems for Live Entertainment* (13.06, Trans.). Focal Press.

Johnson, P. (2011). CoreMIDI Networking Setup. Retrieved 16.03, 2012, from http://www.onereddog.com.au/2011/04/08/coremidi-networking-setup/.

Kaltenbrunner, M. (2005). *Interactive Music for Mobile Digital Music Players*. Proceedings from Inspirational Idea for the International Computer Music Conference, Barcelona.

Kay, R. (2009). Pragmatic Network Latency Engineering. Fundamental Facts and Analysis. Retrieved 26.04, 2012, from http://www.cpacket.com/download/Introduction%20to%20Network%20Latency%20Engineering.pdf.

Kerns, A. (2011). CoreMIDI Brain Dump. Retrieved 03.02, 2012, from http://syntheticbits.com/blog/?p=878.

Lázaro, M., & Marcos, E. (2006). An Approach to the Integration of Qualitative and Quantitative Research Methods in Software Engineering Research. *Workshop on philosophical Foundations of Information Systems Engineering,* 757-764.

Levin, G. Dialtones (A Telesymphony). Retrieved 29.04, 2012, from http://www.flong.com/storage/experience/telesymphony/index.html.

Lyons, G. (2012). 2012 Mobile Market Share [Infographic]. Retrieved 11.06, 2012, from http://connect.icrossing.co.uk/2012-mobile-market-share-infographic_7962.

Madsen, T. (2012). MUSIC IMPRO APP - en empirisk undersøkelse av mobilen som et musikalsk instrument for sosialt samvær i familier.

Marcos, E. (2005). Software Engineering Research versus Software Development. *ACM SIGSOFT Software Engineering Notes*, *30*(4), 1-7.

Merriam-Webster.com. “Information technology”. *An Encyclopædia Britannica Company* Retrieved 16.06, 2012, from http://www.merriam-webster.com/dictionary/information%20technology.

MIDI Manufacturers Association. MIDI messages. *Learn About MIDI* Retrieved 13.05, 2012, from http://www.midi.org/techspecs/midimessages.php.

MIDI Manufacturers Association. Tutorial: History of MIDI. *Learn About MIDI* Retrieved 16.02, 2012, from http://www.midi.org/aboutmidi/tut_history.php.

Myers, M. D. (2011). Qualitative Research in Information Systems. *MISQ Discovery* Retrieved 12.03, 2012, from http://www.qual.auckland.ac.nz/.

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research*, *2*(1), 1-28.

Pavlicevic, M. (2002). Dynamic Interplay in Clinical Improvisation. *Voices: A World Forum for Music Therapy*, *2*(2).

Plos, O., & Buisine, S. (2006). *Universal Design for Mobile Phones: A Case Study*. Proceedings from CHI ´06 extended abstracts on Human factors in computing.

(2012). BlackBerry Developer. Retrieved 11.06, 2012, from https://developer.blackberry.com/devzone/appworld.

Rheingold, H. (1999). Look Who´s Talking. *Wired*, *7.01*.

Richardson, S. The MIDI Specification. Retrieved 24.06, 2012, from http://www.gweep.net/~prefect/eng/reference/protocol/midispec.html.

Robertson, P. (2001). Music and health. *Design and Health—The Therapeutic Benefits of Design*.

Ruud, E. (1997). Music and the Quality of Life. *Nordisk Tidsskrift for Musikkterapi*, *6*(2), 86-97.

Schiemer, G., & Havryliv, M. (2006). *Pocket Gamelan: tuneable trajectories for flying sources in Mandala 3 and Mandala 4*. Proceedings from International Conference on New Interfaces for Musical Expression (NIME06).

Schiemer, G., & Havryliv, M. (2007). *Pocket Gamelan: interactive mobile music performance*. Proceedings from Mobility Conference 2007: The 4th International Conference on Mobile Technology, Applications and Systems, Singapore.

Shaffer, L. H. (1984). Timing in solo and duet piano performances. *The Quarterly Journal of Experimental Psychology Section A: Human Experimental Psychology*, *36*(4), 577-595.

Skotterud, J. O., Madsen, T., Sethre, G., Vestvik, K., & Bording, J. (2011). The Music Impro App - Final Report. Retrieved 28.05, 2012, from http://www.uio.no/studier/emner/matnat/ifi/INF5261/v11/studentprojects/music-impro-app/Final%20Report%20-%20Music%20Impro%20App.pdf.

Solli, H. P. (2009). Musikkterapi som integrert del av standard behandling i psykisk helsevern. *Musikk i psykisk helsearbeid med barn og unge*, *5*, 15-36.

Sorrel, C. (2011). Nokia Kills Symbian, Teams Up With Microsoft For Windows Phone 7. Retrieved 11.06., 2012, from http://www.wired.com/gadgetlab/2011/02/microsoft-and-nokia-team-up-to-build-windows-phones/.

Story, M. F., Mueller, J. L., & Mace, R. L. (2011). The Universal Design File: Designing for People of All Ages and Abilities. *Design Research and Methods Journal*, *1*(1).

Trueman, D., Cook, P., Smallwood, S., & Wang, G. (2006). *PLOrk: The Princeton Laptop Orchestra, Year 1*. Proceedings from Proceedings of the International Computer Music Conference, New Orleans.

Turkle, S. (2012). Connected, but alone? *TED talks* Retrieved 02.05, 2012, from http://www.ted.com/talks/sherry_turkle_alone_together.html.

Tyson, M. (2011). Experiments with precise timing in iOS. *A Tasty Pixel* Retrieved 06.07, 2012.

Wang, G., Essl, G., & Penttinen, H. (2008). *Do Mobile Phones Dream of Electric Orchestras?* Proceedings from International Computer Music Conference.

Weinberg, G. (2003). Interconnected Musical Networks - Bringing Expression and Thoughtfulness to Collaborative Group Playing.

Weinberg, G. (2005). Interconnected Musical Networks: Toward a Theoretical Framework. *Computer Music Journal*, *29*(2), 23-39.

Weinberg, G., & Gan, S.-L. (2001). The Squeezables: Toward an Expressive and Interdependent Multi-player Musical Instrument. *Computer Music Journal*, *25*(2), 37-45.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 94-104.

Wöhrmann, R. (2011). Open Music App Collaboration Manifesto. Retrieved 21.03, 2012, from https://docs.google.com/document/d/1UW-8vPEf95p0zO0hV1lpwD5MTgefKB1y-jdWR-nFYM8/edit?hl=en_US&pli=1.

Wright, M., Cassidy, R. J., & Zbyszynski, M. F. (2004). *Audio and Gesture Latency Measurements on Linux and OSX*. Proceedings from Proceedings of the ICMC.

Wright, M., & Freed, A. (1997). *Open Sound Control: A New Protocol for Communicating with Sound Synthesizers*. Proceedings from International Computer Music Conference.

Appendix A

Handling discovery of MIDI devices on the network through Bonjour:

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)aBrowser
            didFindService:(NSNetService *)service
              moreComing:(BOOL)more
{
    [service retain];
    service.delegate = self;
    [service resolveWithTimeout:5];
}


- (void)netServiceDidResolveAddress:(NSNetService *)service
{
    NSString *name = service.name;

    // Don't add the local machine
    if (![[service name] isEqualToString:[[UIDevice currentDevice] name]]) {

        // Add the device
        [services addObject:service];

              // Create host object
        MIDINetworkHost* contact = [MIDINetworkHost hostWithName:name
                                                      netService:service];

        BOOL exists = NO;
        for (MIDINetworkHost* theHost in session.contacts) {
            if ([theHost.name caseInsensitiveCompare:name] == NSOrderedSame) {
                exists = YES;
                break;
            }
        }
        // Add the host if it is not already added
        if (!exists) {
            [session addContact:contact];
        }
    }
    [service release];
}
```

Appendix B

Preparing and sending 6 MIDI clock messages given the current host time:

```
- (void) sendClockTicksQueued:(UInt64)currentTime {

    // Create a MIDI clock status byte
    const UInt8 clockTick[] = { 0xF8 };

    // Calculate timestamp of the first clock message
    UInt64 atStartTime = currentTime + latencyAbsolute;

    UInt32 size = sizeof(clockTick);
    Byte packetBuffer[size+100];
    MIDIPacketList *packetList = (MIDIPacketList*)packetBuffer;
    MIDIPacket     *packet     = MIDIPacketListInit(packetList);
    UInt64 timeAhead;

    for (int i = 0; i < CLOCKTICKS_PER_PACKAGE; i++) {
        timeAhead = atStartTime + (tickDelta*(float)i);
        packet = MIDIPacketListAdd(packetList, sizeof(packetBuffer),
                                   packet, timeAhead, size, clockTick);
    }

    CheckError(MIDISend(outputPort, dest, packetList), "Error sending MIDI data");
}
```

Appendix C

Code handling the timing of clock generation and audio playback:

```
OSStatus renderCallback(void *inRefCon, AudioUnitRenderActionFlags
                *ioActionFlags,
                const AudioTimeStamp *inTimeStamp, UInt32 inBusNumber,
                UInt32 inNumberFrames, AudioBufferList *ioData) {

   // Pointer to class instance
   PdAudio *controller = (PdAudio *) inRefCon;


   // MIDI clock should be generated
   if (controller->generateClock) {

      // Calculate elapsed host time since previous
      UInt64 delta = inTimeStamp->mHostTime - controller->prevTimeStamp;

      if (delta > controller->machTimePerStep) {

         UInt64 deviation = 0;
         UInt64 advanceTimestampCorrected = 0;

         if (controller->prevTimeStamp != 0) {
            // prevTimeStamp is set, advance time one step
            controller->prevTimeStamp += controller->machTimePerStep;

            // calculate time deviation and correct the timestamp
            deviation = delta - controller->machTimePerStep;
            advanceTimestampCorrected = controller->prevTimeStamp - deviation;
         }
         else {
            // on startup, set prevTimeStamp
            controller->prevTimeStamp = inTimeStamp->mHostTime;
            advanceTimestampCorrected = controller->prevTimeStamp +
                                                   controller->machTimePerStep;
         }

         // generate 6 MIDI clock messages from calculated timestamp
         [controller sendClockTicks:advanceTimestampCorrected];

         // advance audio sequencer one step after delay indicated by "deviation"
         [controller advanceSeqStepTimed:deviation];
      }
   }
}
```