Control Strategies in Real-Time Interactive Digital Sound Synthesis

Submitted by
Chinmay Prafulla Pendharkar
Department of Electrical & Computer Engineering

In partial fulfilment of the
requirements for the Degree of
Bachelor of Engineering
National University of Singapore

ABSTRACT

Real-time interactive digital sound synthesis has become an increasingly important component in a variety of applications including music and video games across a variety of platforms from networked computers to mobile phones. The paradigmatic approach to interactive sound synthesis can be conceptualized as a hierarchical structure with a sound synthesis engine generating sound at the lowest level, and a human or automated controller at the highest level. The controller generates some number of digital control signals as input to the synthesis system, which must be mapped to the parametric control handles of the sound synthesis engine. Even with rich synthesis schemes capable of dynamic, responsive sounds, inadequate control strategies and mappings can result in dull, "mechanical" sonic output.

This project tried to address the issue by proposing a novel control system to control the synthesis of sound. The three important modules of the system, the communications system, the data-acquisition system and the mapping system are designed and developed through this project. The final method yields a control system that is able to provide the user, expressive, intuitive and comprehensible control over the synthesis of the sound.

ACKNOWLEDGMENTS

I would like to thank Prof. Lonce Wyse for his valuable guidance and advice throughout the course of the project. I would also like to thank Prof. Ng Chun Sum for supervising me on the project. I also express my gratitude to Michael Gurevich from I^2R's Mixed Modelling Lab, for his help, guidance and support during this project. I thank Mr. Lim Sheng Hwee for his help as well as reviews of certain part of the project. I would also like thank Institute of Infocomm Research for allowing me to use their resources for the project.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

ADC	*Analogue to Digital Converter*
API	*Application Programming Interface*
ARP	*Address Resolution Protocol*
DAC	*Digital to Analog Converter*
DHCP	*Dynamic Host Configuration Protocol*
GNU	*GNU's Not Linux*
GUI	*Graphical User Interface*
GPL	*GNU Public License*
GPS	*Global Positioning System*
HID	*Hardware Interface Device*
HTTP	*Hyper Text Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IP	*Internet Protocol*
I/O	*Input-Output*
JOE	*Java OSC Extension*
LCD	*Liquid Crystal Display*
MIDI	*Musical Instrument Digital Interface*
MIPS	*Million Instruction Per Second*
MOE	*Microcontroller OSC Ethernet*
NIC	*Network Interface Card*
NTP	*Network Time Protocol*
OSC	*Open SoundControl*
RISC	*Reduced Instruction Set Computer*
SRAM	*Static Random Access Memory*
TCP	*Transfer Control Protocol*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*

CHAPTER 1
Introduction

1.1 Overview

This chapter introduces sound synthesis and explores its various forms. It then looks at the issues involved in sound synthesis and sound models. It goes on expresses the problem statement addressed in this project. This is followed by the organisation of the thesis.

1.2 Sound Synthesis

Digital sound synthesis is the process of electronically generating sound. This includes a range of sounds from sounds that already exist in nature to completely original sounds. Digital sound synthesis has been a field of much development for several years. The advent of new technologies has brought about newer methods of synthesis sound. With these new technologies, newer uses for synthesised sound have also come about.

There are numerous methods of sound synthesis, based on different principles, approaching the process in different domain and perspectives [1, 2, 3]. Many of the methods are used to create specific types of sounds, attributing to the features of the approach. The more popular methods are FM Synthesis, Additive synthesis and Physical Modelling synthesis, which are discussed in the next section

1.2.1 FM Synthesis

Frequency Modulation (FM) Synthesis is a form of audio synthesis where the timbre of a simple waveform is changed by modulating it with a modulating frequency which is also in the audio range, resulting in a more complex waveform and a different-sounding tone [4]. From a mathematical perspective, the resultant sound wave would obey equation (1).

$$s(t) = a \cdot \cos(2\pi \int_0^t [f_c + f_\Delta \, x_m(\tau)] \, d\tau) \qquad - (1)$$

Here f_c is the original frequency and $f_\Delta \, x_m(\tau)$ is the instantaneous frequency deviation caused by the modulation.

1.2.2 Additive Synthesis

Additive synthesis is a technique of audio synthesis which evolves from the concept of musical timbre. Additive synthesis recreates timbres of a specific sound by synthesizing numerous waveforms having a pitch of the different harmonics in the timbre, shaped by an appropriate amplitude envelope [5]. Additive synthesis is based on the Fourier Series. Since each sound can be decomposed into an addition of infinite (or finite, depending on the harmonic nature of the sound) series of sinusoids, according to the Fourier series; a sound can be generated by adding a large number of sinusoids in a calculated fashion. Thus the Fourier series equation, (2), governs this method of synthesis.

$$s(n) = a_0 \cos(f_0 n + \theta_0) + a_1 \cos(f_1 n + \theta_1) + a_2 \cos(f_2 n + \theta_2) + ... \qquad - (2)$$

Where s(n) is the sound wave being synthesized and the fi are the frequency components.

1.2.3 Physical Synthesis

Physical synthesis is a newer concept in sound synthesis. It takes a bottom-up perspective at the process. Using sets of equations and algorithms, it simulates the physical source of sound. Sound is then generated governed by parameters that describe the physics of the source and the various actions that can be performed on the source [1].

Physical synthesis is based on the physical knowledge of a sound source. Since most sound generating objects can be modelled, especially their acoustic aspect, physical synthesis offers an intuitive and yet modularisable method of synthesizing sound. Parts of a source can be modelled separately and coupled to each other to create the whole source. For example, to model a guitar, the strings can be modelled separately to the body, and combined during the synthesis process or after it.

1.3 Java Synthesis System

Any type of sound synthesis requires a synthesis system or ‘a synthesis engine’. A mathematical engine which does the calculations and logical operations required to generate the sounds.

The Java Synthesis System is a sound modelling based synthesis system developed by Prof. Lonce Wyse from the Mixed Media Modelling Lab at the Institute of Infocomm Research (I^2R) [6]. The system allows real-time and interactive synthesis of sound based on various techniques, with physical synthesis being its main forte. This thesis uses the Java Synthesis System as an example synthesis system to demonstrate the ideas and methodologies developed throughout.

The Java Synthesis System uses sound models as an abstract when dealing with synthesis techniques. However, the concept of sound models is not restricted to the Java Synthesis System and can be used in the study of synthesis techniques and tools for better understanding of the subject matter.

1.4 Sound Models

Sound Models are abstracts used to define a sound producing object, which exposes a set of parameters and defines a sound space. Sounds Models can be thought of as software analogues of individual Synthesizers, and can be considered as a primitive structure in software based synthesis system.

Model can produce sounds using any synthesis technique. They are not restricted to use any technique or to use just one technique. In a much wider context, a model could just play back a pre-recorded sound on certain activity on a parameter.

Each Sound Model exposes a set of parameters. These parameters are the properties of sounds which the model produces. For example, a guitar string model could have a parameter named ‘string length’, which, when changed, would allow the model to produce a sound of a guitar string of the specified string length. The type of parameters and the effect of each parameter on the sound produced depends on the type of synthesis technique being used, as well as the specific model. The earlier example would probably come from a model using physical modelling synthesis technique. Parameters also are the element of interactivity in Sound Models. They allow a user to change the sound being produced by such a model by performing some activity on some of its parameters.

Each sound model can produce a multitude of sounds. The whole set of sounds which can be produced by a specific sound model can be considered to span a space of sounds, with the parameters as dimension of the space. Thus, a model with *n* parameters is said to span an *n*-dimensional parameter sound space.

1.5 Control

Synthesis allows us to produce a large variety of sounds. However, in most application we need to be able to control the type of sounds produced. Different applications require different types of controllers. For example, for a sound needed to be synthesised in a computer game, the controller is game software. When mimicking musical instruments using synthesis techniques, the controller could be hardware based gesture acquisition systems [7].

Synthesis using Sound Models can be controlled using some parameters or handles. Such parameters are exposed to the controller of the synthesis system, which can be a software controller or a hardware based controller.

1.5.1 Interactive Control

Static control might be useful in some cases of synthesis, but in most modern applications, there is a need for user based interactive control. Using the earlier example of computer games, the game player would generate control data which will change the type of sound being produced. For a better example, if the computer games includes a car, which can be driven by the user, and the sounds produced by the car are being synthesised by an synthesis engine, control parameters like the speed and the type of road, would have to change the type of sound being produced. Such applications are more common, than those where there is no dynamic control from an external entity to the synthesis system as to what kind of sound is to be produced.

1.5.2 Gestural Control

Gestural control is another aspect of control related to sound synthesis, especially in terms of musical sound synthesis [7]. Gestural control uses physical gestures by human users, like movement of arms and legs, to generate control data.

It has always been regarded as a challenging method of generating control data, because of the various types of sensors needed and a large amount of processing required before the data could be used. However, with newer technology in sensors and gestural data-acquisition, gestural control has become a viable option for interactive sound control applications.

1.5.3 Real-Time Control

Another aspect to control of synthesis systems, in this project is real-time. From a very simplistic perspective, real-time control ensures that any change in the control data, is reacted upon immediately by the synthesis system and the sound being produced by the system should change immediately. However, in practical applications, this definition has to change, since use of delays might change response time of the synthesis systems. Furthermore, in some cases delays could be required feature of the synthesis system. Thus, the 'real-time'ness of the system should depend on how accurate it is with respect to our expectations or design.

1.6 Control in Synthesis Techniques

Controlling synthesis systems has always been a problem. All the major synthesis techniques have their own disadvantages when it comes to a varied use.

While FM synthesis is based on a very simple concept of frequency modulations, the synthesis technique is indeed very difficult to control and obtain useful sound from. The complexity of the governing equation hinders the use of this technique. Much research has been done to use this technique for synthesis applications with varied results.

Additive synthesis is an exhaustive technique, and theoretically, it can produce any sound algebraically conceivable. However, it is impractical, and is limited by the number of frequency components are able to be summed to produce the sound.

This technique suffers greatly when real-time limitations are added to the synthesis system.

While physical synthesis systems provide a viable methodology of synthesis, offering a wide variety of sounds to be produced and a very fine control on the synthesis process itself, the control of such systems can be extremely complex. The complexity of the algorithms and the equations involved in the synthesis can cause the technique to be difficult to understand, thus limiting the scope of the user.

Typically Sound Models using the above techniques expose 20 to 30 parameters for the controller to control. This is too far great a number for users to control individually. Generally, a human user with a gesture based controller is unable to comprehend, and thus control, more than 5-9 handles [8]. Many times, the required input control handles available for synthesis are not related directly to the synthesis parameters, and thus have to be processed before being passed on to the synthesis systems.

Thus, to be able to fully utilise the potential of various synthesis systems, effective control systems have to be developed. Control systems which allow easier and more effective real-time, interactive, control over the various types of synthesis techniques. This thesis proposes a set of novel techniques and tools for better control of synthesis systems.

1.7 Organisation of Thesis

The second chapter discusses the approach taken to address the problem defined by this thesis. The third chapter looks at the current state of the art for the various systems involved in the proposed solution to the problem. The fourth chapter discusses the design and the development of the data-acquisition system. The fifth chapter discusses the design and the development of the communication sub-system. The sixth chapter discusses the design and the development of the mapping system, the main part of this thesis. The seventh chapter evaluates the performance of the system and looks as some example applications of the system. The eighth chapter concludes the thesis and suggests topics for future work related to this thesis.

CHAPTER 2
Approach

2.1 Overview

This chapter looks at the approach to the problem statement, discussed in the previous chapter. It discusses the concept of control for synthesis, and the various important subsystems of a control system.

2.2 Controlling Synthesis Systems

A simple interactive synthesis system can be modelled as below. The control data flows from the controller through the communication linkage to the synthesis engine. The feedback data flows through back from the synthesis engine, to the controller.

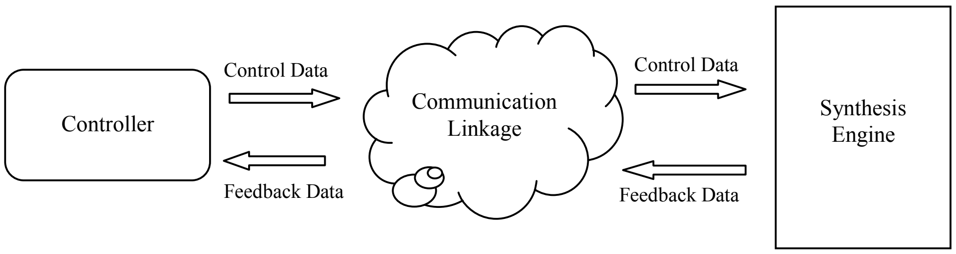

Figure 1:Model of a simple interactive synthesis system.

This simple model shows that an effective control of a synthesis system can be a challenge. There are many aspects to this challenge. Firstly, an interactive controller has to exist which can be used to acquire the right inputs and give appropriate feedback to the user. Secondly the communication linkage has to be able to transmit both the control as well as the feedback data, with great logical as well as temporal fidelity. Finally, the interfaces have to be defined. If the number and/or types of control outputs from the controller do not match the parameters exposed by the sounds models in the synthesis engine, there needs to be some kind of mapping in order to couple these two systems.

To approach this challenge of developing an effective synthesis control system, a generic interactive controller needs to be defined and built to support all tests and experimental setups of this system. Also a proper communication mechanism needs to be laid out which is capable of supporting such communication at the required fidelity. And finally a mapping system needs to be created to allow the controller to control the synthesis system effectively. Thus, we can logically separate the approach into three subsystems. Each dealing with its own domain, and yet coming together as one to form the whole synthesis control system. The approaches to individual systems are defined in the following sections.

2.2.1 Interactive Data-Acquisition Subsystem

An interactive gestural data-acquisition system is the front end of a synthesis control system. Otherwise called ‘the controller’, this would consist of sensors and some type of processing hardware, and some feedback devices. It would then be able to connect with the communication subsystem and deliver the acquired control data to the synthesis system.

Complex gestural data-acquisition systems can be considered in this approach. These wouldn’t be much different, except for the types of sensors needed, and the amount of processing required. For feedback on such controllers, haptic devices could be explored as an option.

While, large, powerful and expensive data-acquisition systems are widely available, such systems are limited in their usages because of their size, power requirements and costs. Thus, a low-cost, simple microcontroller based system might serve as a useful and practical data-acquisition system, especially in the case of this project. Furthermore, with the focus of new technology towards embedded devices, such a system can be extendible for the future addition for newer technologies.

2.2.2 Communications Subsystem

The acquired data has to be passed on the data to the synthesis system. This requires an efficient and practical communications system. Such a communication system should be able to transmit data in different forms, over various mediums, with high logical and temporal fidelity, for a generic solution for the problem. Templar fidelity of the system is necessary to allow a real-time control of the synthesis. Since an embedded microcontroller based solution is being considered for the data-acquisition systems, the communication system should also be able to support such a system.

Keeping the rapidly changing communication technologies in mind, the communication system for such an application has to be extendible for future usage, and independent of medium, thus able to work in multiple scenarios, including over mediums which might not yet exist. Such a generic system allows this solution to work in many different application environments, across many different platforms.

2.2.3 Mapping Subsystem

The final and the most important part of the control system is the mapping subsystem. The control inputs extracted from the data-acquisition system might not necessarily correspond to the synthesis parameters. The control inputs need to be adjusted for range, number, and type (discrete or continuous) to match the synthesis parameters. They may also need to be having various functional relationships with synthesis parameters. For example, the 'volume' synthesis parameter has to change over an inverse square relationship with respect to a 'distance' control input. It is also possible in many cases that the exact nature of such a mapping in not known and might need to be dynamic, changing with respect to time, or non-deterministic, random in nature. Finally, there can be multiple control inputs that affect a single synthesis parameter and vice versa.

To be able to achieve all such types of adjustments and processing that have to be done to the control input, a mapping scheme is the effective solution. A generic mapping scheme can be conceptualised as a functional relationship between the controls input vector and synthesis parameter vector. In *2*-dimensions, the functional relationship can be visualised as a plane containing functions and the control inputs are adjusted using those functions depending their positioning on the plane. The following figure, Figure 2, illustrates the idea

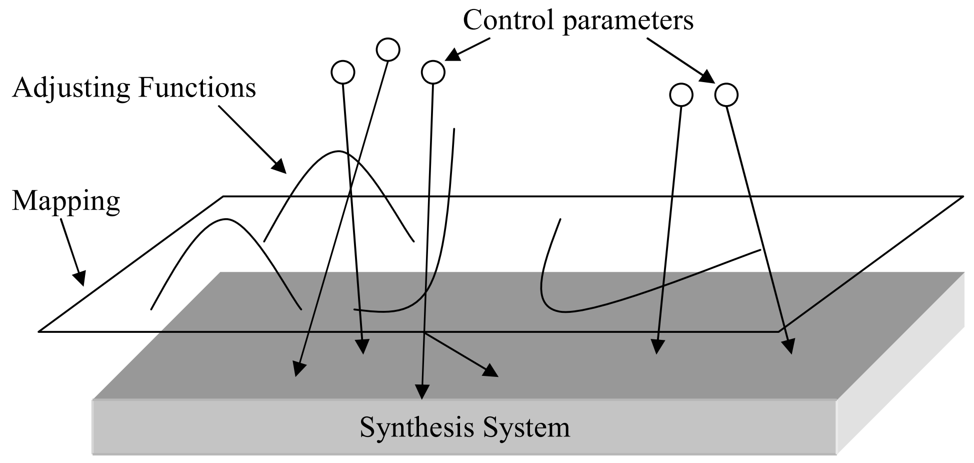

Figure 2: 2-D Conceptual representation of mapping.

The control parameter's position on the 2-D plane is defined by their values. They are mapped to synthesis parameter on another 2-D plane. This is of course a simplification for visualisation purposes. Actually mappings may be more complex and have dimensional asymmetry, where the synthesis and control parameters have different dimensions.

Thus, a mapping scheme, which is able to effectively map various types of control input to a given synthesis engine, would form the mapping subsystem. If designed and developed well such a system could yield a control system able to make the synthesis system produce a variety of sounds, in the most effective and user-friendly way.

2.3 Modularisation of Control Subsystems

An important aspect of the approach taken in this project is the modularisation of control system. The system is designed to be modular, such that, any of the modules or subsystems is able to work independently. The above mentioned three subsystems can be used in other applications, and thus should not be restricted to this specific application. A modular design allows for such development. Figure 3 illustrates the envisioned modularised control system.

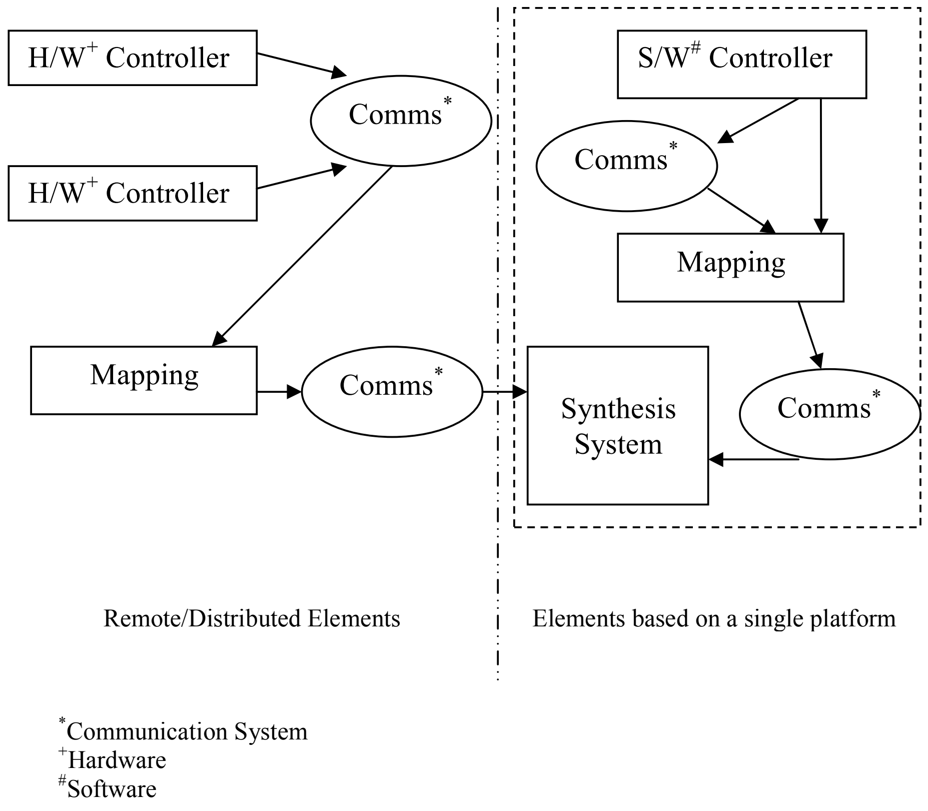

Figure 3: Envisioned modularised control system.

However, it should be designed so as not create too many interface layers; as such it would slow down the control process itself. Thus would be detrimental to the real-time aspect of the control system.

2.4 Control Tool Chain and Synthesis

The problem of control of physical synthesis systems is approached by defining the three important modules, which it should contain. And looking at each individual module, its requirements and functionalities, leading to a chain of tools corresponding to each module to implement the needed functionalities and fulfil the requirements.

CHAPTER 3
Literature Review

3.1 Overview

This chapter looks at the current state-of-art and written literature for each of the three subsystems defined in the previous chapter. Using this knowledge, a design can be discussed for the implementation each of the subsystems.

3.2 Interactive Data-Acquisition

There exist a number of platforms for developing sensor-based data-acquisition systems. These come in a variety of styles and configurations. Most are essentially DACs, which convert analog sensor voltages into digital signals and encode them according to a communication protocol, and transmit the digital signals over a hardware interface. Accompanied by signal conditioned sensors, data-acquisition systems offer a straightforward way to do gestural control over digital synthesis without any programming and minimal knowledge of electronics.

Jensenius [9] presents an extremely low-cost alternative following the same paradigm of a data-acquisition system by using inexpensive, discarded game-controllers which use the USB Human Interface Device (HID) protocol. They rely on readily available HID drivers to interface with music software applications. Another low-cost microcontroller-based USB data-acquisition system is described in the SensorWiki [10].

Currently available data-acquisition systems for sound synthesis control include the I-CubeX [11], Kroonde, Toaster [12], Eobody [13], EtherSense [14], Teabox

[15] and WiSe Box [16]. Several features of these systems are compared in Table 1.

Table 1: Comparison of selected sound control Data-Acquisition system

System	iCubeX	Kroonde	Toaster	Teleo	Eobody	EtherSense	Teabox	WiSe Box
Physical Connection (s)	MIDI, Bluetooth	Ethernet, MIDI	Ethernet, MIDI	USB	MIDI	Ethernet	SPDIF	802.11b
Data Protocol	MIDI	OSC(UDP),FUDI, MIDI	OSC(UDP),FUDI, MIDI	Proprietary	MIDI	OSC(UDP)	Proprietary	OSC(UDP)
Max. Sample Rate (Hz)	250	200	200	100	900	1000	4000	200
Max. A/D Resolution (bits)	12	10	16	10	10	16	12	16
No. of Analogue Channels	32	16	16	Extendible.	16	32	8	16
Cost (USD)	$400	$1450	$1450	$189	$575	$1200	$395	$1150

More flexible modular systems, such as Teleo [17], also offer analog-to-digital modules, in addition to things like generic digital I/O, pulse-width modulation, and motor controllers. These can provide a wider variety of modalities of interactivity, but require more technical knowledge on the part of the user. The hardware on these devices is not directly programmable, but they require some amount of programming on the PC to which they interface in order to operate.

Simpler systems, such as the AVR microcontroller system presented in [18] are not specifically interactive gestural data-acquisition systems, but rather powerful, general-purpose toolsets that can be used in a variety of embedded applications including synthesis control. They require programming, as well as some basic knowledge of circuits and electronics. Such systems have also been demonstrated to produce novel, powerful interactive projects by relatively inexperienced designers in short amounts of time [18, 19]. Costs saving of this type of systems can also be an important factor for academic institutions and individual developers, and especially for this project.

3.3 Communications

MIDI is currently the most commonly used communication protocol for control of sound synthesis. The MIDI standard consists of a communications messaging protocol designed for use with musical instruments, as well as a physical interface standard. Physically it consists of a simplex digital loop serial communications electrical connection which runs as 31,250 baud [20].

The MIDI message format consists of 1 to 3 bytes of data, corresponding to an instruction to a synthesiser. However, the number of different instructions and the number of different channels that can be communicated upon is limited to 16. In the context of our synthesis system, each channel would intuitively correspond to a specific Sound Model and thus, there is a limitation on the concurrent number of models that can be used. Furthermore, the data arguments of the messages are restricted to single byte, and cause a reduction of resolution. Thus, MIDI communication standards are limited in many aspects. However, MIDI is widely supported and libraries and APIs for MIDI message generating software can be found easily for various platforms including embedded microcontroller based platforms [18].

A newer communication protocol, OpenSound Control (OSC) is an alternative for such communications [21, 22]. OSC is an open, transport-layer-independent, message-based protocol developed for communication among computers, sound synthesizers, and other multimedia devices. The protocol does not define a transport layer or any layer lower than the transport layer as in the OSI model [23], to be used. And thus the protocol can be used over any of common communication transport layers like TCP or UDP and various data-link layers like Ethernet or WiFi [24]. This implies that there is no restriction on the speed of the communication in the OSC protocol itself, unlike MIDI. The speed is determined by the physical layer being used in the specific application.

OSC Message format is extendible and not limited like MIDI. It follows URL type tree based addressing scheme and supports many data types of arguments. Furthermore, multiple arguments are supported for every message, and wild cards messages are defined for sending multiple messages efficiently. A query system has also been proposed for OSC [25], though it has not been standardised yet. Such the query system would allow the communication system to support advanced features like two way communication, which could support feedback in the controllers, as well as intelligent features like automatic exploration and auto-configuration of the other subsystems.

Implementations of OSC in different languages for various platforms exist. These include JavaOSC in Java [27], flosc in flash [28] OSClib for C [29] and OSC library for Max/MSP [30]. Among these implementations, the Java implementation of OSC would be more relevant for this project since the synthesis system itself is built in Java, thus creating the communication interface to the synthesis system would be easier in Java.

JavaOsc is the first implementation of OSC protocol in Java. It supports the creation of OSC messages and bundles, and their transmission and reception over UDP connection. Received messages are extracted and dispatched to their respective OSC Message Handlers. The implementation provides the various data types used in OSC Communications and provides the API for software applications using JavaOSC. Message dispatching is implemented through call back methods using Java Interfaces.

3.4 Mapping

Approaches to control in the form of mapping have existed in fields such as control systems [31] for some time. Only since the relatively recent advent of practical, interactive real-time digital synthesis has it become applicable to sound synthesis.

Over the years, many strategies have been proposed to approach mapping. A popular mapping strategy is to define formal deterministic relationships between control and synthesis parameters. These strategies are mainly based on techniques predominant in control theory, like linear algebra and matrix transformations [32]. Such techniques tend to be ineffective for control of sound synthesis as they lack the dynamics which are important for synthesis, or tend to be too complex to be used in such an application.

Some authors choose to take non-linear and heuristic-driven approaches that lead to practical and interesting mapping strategies. These include using spatial layout of mapping or weighing functions over a representation of the input space [33] and the uses of geometric shapes to define input parameter spaces and mapping such shapes to shapes of higher dimensionality [34].

Some of these systems introduce one or more intermediate or "middle" mapping layers between control and synthesis parameters. In such schemes, control parameters are mapped onto intermediate parameters, which are in turn mapped to synthesis parameters. These intermediate parameters may be arbitrarily-defined, may describe some higher-level, perceptual, features of the desired sound [35, 36], or may represent some virtual system whose features are mapped onto the synthesis parameters. For example, Schatter et al. [37] use a gestural controller to manipulate virtual graphics objects, whose features are mapped to synthesis controls.

Multiple layered mapping can allow the intermediate mappings to be reused. The mapping from the middle layer to the synthesis parameters may remain unchanged, while another controller is mapped to the middle layer [38]. Multi-layered systems are also one way of dealing with the problem of dimensional asymmetry. Dimensional asymmetry occurs when the dimension of the input control parameters and the output synthesis parameters is not equal. As such, mapping can be considered as a dimension reduction problem, and certain dimension reduction schemes, like simplical interpolation [39], can be used as a mapping strategy.

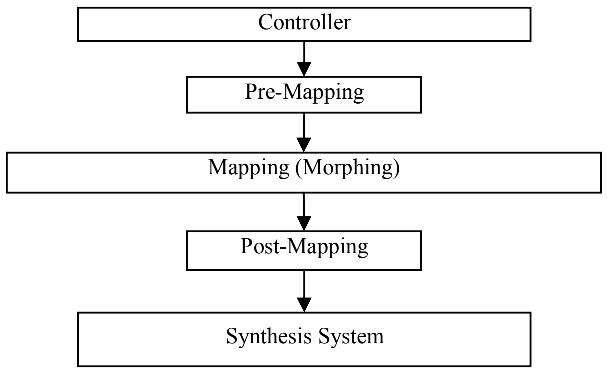

Figure 4: A Multi-layered Mapping Scheme

Though an m-input n-output configurable mapping technique is very generic and widely useful, a mapping technique with restricted input dimension is worthy of consideration. Furthermore, in most applications, fewer dimensional input spaces are sufficient to provide the required control over the synthesis.

Such low input dimension mapping techniques allow users to comprehend and explore the mappings in much greater extent. These techniques are also more intuitive and thus can lead to more expressive mappings. One example of this is a mapping strategy proposed by Bencina [40]. In the strategy the inputs are restricted to a *2*-dimensional space. However no restriction is placed on the number dimensions of the output.

3.5 Review

There is a lot of work that has been done in the areas spanned by this project. Some of these works provide interesting insights on various theories relevant to sound synthesis and its control. Others give interesting ideas and directions for development of various techniques.

The knowledge and understanding gained from the review of all these works are can be used in the development of the system to tackle the problem of effective of digital sound synthesis.

CHAPTER 4
Data-Acquisition System

4.1 Overview

This chapter discusses the design and development process of the data-acquisition system. Initially, the idea of a microcontroller based data-acquisition system is discussed. Then the design of such a system based on a hardware platform is discussed with other hardware and software consideration to implementation of an easy to use data-acquisition system.

4.2 Microcontroller based Data-Acquisition System

The mobility, low power and low memory requirements, low cost and ease of interface with sensors and other devices makes microcontroller a viable solution for data-acquisition for control of synthesis.

The idea is to have a microcontroller based controller with a number of improvements for sound synthesis control and a toolset to go along with the controller. This system has the benefit of low-cost, ease of programming and flexibility. While it does require programming, a simple API can be provided for the user, which may be combined with existing software libraries to greatly simplify the task. For communicating the acquired data the use of Ethernet communication would ensure compatibility with a variety of existing or custom software applications on many operating systems.

4.3 Hardware

The main hardware components of this system are a microcontroller and a network interface chip (NIC) for Ethernet communication. Ethernet is the most simple and easily available form of data-link layer which can support the required communication. These can reside on a single, inexpensive, commercially available development board, like EDTP Easy Ethernet AVR [41]. In the following discussion of the hardware and software of such platforms, the many features of this system that give users the ability to develop new interactive devices related to data-acquisition systems are highlighted.

4.3.1 Microcontroller

The Atmel AVR microcontroller family is an 8bit RISC processor with a well-defined I/O structure. The maximum clock speed of this family is now 24MHz on some devices, offering 24 MIPS. The AVR series feature Harvard architecture, with separate self-programmable Flash program memory and SRAM sections. The microcontrollers come with embedded Analogue-to-Digital converters (ADCs) and other communication protocols like I^2C and SPI [41].

The AVR series [41] includes many devices with different capabilities. An open-source gcc compiler and C libraries offer mostly transparent code portability between devices, and make development of software easy. Hardware development platforms also offer smooth transitions between devices within the family, in the need of extending the capabilities by using a microcontroller with larger memory or more features.

Development boards, such as Procyon Engineering's AVRMini [42] are available for prototyping, and testing AVR microcontroller. The AVRMini is a complete development board with convenient pin headers and sockets to access microcontroller's I/O ports, pin-protecting resistor packs, LEDs and pushbuttons. The AVRMini can also be interfaced with an LCD display with the onboard LCD controller. This can be useful for interactive applications for the controller.

4.3.2 Ethernet Controller

The AVR microcontrollers need an Ethernet controller to communicate using TCP/IP or UDP/IP. The core of the Ethernet controller consists of a NIC. The NIC can connect to the AVR via its extended SRAM interface or using general I/O pins. Several convenient packages allow for a NIC to be easily interfaced to an AVR.

The EDTP Easy Ethernet AVR [43], a development board which includes an AVR microcontroller, Realtek RTL8019AS NIC [44], R3-232 chip and RJ45 network cable connector is a viable solution. The RTL8019AS supports full-duplex communication and the 10BaseT Ethernet standard. It interfaces to the AVR using one of its 8-bit ports for data as well as another 9 pins for addressing and control. Other NIC based development boards are also widely available life the, the EDTP NICholas [43] which uses the 10/100BaseT ASIX AX88796L NIC [44], providing greater speed.

4.4 Software

AVR microcontrollers are supported by excellent development tools and software libraries. These are supplemented by simple libraries for building applications that include sending and receiving OSC messages over Ethernet. This will be discussed in details in the section 5.4.

4.4.1 Development Tools

AVR devices can be programmed in C, using a specialized open-source gcc compiler. The avr-gcc compiler is supported by an array of software tools, including an AVR version of the standard C library, the GNU debugger and simulators. Both open-source and propriety development environments exist for Windows, Macintosh and Linux [45, 46, 47].

4.4.2 AVRLib

The Procyon AVRLib is a thorough and well-documented GPL-licensed library providing high-level access to basic AVR functions like timing and ADC, as well as methods for interfacing with external devices such as LCDs, GPS, hard drives, mp3 chips and accelerometers [41].

4.4.3 Ethernet drivers

Well written drivers are also available for most of the NICs. Mostly written in C, these can be configured and cross-complied using avr-gcc to run on the target microcontrollers.

4.5 Microcontroller based interactive data-acquisition

Thus a simple microcontroller based interactive data-acquisition system can be developed with easily available hardware, tool-chain and a small amount of programming. A variety of sensors can be attached to this system for the intended application, and the system can be programmed to do the required processing on the data before passing it to the synthesis system over Ethernet.

This system allows for a simple, low cost, easy to build, and easy to program solution for data-acquisition. Such a solution is versatile as well as robust for many different types of application in sound synthesis control. Further more it allows for a great control over the acquisition and processing of data.

CHAPTER 5
Communication Subsystem

5.1 Overview

This chapter discusses the design and development process of the Communication system. Firstly, the use of OSC as the communication protocol is discussed. Then the design and the development of the Java OSC Extension implementation of OSC are examined. And finally, an overview of the OSC communication system on the AVR-Mini based data-acquisition systems is done.

5.2 OSC Communication System

OSC is the most appropriate communication system for the type of control applications required. It is extendible and medium-independent. Most of the implementation of OSC in various languages and on various platforms is complete and easily implementable.

The JavaOsc implementation, however, has a few limitations. JavaOsc [27] is a based on a single thread and thus risks the loss of messages in certain conditions. Furthermore, it uses linear searching algorithms for parsing. These searches are slow and tend to reduce the speed at which a message can be dispatched. Moreover, the package as a whole is somewhat difficult to use and lacks a clear and intuitive interface. Finally, a lack of documentation for the API requires examining source code in order to understand how to use it.

Since target synthesis of this project is in Java, the Java implementation of the communication system needs to be complete and practical. Thus, an extension to JavaOsc is designed and developed.

5.3 Java OSC Extension(JOE)

5.3.1 Design of JOE

JOE is an attempt to build upon the design of JavaOsc and improve in the areas it lacks. JOE is based on an OSC Server design as defined in the OSC specifications. This OSC Server is responsible for extracting, parsing and dispatching any OSC Messages that it reads from a UDP socket. Any application, such as a synthesis engine, requiring the use of the OSC server, instantiates the OSCServer class and listen for the OSC Address Patterns that they are interested in. Once an OSC Packet is received, the server extracts the contents, parses the OSC Address and dispatches the message to the application if it has registered to be listening for such an address.

Thus, the OSC Sever consists of three major sections; a) the receiver and extractor, b) the parser and c) the dispatcher (see Figure 5). The three functions are mutually independent and can be separated into individual threads of execution.

Sending of OSC Messages does not require a server and can be done directly by the application at any given moment. This functionality is left unchanged from the original JavaOsc implementation.

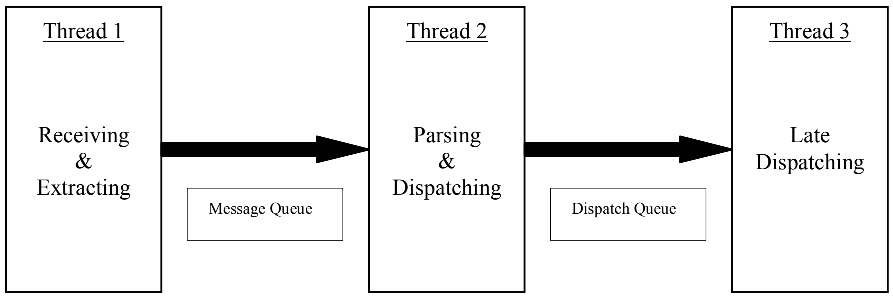

Figure 5: Multi-threaded structure of the OSCServer.

5.3.1.1 Message Handling

The original JavaOsc is based on a single thread. The fetching and parsing, as well as the dispatching are all done in the same thread. This can lead to incoming OSC Messages being missed, since the server does not have a chance to receive the next message until the current message is completely dispatched. If a certain synthesis engine has a dispatch callback method that takes a long time to execute, then by the time the thread returns to allow the server to retrieve the next message from the buffer, it may have been already overwritten by yet another following message.

To overcome this problem, JOE is made multithreaded. It has separate threads to receive new OSC Messages, and to parse the incoming message. The threads are linked through a queue that transmits OSC Message data structures to the parsing thread. This reduces the amount of time a message needs to wait before it is fetched, and thus the likelihood that it will be overwritten by another incoming message.

Incoming message bundles can be time-stamped (described later in more detail). If an OSC Bundle is received which is time-stamped to be dispatched in the future, the messages in the bundle are queued into a synchronized priority queue. In this queue, the priority is defined by the dispatch time. The OSC Message with the earliest dispatch time has the greatest priority. This queue links to another thread, which waits and dispatches the messages on time. In the case of an OSC Bundle that arrives later than its stipulated dispatch time, the synthesis engine can configure the server to either dispatch it immediately or discard it.

5.3.1.2 Argument Handling

JOE supports all the Atomic Data Types defined in the OSC specification. It also supports other commonly used argument types like 64-bit Integer, 4 byte character, TRUE/FALSE, and 64-bit Double. Type tag strings are enforced and messages without type tags are discarded. This is necessary since in such a generic implementation of OSC, it is impossible to predict the type of an argument.

Since the arguments are stored in Java Objects of the various types, they are dispatched as a Java Object array. JOE provides helper methods in the abstract listener class to discover and retrieve the type of the arguments passed on to the synthesis engine.

5.3.1.3 Parsing

The original JavaOsc implementation uses a heap data structure to store the OSC Address Space – simple list of all of the addresses the application handles. The elements of the heap are then individually read and compared to the incoming OSC Message's OSC Address Pattern. This linear storage and comparison is inefficient in terms of the amount of comparisons needed to reach the desired element. JOE addresses this limitation with a better data structure used to store the parsing strings.

Furthermore, JavaOsc implements strict string matching using the isEqual() method on strings treating OSC wildcards as string literals. This means that the use of wildcards is not supported by the server so that the synthesis engine has to handle them.

5.3.1.3.1 Tree Based Storage Structure.

JOE uses tree type data-structures to store the addresses spanning the OSC Address Space. OSC Addresses have similar syntax to URLs and thus span a tree-like address space. Implementation of such a space using a tree data-structure is not only intuitive, but also efficient. A Multi-Way tree, a generic N-nary tree, is used in JOE to keep track of the Address Space (See Figure 6). Such implementation reduces the number of comparisons needed to reach the element of interest.

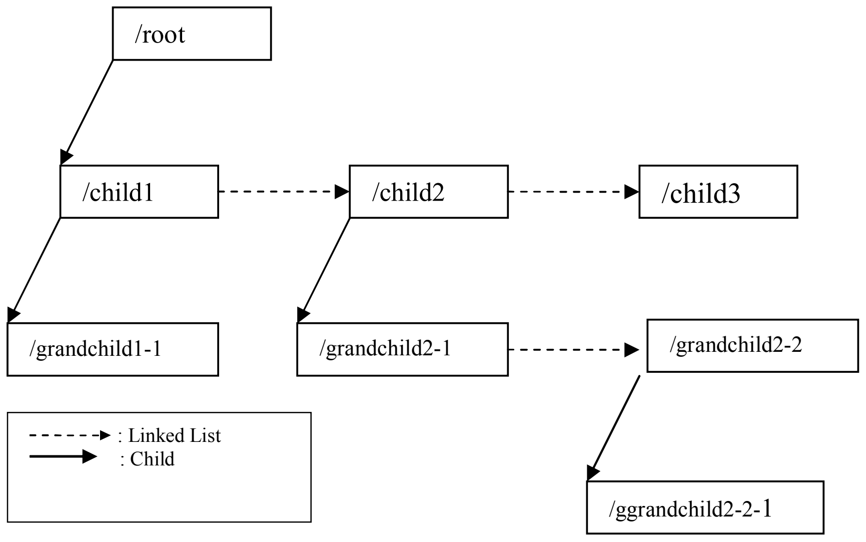

Figure 6: Multi-way Tree Structure.

Since multiple addresses might match an incoming message, the unordered list of addresses used in JavaOsc also means that the whole list must be traversed for matches to each incoming message. In JOE, the ordered insertions of new addresses allow further improvement on traversal complexity, since lexicological comparisons allow searches to complete without traversing the entire tree. Although these increases the time necessary for insertions, insertions are typically done prior to real-time operation, and when they are not, they are still generally far less frequent than search traversals, so that this strategy yields a better overall performance.

5.3.1.3.2 Intelligent OSC Parsing

JOE parses the OSC Address Patterns of incoming OSC Messages intelligently. OSC Address Patterns can contain wildcards as defined by the OSC Specifications. Such wildcards can match to multiple addresses. JOE uses regular expressions to match the OSC Address Patterns to nodes in the address space.

However, since the syntax for wildcards in OSC and regular expressions is different, JOE converts the OSC wildcards into the equivalent regular expressions wildcards. This allows for simple and elegant matching algorithms.

JOE also handles brackets in OSC Address Patterns. Brackets are opened up by the parser and converted into individual strings without the brackets. They are then parsed individually. Address patterns with incomplete or incoherent brackets are discarded.

Implementation of wildcards and brackets allows for the creator of the address space to be abstracted from the knowledge of the OSC Address format and especially the significance of wildcards. Wildcards permits the controller to send fewer messages and yet attain the same effect.

5.3.1.4 Blobs

An OSC Blob is a data type used to support a generic array of binary data. It is one of the Atomic Data Types defined in the OSC specification. Blobs, however, are not supported by JavaOsc.

Blobs are useful to applications that need to implement unsupported data types and extend the capabilities of OSC. The OSC Blob is implemented in JOE as a Java class. It is also supported as an argument to the OSC message and appropriately extracted out to the argument array.

5.3.1.5 Time Tags and Dispatching

OSC uses OSC Time Tags to define when a certain message gets dispatched. JavaOsc does not handle time tags by itself. Time tags are extracted and handed over to the synthesis engine to be dealt with. Some real-time synthesis engines may not be capable of handling time-stamped events, and furthermore, if a message is sent to multiple synthesis engines, then the dispatching effort must be duplicated in each of them.

JOE handles time tags on the server. Messages tagged to be dispatched at future times are held in a queue and dispatched at a later time from within JOE itself. Synthesizers can optionally request that future events be sent immediately when JOE receives them.

5.3.1.5.1 OSC Time Tags

Time Tags are only attached to OSC Bundles. The time tags are based on the NTP Timestamp protocol [49] and represent a range of 136 years with a resolution of 236 picoseconds. Using time tags, a controller can attain fine grain temporal control over the synthesis engine, even if the delivery mechanism does not guarantee temporal accuracy.

Nested bundles cannot have time tags indicating earlier dispatch time than the parent bundle. In JOE, such messages are considered erroneous and discarded. In some cases it might be necessary to ignore Time Tags. This configuration option on the server is available to the user.

5.3.1.5.2 Java Date Object and Accuracy

In JavaOsc, Time Tags are internally represented by a Java Date object which uses a millisecond resolution. This allows the easy implementation of time tag comparison while dispatching. The drawback of the conversion is the loss of accuracy compared to the NTP timestamps.

In most synthesis environments, however, accuracy of events of up to a millisecond is adequate. Furthermore, since these time tags are compared with a reference clock, the clock itself has to provide time with as much accuracy as the time tag. Java's system time clock relies on the operating system for time keeping. Generally, operating systems do not generally provide clocks with greater than millisecond accuracy making the NTP picoseconds information irrelevant.

5.3.1.5.3 Synchronization of Clocks

For the effective use of time tags, the clocks across the participating platforms involved have to be synchronized. This question does not arise when the control application and the synthesis engine are on the same physical machine.

Protocols like NTP allow accurate synchronization of system clock to the internet time servers. Most of the modern operating systems have the feature of allowing such synchronization. However, NTP synchronization can be hampered by network transport layer delays and thus can only provide accuracy of about 300ms.

These problems can be avoided by sending OSC Messages well in advanced such that they will reach the OSC Server well before they have to be dispatched.

5.3.1.5.4 Late Dispatching

Late dispatching of messages is required when a parsed message is found to have a time tag with a future dispatch time. Such messages are stored in a priority queue and dispatched when their time comes. The dispatching is done by the late dispatch thread. The thread checks the highest priority element in the queue at a frequency of one millisecond, and waits for its dispatch time to arrive before dispatching it to the application.

5.3.1.6 Listeners and Interfaces

The OSC Server communicates with the synthesis engine through listeners. Listeners are a standard Java feature used to implement callback methods. JOE uses the same type of listeners as JavaOsc, the OSCListner interface.

When the synthesis application wants to receive a certain OSC Message, it registers a listener with the OSC Server. This listener will be activated by the OSC Server when that specific message arrives.

The OSC Server allows dynamic registration and de-registration of listeners. It also supports multiple listeners for an individual address. This allows the same

OSC message to be dispatched to different parts of the same synthesis engine, or even different synthesis engines.

Listeners are easily implemented using Java Interfaces. JOE uses the OSCListner interface defined in JavaOsc. This backward compatibility allows most applications using JavaOsc to switch to JOE with minimal change.

5.4 OSC on AVR-Mini

Since the AVR-Mini based data-acquisition platform supports C language programming, the OSC library for C [29] can ported to that platform without much efforts. However, for an easy to use application of the data-acquisition systems, an API is developed.

This API provides a simple interface for building an application that sends and receives OSC messages in UDP packets over Ethernet from an AVR microcontroller. This can be combined with powerful AVRLib functions to create a device able to generate and send OSC messages based on sensor inputs or other processes. It is based on uIP-AVR v0.90 and the reference OSC library [29]. uIP-AVR is a port of uIP, an open-source TCP/IP stack designed to provide connectivity to embedded 8-bit microcontrollers. It supports many of the basic Internet communication protocols like TCP, UDP, ICMP, and ARP.

5.4.1 MOE API

The Microcontroller OSC Ethernet (MOE) API provides an interface to shield the details of the uIP stack, NIC drivers and network protocols from the user. A single header file contains flags and configuration options including the port addresses used to interface to the NIC, maximum number of connections, local and remote IP addresses and UDP port numbers. It is configured by default to work with the Easy Ethernet AVR board.

The user has to implement two functions, and is given guarantees of their execution time and order. The app_init() can be used to initialize the application internal and external components. The app_main() function is called in a tight loop by the uIP stack, as fast as the execution speed permits. Library functions allow the user to check if OSC messages were received, and to bundle and send OSC messages. The following 8-line example program is an 8-channel data-acquisition system sending 10-bit ADC values as OSC in UDP packets.

```
OSCbuf osc_buf;              // OSC buffer
char type[] = ",ii";         // OSC type string
char msg[] = "/sensor";      // OSC address

for (i=0; i<8; i++) {
OSC_writeAddrAndTypes(&osc_buf,msg,type);
OSC_writeIntArg(&osc_buf,i);
OSC_writeIntArg(&osc_buf,a2dconvert10bit(i));
sendOSCBuf(&osc_buf);
```

Figure 7: A simple microcontroller program.

5.4.2 API and Library

The choice of an API implementation for the embedded OSC software provides some constraints over the types of applications that the user can develop. Compared to a straight API-less function library, there are limitations imposed by the structure of the program. Other complex network tasks such as DHCP for IP address acquisition or serving web pages via HTTP are entirely possible with this hardware and software, but are excluded from the MOE API. This is chosen to limit the networking paradigm for the sake of simplicity, but leave the full power of the microcontroller's memory, timing, ADC, hardware and peripheral interfaces available for complex sensing and feedback tasks. A library implementation would make advanced network tasks available alongside these, but would require the user to understand and construct a complete network stack from scratch or modify an existing one.

5.4.3 Capabilities of MOE API

The benefits of a simple API implementation justify the constrained functionality. A software library alone would render the system simply unusable for many users. The Microcontroller OSC Ethernet (MOE) API provides a robust interface with a fast learning curve that limits the possibility of errors. The user does not require any detailed knowledge of the inner workings of the network stack in order to use the API. These components are available for inspection, but do not require modification. Since the MOE API is built on top of open-source software libraries, the underlying system code and libraries are always present for advanced users in order to build their own customized network systems.

In addition to the powerful microcontroller features it makes available, the MOE API also still offers more options than are available on many other data-acquisition systems, in spite of its constraint. It includes support for multiple UDP connections with different hosts or different ports on the same host. The OSC addresses and message formats are customizable, and it can send OSC bundles with timetags. Support for multiple incoming connections is also available.

Parsing of incoming OSC messages is presently not provided, but can be made available. One caveat of receiving OSC is that string operations can significantly slow down processing time; however, using short messages and simple OSC address spaces the delay can be avoided and 'real-time'ness ensured.

There is significant overlap between the functionalities of uIP-AVR and MOE. For the sake of creating a simpler API some parts of uIP-AVR were either removed were extracted as configurable options at compile time. TCP support is one such example. Most OSC-capable PC applications only support UDP as the data-link layer protocol [23]. This is because it requires less overhead and users are willing to accept the risk of packet loss for the sake of faster, unverified delivery. TCP functions are available in MOE and may be included by setting a configuration flag. Excluding TCP support saves a significant amount of code space for the user's application.

5.5 OSC Communications

OSC can be used as a communication system for such control purposes. OSC can be implemented on various platforms, in various languages with different foot print sizes. And yet, it allows great control over the communication of the control and feedback data over all its implementations.

CHAPTER 6
Java Mapping Tool

6.1 Overview

This chapter looks at the mapping subsystem. First it looks at how mappings can be evaluated. Next, a novel mapping technique is discussed. Finally, the implementation and a tool created to generate such mappings are discussed.

6.2 Creating and using Mappings

An important separation exists between, making and using the maps, which needs to be addressed. The tool being developed should allow the creation, audition as well as the use of the maps or ‘mappings’. The map has to be created, by a user, or ‘the creator’. It can be auditioned while being created. The map can then be used by the same user or by someone else, ‘a user’.

6.3 Mapping as transformations

Through out the literature about mapping, a basic idea can be seen repeatedly, described in various ways. Mapping is the transformation of data from one domain to another. The way this transformation is done is dependent on the mapping scheme being used. However, there are three important aspects the transformation which can be extracted as being important in for this project. Mapping is the transformation from control parameters to synthesis parameters. Mapping is the transformation form a low dimensional control space to a high dimensional control space. And finally, mapping is a transformation from a Perceptual Sound Space to a Parameter Space. The model in figure 8 exemplifies these three concepts.

The Sound Space is the perceptual sound space as a human user envisions the synthesis system exposing. This is the space of all sounds which the user's idea of the synthesis system can produce, and their relation with each other.

The Control Space a space of control signals or control data which the user should have to input via the controller, either gestural or otherwise, to produce the types of sounds he believes he can get. This should directly correlate with the perceptual sound space, so that the user can easily control the synthesis.

The Parameter Space is the space exposed by the sounds models in the synthesis engine for control. This space can be, and is generally different from the Control Space and might not correlate, in dimensions, or data types. A mapping should transform the control data from Control Space to the Parameter Space.

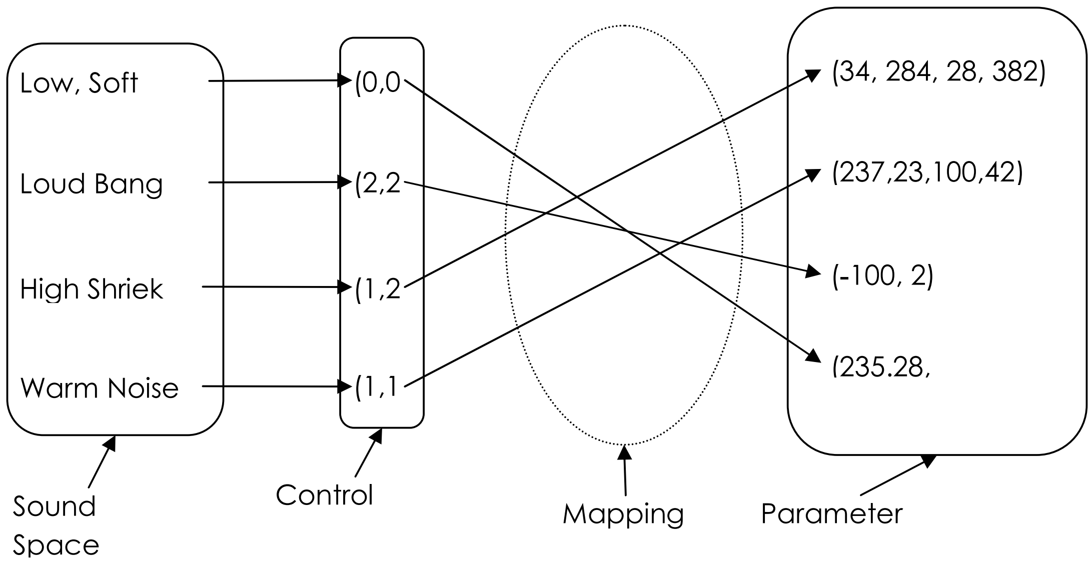

Figure 8: Conceptual Representation of Mapping

6.4 Evaluating Mappings

Three important criteria can be defined for analyzing mappings: complexity, expressivity, and intuitiveness. There is certainly a strong relationship between a mapping technique and the degrees to which these properties are reflected in the mappings that are generated. One could conceive of quantitative metrics for assessing these, but this is beyond scope of this thesis. These can be left as subjective assessments in order to create a vocabulary for discussing and informing the design of mapping techniques.

The complexity, especially the computational complexity of mappings, is an important factor when designing mapping for a real-time audio synthesis engine. Complexity can be conceived as related to the number of computational or logical steps required to generate a synthesis parameter from an input control change. Expressivity refers to the ability of the user to generate a rich variety of sonic output from the synthesizer, under deliberate control. Mappings resulting in output that on one hand might “always sounds the same” or on the other hand feel “random” or “uncontrollable” from the user's perspective are not expressive. Intuitiveness in a mapping can be seen as related to the user's mental model of the mapping [50]. If the user requires only a simple conception of how the mapping works, then it can be considered intuitive. With an intuitive mapping, the perceptual sound space conforms closely to the parameter space of the sound model.

These three criteria are not always independent. Intuitive mappings are more likely to be expressive since the user can more easily comprehend the system. Excessively non-complex mappings are likely to be intuitive, since the user may be able to have a complete mental model of exactly how the system works (a “surrogate model”, in Young's [51] terms). However, these may not be expressive at all. Take the following system for example: two knobs generating 0-5V control signals which are independently mapped to frequency (20-20000Hz) and amplitude (0-90dB) of a sine wave oscillator. This mapping is extremely intuitive, which means to say, a novice user can comprehend it almost instantaneously and very simple, but not likely to be very expressive, at least to a new user.

In spite of these apparent relations between the criteria, a number of similar examples easily show that they are not deterministic, and therefore the criteria must also be considered independently. For example, complex mappings may in fact be quite intuitive. Imagine a hypothetical control system that extracts vectors of 3-D position data over time, based on the position of a user's finger in space. This position data is mapped to parameters of a speech synthesis system that “speaks” based on the letters that are being drawn in the air. This requires an extremely complex mapping, probably first mapping the position data to letters, and then mapping these to a large number of synthesis parameters in order to generate speech sounds. However, it is quite intuitive; all the user needs to do is write in the air, and the mapping converts it to speech. And such a system is capable of a broad range of expression. Obviously, the complexity of such a system is beyond anyone's present capability (at least to produce human-sounding speech).

Strategies involving linear algebra and matrix transformations [32] are generally complex as well as unintuitive, form the user's perspective. However, the expressiveness of such mapping is undeniably exhaustive, as with such transforms, one can describe any linear relationship between the input and output space. However, such relationships are difficult for the user to conceive, and thus even more difficult to manipulate and explore.

Multi-layered systems are often successful at making complex mappings more intuitive. They can provide the user with a more useful conceptual model than what a direct mapping to synthesis parameters may provide.

The goal in creating a successful mapping strategy is to achieve an optimal combination of the three criteria. A good mapping strategy should be intuitive and should produce expressive mappings, with the least amount of complexity possible in the mapping, but no less. These criteria should not only guide the way the mapping tool is designed from the user's perspective, but also how it is designed from the creator's perspective.

6.5 Parameterized Morphing

Morphing is a process used widely in image processing. It is used in animations and motion pictures to change from one image to another through a seamless transition. This is generally achieved by interpolating between certain common features in the initial and final images.

Audio morphing normally describes the generation of a transition between two segments of recorded or synthesized audio. As in image morphing, it is normally achieved by interpolating between sets of features that are determined after the sound has been rendered, based on analysis of the audio signal in the time, frequency, or perceptual domains. For example, Slaney [52] describes a technique for morphing between sounds by interpolating between extracted spectral shapes and pitch.

6.5.1 Morphing in Parameter Space

With parametric, real-time synthesis, one can appropriate the term morphing and extend its application to interpolation in the domain of the synthesis parameters themselves, since they are available at the time of sound generation [53]. If for a particular synthesis model, sound A can be specified by a vector of parameters $\mathbf{P_A}$, and sound B can be specified by a vector of parameters $\mathbf{P_B}$, then a morph between A and B can be generated by interpolating between $\mathbf{P_A}$ and $\mathbf{P_B}$. Of course, the extent to which this creates a perceptual morph depends on the nature of the model and on the interpolation functions.

6.5.2 Mapping by Parameterized Morphing

This concept of parameter-space morphing can be used in the context of mapping. The basic mapping scheme follows. For any synthesis model, sets of parameters are defined in advance, corresponding to particular desirable or interesting points in the model's sound space. A pair of parameter sets can be chosen and set to be the end points for the morph. The morphing control then chooses the extent to which each of the parameter sets contribute to the output of the mapping. The following figure shows a simplistic model of this concept.

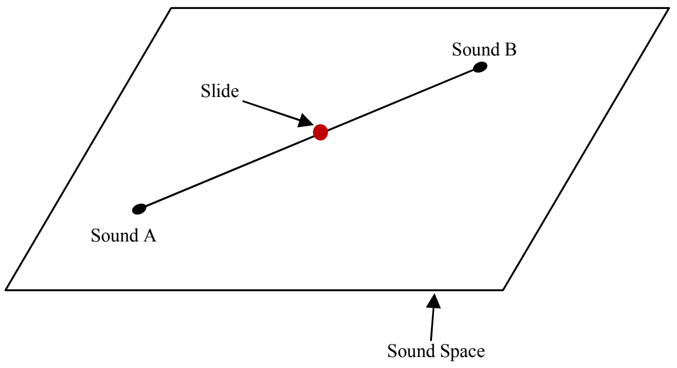

Figure 9: Simple model of morphing

This is achieved by with the use of some interpolating function between each common parameter in the parameter sets. The user can also set the parameters at the end point dynamically, thus exploring the synthesis space.

This scheme leads to a 3-n mapping, where the mapping takes in the 2 discrete control inputs, and a continuous control input, and yields any number of continuous synthesis parameter outputs. The 2 discrete control inputs are used to choose which parameter sets are to be the end points of the morph, by indexing them from a list. The continuous input, a slider, then determines the extent of the morph. (See Figure 9)

6.5.3 Interpolation

A simple linear interpolation scheme will reduce the mapping to a simple range modification function and thus, limit the utility and expressivity of the mapping. For example, while synthesizing the sound of a helicopter, if the desired control parameter of the mapping is the distance of the listener from the helicopter, the relationship between the input and the volume parameter of the model needs to follow the inverse square law. If the interpolation is limited to simple linear schemes, such common scenarios would not be addressable. Thus, a need for non-linear interpolation schemes arises.

An interpolation scheme in morphing determines the output value of a certain feature, or parameter in sound synthesis, according to the two values being interpolated between. In the general case, the mathematical expression for the interpolation would be.

$$\vec{P}_O = (1-\eta(s))\cdot\vec{P}_A + \eta(s)\cdot\vec{P}_B \tag{1}$$

Where $\eta(x)$ is the non-linear weighting function for the input, $\vec{P}_A$ & $\vec{P}_B$ are the two parameter sets to be morphed between, s is the normalised value of the slider and $\vec{P}_O$ the output of the morphing. The following figure shows a conceptual representation of the parametric morphing concept.

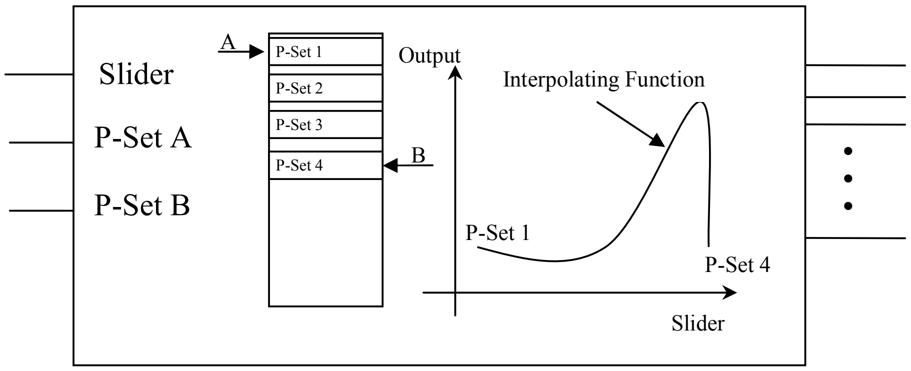

Figure 10: Conceptual Representation of Mapping

6.6 Additional Features

With this morphing scheme as a basis, a series of extensions can be added in order to design a usable mapping software system that attempts to satisfy the criteria of complexity, intuitiveness and expressivity. The objective is to design a mapping application that can operate with a variety of synthesis systems and controllers (physical or virtual). However, the large numbers of available synthesis systems and controllers present a challenge to design a single application that offers universal connectivity.

Therefore, a general-purpose mapping application is defined, around which different interfaces can be defined, both for incoming control signals and outgoing synthesis parameters. The front-end interface can transform the control signals to the appropriate internal representation of the mapping system; this is a pre-mapping. Similarly, the back-end interface transforms the internal representation of the generated synthesis parameters into a protocol and format supported by the receiving application. See Section 6.7.5 for further details of the implementation.

6.6.1 Partitioning and Maplets

An extension to the morphing scheme arose from the need to have more control over the mapping. If all the parameters are considered to span an *n*-dimensional space, then each parameter set of length *m* defines a *(n-m)* dimensional subspace of the original *n*-dimensional space. Morphing between two such parameter sets, the output defines a sequence of *(n-m)* dimensional spaces. Parameter sets allowed the shrinking of the output parameters space dimension from *n* to *(n-m)*, however, the remaining *(n-m)* output parameters become uncontrollable. Furthermore, this scheme cannot produce divergent mappings as defined by Rovan et al. [54].

To address both these limitations, concept of maplets is introduced. Maplets are objects which contain the ability to do a single parameterized morph as discussed in section 6.5.2 . A complete map can contain one or more maplets.

Maplets are based on the idea of partitioning the set of all output parameter into groups of parameters that are related, especially in their dynamics with respect to input parameters. For example, all parameter which change exponentially over a certain input parameter can be grouped together. Such groups can be assigned to individual maplets, thus, allowing each group to have its individual morphing functionality and inputs. maplets serve as the atomic mapping entity.

Furthermore, maplets allow having multiple pairs of parameter sets to be morphed over simultaneously. This is useful when controlling complex sound models with multiple sets of parameters closely related in their behaviour. This feature is also needed when different interpolation functions are needed to map different groups of parameters. For example, when controlling a sound model of a car, it is effective to control all the parameter related to the tyres separately from all the parameters related to the engine.

Finally, multiple maplets allow a single mapping to control multiple unrelated models using a single or multiple set of controls. Thus allowing single input to single model mappings, as well as mappings where single input can control multiple models. Such a scheme allows greater exploration of the sound space exposed by the models, and still keeps the mapping intuitive. Divergent mappings can also be generated by this scheme [54].

6.7 Implementation

A tool was developed to build mappings using the parameterized morphing discussed above. The aim in the development of such a tool is to have a user friendly and intuitive means for designing mappings, and to test the practicality and verify the concept of the mapping scheme.

6.7.1 Java

Java was chosen as the development language for this project. Java has grown significantly in scope and in popularity over recent years. Its mobility and portability have been incentives for its use in the development of a wide variety of applications for servers, PCs and mobile devices. The choice of using Java is mainly influenced by the ease of creation of Graphical User Interface (GUI) in Java and the available resources in Java for communication with other systems. Furthermore, the Java Synthesis System is written in Java [6], thus a Java based mapping tool can integrate easily with the synthesis system.

6.7.2 Modularity

Modularity is important in the design of the system, since many of the subsystems are individual components that can be used on their own in a variety of mapping-related applications. In order not to restrict further development of any of these subsystems and the ability to work autonomously, under various situations, a completely modular approach is taken in the development of this tool.

The core mapping functionalities are separated from the communication subsystem and the GUI. The core mapping functionality contains the interpolating and morphing logic of the mapping. It contains the maplets, the pre-defined parameter sets and the interpolation functions. However, it has no user interface and is only accessed through method calls. This helps to keep the core small and computationally efficient. Furthermore, such a module can easily be embedded into a synthesis system or models in a model based synthesis systems.

The communication subsystem is based on Java Interfaces, which are provided in various places in the core to allow communication subsystem to be attached to the core. The communication subsystem, often referred to as the I/O interface, can be designed and implemented by the user according to the type of communication required by the application. For example, if the tool needs to communicate with a microcontroller based gesture acquisition system, which can output data as MIDI messages, then the mapping needs a MIDI I/O interface that can identify MIDI messages related to the inputs exposed by the mapping and call the required methods.

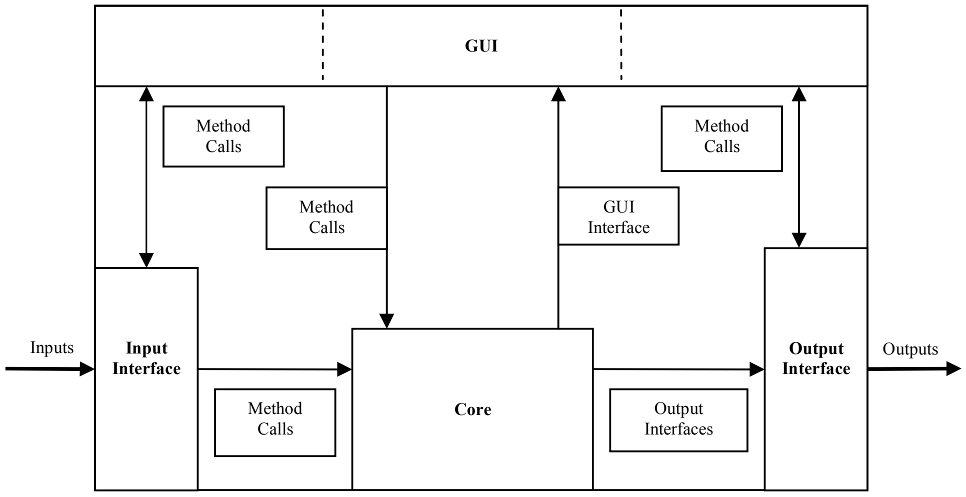

Figure 11: Structure of the Mapping Tool

The GUI allows the user to use the tool interactively. It provides functionality to test the mapping that are created and attach interfaces to the mappings. It also gives the user a view of the current state of the system and the ability to store and load maps dynamically. This is done using the serializable property of classes in Java. As such, mappings can be interactively generated using the GUI and saved. They can then be loaded into other systems, including systems on which having a GUI is not possible.

6.7.3 Sound Models

The tool uses the abstract concept of sound models for mapping sounds. Every parameter being control by the tool needs to have some physical correspondence, such that it can be controlled effectively. This is attained by forcing each parameter to be a part of a sound model. Thus, sound models are logical representations of physical sound models that are being controlled by the mapping. They may or may not correspond to actual Sound Models (see section 1.4). Such models also allow modularisation of complex physical sound models as they can be mapped to multiple sound models in the context of the tool, thus making the partitioning of parameter sets simpler and straight forward.

6.7.4 Interfaces and OSC

The mapping core itself is designed to only communicate using method calls. Thus, there is a need to allow various forms of interfaces in order to use the mapping scheme in various applications, especially with plethora of synthesis systems and controllers available to be supported. Since most of these systems communicate with their own protocols, it seems reasonable to allow the user to implement an interface and attach it to the mapping tool. Such interfaces form the Pre-Mapping and Post-Mapping layers (See Figure 4).

In addition, an OSC based interface is developed as the default interface system for the tool. OSC provides a simple, extensible and transport-layer-independent protocol for communications [22]. Such a protocol is useful to support communication with various other software and hardware platforms, including synthesizers, microcontroller based data-acquisition modules and hardware controller platforms. Furthermore, the availability of OSC libraries on many platforms and in many languages, allows for easier development of interfaces on the synthesiser [28, 29, 30]. JOE was used to create the Input Interface for the mapping tool.

6.7.5 GUI

With intuitiveness and expressivity of mapping an important feature of this scheme, a simple and easy to use GUI, is required. This GUI allows the user to visualize the various elements of the mapping and also to interact with these elements in real-time. Figure 12 shows screenshot of the main GUI screen.

6.7.5.1 Maplets List

The maplet list GUI gives the user a view of the currently active maplets in the core. It shows which parameter sets are selected in each of the maplets. It also shows the current value of the slider, using a slider graphical object. This graphical representation of the maplet, tries to conform to the mental idea of a maplet, thus making the mapping more intuitive to the user.

The maplet list GUI also allows the user to interact with the map. The user can change the Slider and the Parameter Sets that are being used and hear the difference real-time since the output is computed at every change of the GUI. There is also additional functionality in the maplet lists to allow the user to decouple the GUI from the core. This allows the user to change the GUI and yet not affect core and vice-versa. This functionality can also prove useful as a trigger utility, by pre-setting the sliders and parameter sets and then coupling the GUI and the core.

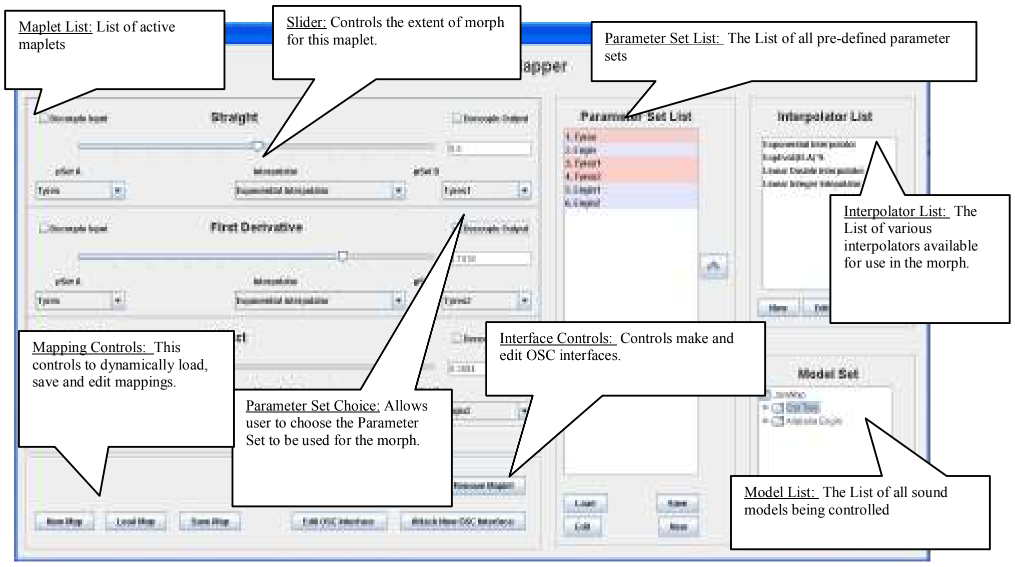
Maplet List: List of active maplets
Slider: Controls the extent of morph for this maplet.
Parameter Set List: The List of all pre-defined parameter sets
Interpolator List: The List of various interpolators available for use in the morph.
Mapping Controls: This controls to dynamically load, save and edit mappings.
Parameter Set Choice: Allows user to choose the Parameter Set to be used for the morph.
Interface Controls: Controls make and edit OSC interfaces.
Model List: The List of all sound models being controlled
Straight
First Derivative
Parameter Set List
Interpolator List
Model Set

Figure 12: GUI Main Screen.

6.7.5.2 Parameter Set Maker

Creating parameter sets is an important part of this mapping scheme. The GUI allows the user to create parameter sets corresponding to particular desirable or interesting points, interactively.

This ability to hear the synthesis output while designing the parameter sets of the mapping improves the intuitiveness of the mapping scheme, since the user is able to create a mental model [50] with the help of the sound created by these parameter sets.

6.7.5.3 Interface Maker

The interface maker allows the user to set up the input and output interfaces for the mapping. Since the default interface uses OSC, this GUI element is specifically implemented to setup an OSC interface. This interface also implements a pre-mapping layer, which allows the input value of the slider to be to be normalised so that natural range of the controllers can be accommodated. This is achieved by forcing the user to set a minimum and maximum value of the expected input. The value is normalised internally before being passed on to the core. This ensures correctness of data being fed to the core.

6.8 Mapping

The mapping tool, called Java Mapper, developed during this project, implements the mapping technique of parameterised mapping proposed by this project. The technique yields a novel approach to mapping, which is intuitive, expressive and yet not too complex. The tool allows the users to create and use the various types of mappings generated. The various components of the tool are developed to allow for easy creation and flexible usage of the mappings.

CHAPTER 7
Evaluation

7.1 Overview

In this chapter the, implemented systems are evaluated based on their performance under testing situations. Consequently, an application usage of the whole system is discussed.

7.2 Performance

7.2.1 Data-Acquisition System

The performance of the data-acquisition system depends on the resolution of the sensors, the frequency of sampling and the speed at which data can be processed. The resolution of the sensors depends on the type of sensor and the ADC, if any is used. The sampling frequency and the processing speed depend on the microcontroller.

To test the performance of the data-acquisition system, a test application was built using the Easy Ethernet AVR with an ATmega32 and MAX127 I^2C ADC [55] to do a single channel of ADC and send out the result as OSC over UDP. The conversion result was encoded as an argument to a single OSC message. The average incoming message rate measured over a period of several minutes was over 1300Hz. While the raw ADC conversion speed of the internal ADC was 148 kHz.

Sampling rates such as these suffice the target usage of the system. Thus, the microcontroller based data-acquisition system performed as expected while running the OSC based communication protocol and using internal as well as external ADCs.

7.2.2 JOE

Performance tests were done to compare the performance of JOE with JavaOsc, since the aim was to improve the performance of JavaOsc. In these tests, a test application was created to set up the OSC Server and register several dummy listeners. These listeners simulated the processing time taken to handle a message by waiting for a random amount of time and then printing out the integer argument of the message received. On a different machine, OSC Messages were sent out using pd [26]. Pure Data (pd) is a real-time graphical programming environment for audio, video, and graphical processing known for solid timing characteristics. Each message was given an integer argument according to its sending order. This allowed us to check if all the messages arrived at receiver. Using this setup, the ability of JOE and JavaOsc to handle a barrage of messages, including those with wildcards triggering multiple dispatches, delivered at various rates was tested.

JavaOsc fails to receive messages in certain cases. It was observed that when the periodicity of incoming messages is greater than the time taken to dispatch a message, packets are lost. Initially for a few packets there were no lost packets, however, after some time (depending on the rates of sending and dispatching) some packets failed to arrive.

Java Socket classes allow messages to be buffered internally. However, when these buffers overflow, older messages are overwritten by new ones. Such overflows are bound to happen when the processing time of messages is longer than the rate at which they arrive. The large size of these buffers allows temporary resolution of the problem, but prolonged usage will definitely lead to message loss. On PC or Mac based platforms, the problem can often be addressed by increasing the buffer size. However, on mobile platforms where memory is not freely available, small buffer sizes could be an issue.

Using JOE, under no circumstances were any messages lost, even with wildcards that triggered multiple dispatches. When Time Tags with "future" times were used, JOE delivered messages to applications with the targeted millisecond accuracy.

7.2.3 Java Mapper

The performance of the Java Mapper is a subjective concept, and dependent on many external factors like the controller and synthesis systems. It might work well with certain type of synthesis system and even specifically certain models. There is no quantitative analysis that can be done to evaluate the performance of Java Mapper. However, the mapping scheme can be evaluated, using the criteria defined in Chapter 6.

7.2.3.1 Multi-Layered Mappings

The parameterized morphing based implementation represents a multi-layered strategy, in that the notion of parameter sets represents an intermediate construct between the input and output parameters. One feature of this, as Wanderley [7] highlights, is that the mappings themselves, from parameter sets to synthesis parameters, can persist regardless of the input parameters. Parameter sets can also be reused in different maps and maplets. From the user's perspective, the concept of parameter sets also simplifies the mental representation of a potentially large space of synthesis parameters. Once parameter sets representing desirable points in the synthesis space have been defined in an off-line process, the user can ignore the target parameters themselves, and rather conceive of two sounds and a morph between them, thus making the mapping intuitive.

7.2.3.2 Convergent and Divergent Mappings

Mappings can be classified as one-to-one, convergent and divergent [53]. A good mapping scheme allows for all three types of mappings to be defined. The parameterized morphing based mapping scheme, allows one-to-one, as well as divergent mapping. However, it is unable to produce a convergent mapping. Such a mapping does not fit easily into the morphing framework used in this mapping scheme. If there is indeed a need for simple convergent mapping to be coupled with this mapping scheme, they can easily added as the pre-mapping layer to this mapping scheme.

7.3 Application

7.3.1 Microcontroller based Controller and a Mapping Server

To showcase the usability of the whole tool chain and as a proof of concept, an application was setup to use all the parts of the control system. A prototype AVR microcontroller based data-acquisition system, was the front end of the control. It provided knobs and buttons as interfaces to the user to control the synthesis. The communication was done over OSC. The microcontroller created OSC messages corresponding to the change of every interface, and sent them over Ethernet.

This was connected to a PC running the Java Mapper server. The server was pre-configured, by auditioning and listening to the sounds of various parameter sets on the same system. The server read the inputs using JOE and mapped them to the corresponding outputs. Every time incoming OSC message caused an output parameter to change, another OSC message was created and sent over Ethernet to another PC. This machine ran the Java Synthesis System. JOE was attached as its communication front end. Thus the messages were parsed and the synthesis parameter manipulated. Figure 11 shows a representation of the system setup.

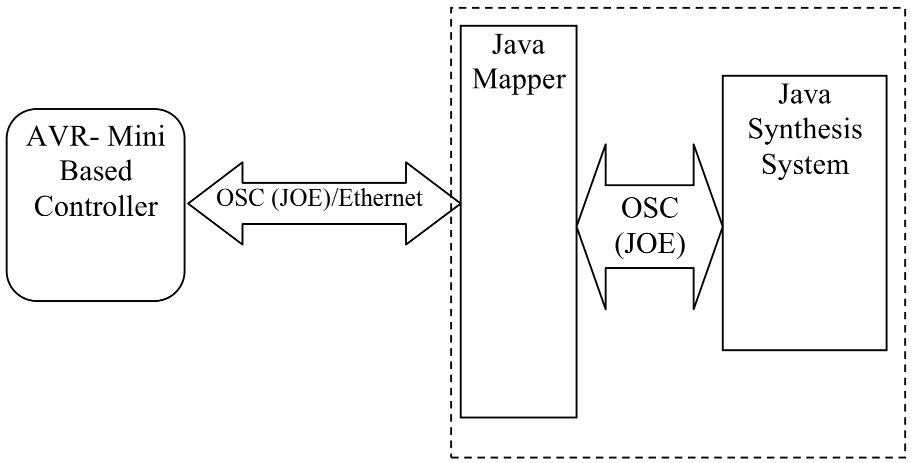

Figure 13: Representation of the first setup

This setup showcased the ability of the system to remotely as well as effectively control the synthesis. This also proved that the system especially the mapping was able to work as a distributed system, on various platforms. The various modules of the control system were able to work synchronously and without any glitches.

7.3.2 Software Controller and an Embedded Mapping

Another setup was tested, using software based controller, and an embedded mapping scheme. The software based controller was implemented using pd on a PC. A pd patch was created which output OSC messages according to the inputs that can be changed using the GUI. For testing purposes, a human user manually changed these values. The OSC messages were sent out over Ethernet to the same PC, using a localhost feedback. Although other method of transmission of OSC messages could have been used, this was the simplest and did not require any changes to the systems. Figure 14 shows a representation of the system setup.

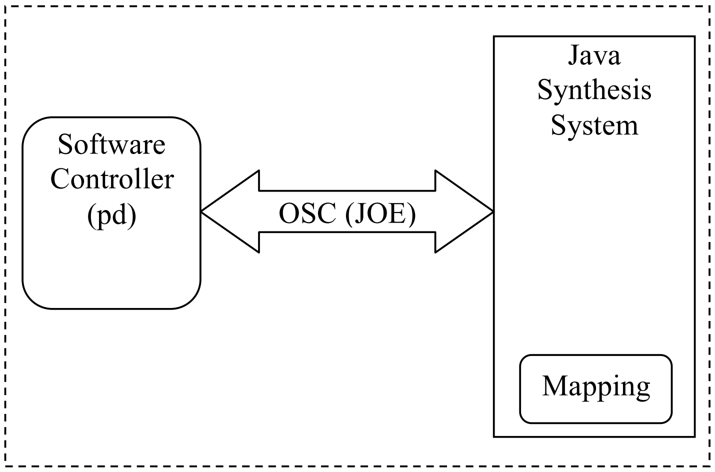

Figure 14: Representation of the second setup

The OSC messages were received by Java Synthesis System with JOE as the communication front end. The mapping was embedded inside one of the sound models loaded by Java Synthesis System. This mapping was created using the Java Mapper tool, using its GUI. It was saved as a file and loaded inside Java Synthesis System while creating one of the models. This new model itself controlled multiple smaller models using the mapping, which it defined.

This application showcases the flexibility of the implementation of the mapping scheme as well as the flexibility of the communication system. The system allows the mapping to be stand alone server on remote systems as well as embedded into models to produce higher level sound models which can be used by the synthesis system itself to gain better control over the synthesis.

CHAPTER 8
Conclusion

8.1 Conclusion

Synthesized sound has a lot of potential to become a standard in many audio applications, from multimedia and games to compression techniques. However, the nature of most types of synthesis techniques makes synthesis difficult to control. The sheer number of parameters and their possible lack of direct relationship to tangible abstracts in the domain of the user is the limiting factor of the practicality of most synthesis techniques.

This thesis address this issue by designing and developing a control system that offers a more expressive and intuitive control over synthesis. This is achieved by a novel approach to mapping of parameters, using a parameterized morphing technique. Coupling this with appropriate communication and data-acquisition systems, yields a flexible and yet effective control system. The modular design of such systems allows for usage of these systems in other areas related to synthesis. This also allows further development of each of these systems since they are not limited by other functionalities.

In conclusion, with better control techniques, sound synthesis can be used to attain its potential to offer newer and better sound and audio experiences.

8.2 Further Work

The modular design of the systems allows them to be improved individually, and still work in unison. The suggestions for future work on these systems are discussed individually.

8.2.1 Data-Acquisition System

Subsequent to the development of the system, especially MOE, several components of uIP-AVR have also been extracted and incorporated into the latest release of the AVRLib. The consistently clean and efficient libraries of AVRLib can help improve the readability and efficiency of MOE. This could also offer greater capabilities to MOE.

There is also a possibility of using interrupt based function calls to emulate multiple application threads, rather than polling in a tight loop. Synchronization of the tightly-coupled threads is a major hurdle that needs to be overcome. The use of lightweight "protothreads" which have recently implemented for AVR can be a potential alternative implementation.

8.2.2 JOE

There are a few areas for improvement that can be looked at in further work for JOE. Clock synchronization of the various platforms on which the system can run is necessary for accurate control of synthesis. This could be done by using NTP and having the OSC Server keep its own high resolution clock. Drift needs to be addressed in this case.

The speed of parsing can also still be improved. Using more efficient data structures and some data structure conversions mechanisms such as converting Multi-Way trees to Binary trees, the increased speed of parsing, especially for very large address spaces, would make a difference.

Other areas of improvement include adding support for automatic exploration and configuration of JOE systems over a network, to allow plug-and-play type capabilities for such devices. Methods for this are discussed further in the next section.

8.2.3 Java Mapper

One of the possible refinements is to allow user specification of the interpolation function. A tool that allows users to graphically specify the shape of the interpolator in real-time will be a useful addition to the tool chain.

Currently, the OSC interface that can be attached to the core in Java Mapper must be manually specified by the user. This can extended with a tool that can automatically explore the OSC address spaces of both the controller and synthesizer. In the case of the synthesizer, the tool could generate an internal representation of the sound models and their parameters, based on the OSC address space. This will be a key application for the implementation of a proposed OSC query scheme [25].

Another improvement to the Java Mapper is a better, and more user friendly GUI, with simple yet useful addition like colour coding of parts for an intuitive use of the GUI, default OSC message address configuration and movable windows for ease of use.

REFERENCES

[1] Rabenstein R. and Trautmann L., “Digital sound synthesis by physical modeling” in Proceedings of International Symposium on Image and Signal Processing and Analysis (ISPA’01) , Pula, Croatia, June 2001.

[2] Chowning J.M. “Digital Sound Synthesis, Aoustics, and Perception a Rich Intersection in Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), Verona, Italy, December 7-9, 2000 DAFX-1

[3] Karjalainen M, Välimäki V and Jánosy Z, “Towards High-Quality Sound Synthesis of the Guitar and String Instruments” in Proceedings of International Computer Music Conference, September 10-15, 1993, Tokyo, Japan

[4] Karplus, K; Strong, A “Digital synthesis of plucked-string and drum timbres.” Computer Music Journal, 7(2), pp. 43-55. 1983

[5] Desainte-Catherine M. and Marchand S. “Structured Additive Synthesis: Towards a Model of Sound Timbre and Electroacoustic Music Forms”. in Proceedings of the Inter- national Computer Music Conference (ICMC’99, Beijing), 1999.

[6] Wyse L. “A Sound Modeling and Synthesis System Designed for Maximum Usability” in Proceedings of the 2003 International Computer Music Conference (ICMC2003), Singapore, 2003.

[7] Wanderley M. and Depalle P. “Gestural Control of Sound Synthesis” in Proceedings of The IEEE, Vol. 92, No. 4, April 2004.

[8] Miller, G.A. “The magic number seven plus or minus two: Some limits on our capacity for processing information”. Psychological Review. 63, 2, pp 81-97, 1956.

[9] Jensenius A. R., Koehly R., and Wanderley M., “Building low-cost music controllers.” in Pre-Proceedings, 3rd International Symposium on Computer Music Modelling and Retrieval, pages 252–256, 2005.

[10] Building a USB sensor interface. Retrieved on 29th March 2006 from

http://www.sensorwiki.org/index.php/Building_a_USB_sensor_interface.

[11] I-CubeX. Retrieved on 29th March 2006 from http://infusionsystems.com/.

[12] La Kitchen. Retrieved on 29th March 2006 from http://www.la-kitchen.fr/.

[13] Fl´ety E. and Sirguy M.. "EoBody: a follow-up to AtoMIC Pros technology." in Proceedings of the 3rd International Conference on New Interfaces for Musical Expression (NIME03), 2003. Retrieved on 29th March 2006 from http://hct.ece.ubc.ca/nime/2003/.

[14] E. Fl´ety, N. Leroy, J. Ravarini, and F. Bevilacqua. Versatile sensor acquisition system utilizing network technology. In Proceedings of the 4th International Conference on New Interfaces for Musical Expression (NIME04), 2004. Retrieved on 29th March 2006 from http://hct.ece.ubc.ca/nime/2004/.

[15] Allison J. T. and Place T. A., "Teabox: A sensor data interface system." in Proceedings of the 5th International Conference on New Interfaces for Musical Expression (NIME05), 2005. Retrieved on 29th March 2006 from http://hct.ece.ubc.ca/nime/2005/.

[16] Fl´ety E.. "The WiSe box : a multi-performer wireless sensor interface using WiFi and OSC." in Proceedings of the 5th International Conference on New Interfaces for Musical Expression (NIME05), 2005. Retrieved on 29th March 2006 from http://hct.ece.ubc.ca/nime/2005/.

[17] Teleo. Retrieved on 29th March 2006 from http://www.makingthings.com/teleo.htm.

[18] Wilson S., Gurevich M., Verplank B., and Stang P., "Microcontrollers in music HCI instruction: Reflections on our switch to the Atmel AVR platform." in Proceedings of the 3rd International Conference on New Interfaces for Musical Expression (NIME03), 2003. Retrieved on 29th March 2006 from http://hct.ece.ubc.ca/nime/2003/.

[19] Gurevich M., Verplank B., and S. Wilson. "Physical interaction design for music." in Proceedings of the 2003 International Computer Music Conference, Singapore, 2003.

[20] MIDI Manufacturers Association. Retrieved on 29th March 2006 from http://www.midi.org/

[21] Wright M., Freed A., and Momeni A., “OpenSound Control: State of the art 2003.” in Proceedings of the 3rd International Conference on New Interfaces for Musical Expression (NIME03), 2003. Retrieved on 29th March 2006 from http://hct.ece.ubc.ca/nime/2003/.

[22] Open SoundControl. Retrieved on 29th March 2006 from http://www.cnmat.berkeley.edu/OpenSoundControl/

[23] Zimmermann H., “OSI Reference Model-The IS0 Model of Architecture for Open Systems Interconnection” in IEEE Transactions On Communications, Vol. Com-28, No. 4, April 1980

[24] Flety E., Leroy N., and Schwarz D., “EtherSense an OpenSoundControl-based sensor platform”, at OSC Conference, CNMAT, Paris, France, 2004.

[25] Schmeder A. and Wright M. “A Query System for Open Sound Control”, at OSC Conference, Paris, France, 2004.

[26] pd~ The Pure Data Portal. Retrieved on 29th March 2006 from http://puredata.info/about

[27] Illposed Software, JavaOsc. Retrieved March 21, 2006, from http://www.mat.ucsb.edu/~c.ramakr/illposed/JavaOsc.html

[28] flosc : Flash OpenSound Control. Retrieved on 29th March 2006 from http://www.benchun.net/flosc/

[29] The OpenSound Control Kit. Retrieved on 29th March 2006 from http://www.cnmat.berkeley.edu/OpenSoundControl/Kit/

[30] CNMAT, OpenSound Control in Max/MSP for Macintosh and Windows. Retrieved March 21, 2006, from http://www.cnmat.berkeley.edu/OpenSoundControl/Max/

[31] Chang, S. S. and Zadeh, L. A., “On fuzzy mapping and control.” in Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers By Lotfi A. Zadeh, G. J. Klir and B. Yuan, Eds. World Scientific Series In Advances In Fuzzy Systems, vol. 6.

World Scientific Publishing Co., River Edge, NJ, pp 180-184.

[32] Bevilacqua F., Müller R. and Schnell N., “MnM: a Max/MSP mapping toolbox” in Proceedings of the 2005 International Conference on New Interfaces for Musical Expression (NIME05), Vancouver, BC, Canada, May 2005.

[33] Momeni A. and Wessel D. “Characterizing and Controlling Musical Material Intuitively with Geometric Models.” In Proceedings of the of the 2003 International Conference on New Interfaces for Musical Expression (NIME03), Montreal, Canada, 2003.

[34] Nort D. V., Wanderley M. M., Depalle P. “On the Choice of Mappings Based on Geometric Properties.” in Proceedings of 2004 International Conference on New Interfaces for Musical Expression (NIME04), Hamamatsu, Japan, 2004.

[35] Arfib, D., J. M. Couturier, L. Kessous, and V. Verfaille. “Strategies of Mapping between Gesture data and Synthesis Model Parameters using Perceptual Spaces.” Organised Sound, 7(2) pp 127-144., 2002

[36] Hunt, A., and M. M. Wanderley. “Mapping Performer Parameters to Synthesis Engines.” Organised Sound, 7(2) pp 97-108, 2002

[37] Schatter G., Züger E., Nitschke C. A “Synaesthetic approach for a Synthesizer Interface Based on Genetic Algorithms and Fuzzy Sets.” in Proceedings of the 2003 International Computer Music Conference (ICMC2004), Barcelona, Spain, 2005

[38] Wanderley M. and Depalle P. “Gestural Control of Sound Synthesis” in Proceedings of The IEEE, Vol. 92, No. 4, April 2004.

[39] Goudeseune C. “Interpolated Mappings for Musical Instruments.” Organised Sound 7(2) pp 85-96, 2002.

[40] Bencina R. “The Metasurface – Applying Natural Neighbour Interpolation to Two-to-Many Mapping.” in Proceedings of the 2005 International Conference on New Interfaces for Musical Expression (NIME05), Vancouver, BC, Canada, 2005.

[41] Atmel AVR 8-Bit RISC Overview - high performance low power flash

microcontroller. Retrieved on 29th March 2006 from http://www.atmel.com/products/AVR/overview.asp

[42] Procyon Engineering. Retrieved on 29th March 2006 from http://www.procyonengineering.com/.

[43] EDTP Electronics. Retrieved on 29th March 2006 from http://www.edtp.com/

[44] RTL8019AS ISA Full-Duplex Ethernet Controller with Plug and Play Function. Retrieved on 29th March 2006 from http://www.realtek.com.tw/products/products1-2.aspx?modelid=1

[45] AX88796B -- Low-Pin-Count Non-PCI Single-Chip 8/16-bit 10/100M Fast Ethernet Controller. Retrieved on 29th March 2006 from http://www.asix.com.tw/products.php?op=pItemdetail&PItemID=80;65;86&PLine=65

[46] Atmel Corporation, AVR Studio 4. Retrieved on 29th March 2006 from http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725.

[47] WinAVR (AVR GCC). Retrieved on 29th March 2006 from http://winavr.sourceforge.net/.

[48] M. Wright. Using avr microprocessors under os/x. Retrieved on 29th March 2006 from http://ccrma.stanford.edu/\%7Ematt/avr-osx.htm.

[49] NTP: The Network Time Protocol. Retrieved on 29th March 2006 from http://www.ntp.org/

[50] Norman D. A. “Some Observations on Mental Models,” in D. Gentner and A. L. Stevens, Eds., Mental Models, Erlbaum, 1983.

[51] Young R. M. “Surrogates and Mappings: two Kinds of Conceptual Models for Interactive Devices,” in D. Gentner and A. L. Stevens, Eds., Mental Models, Erlbaum, 1983

[52] Slaney M., Covell M., and Lassiter B., “Automatic Audio Morphing” in Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Atlanta, GA, May 7-10, 1996.

[53] Altman, E. and Wyse, L. “Emergent Semantics from Media Blending”, in

(Srinivasan and Nepal Eds.) Managing Multimedia Semantics. The Idea Group Inc, 2004.

[54] Rovan J., Wanderley M. M., Dubnov S. and Depalle P. "Instrumental Gestural Mapping Strategies as Expressivity Determinants in Computer Music Performance." Technical Paper, Synthesis Team/Real-Time Systems Group, IRCAM, France, 1997.

[55] Maxim MAX127. Retrieved on 29th March 2006 from http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1890