A Tool for Configuring Mappings for Musical Systems using Wireless Sensor Networks

Vijay Rudraraju

Music Technology Area
Schulich School of Music
McGill University
Montreal, Canada

December 2011

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Arts.

2011/12/12

Abstract

Digital musical instruments, which are defined here as interactive musical systems containing a control mechanism and a sound generation mechanism, are powerful tools for analyzing performance practice and for transforming and reimagining the bounds of musical performance. However, the transitory nature of digital technology and the complexity of maintaining and configuring a digital musical instrument involving tens, if not hundreds, of interconnected, discrete components presents a unique problem.

Even the most mechanically complex acoustic musical instruments, like a piano, are robust enough to withstand the daily grind without expert intervention by someone with intimate knowledge of the material and mechanical construction of the instrument. Furthermore, they are standardized enough that repairs can be conducted by any number of trained professionals. By contrast, digital musical instruments are often configured differently for each performance (this configurability being one of the virtues of a digital musical instrument), incorporate any number of non-standard pieces of hardware and software, and often can only be reliably configured by their creator.

This problem is exacerbated as the number of sensors that make up the control mechanism in an instrument increases and the interaction of the control mechanism with the sound generation mechanism grows more complex. This relationship between the control mechanism and the sound generation mechanism is referred to here as the "mapping" of the instrument. The mapping for an instrument represents the aspect of an instrument that is usually most configurable because it is defined by software (as opposed to hardware) and also most crucial to the character of the instrument. In the case of a digital musical instrument, being able to easily configure the musical instrument becomes a point of artistic freedom in addition to a point of maintainability.

This thesis builds upon work encompassed in two projects at the Input Devices and Musical Interaction Lab, the Digital Orchestra Project and Libmapper, to tackle the problem of building an interface/system for configuring a complex musical system without expert programming skills. The intent is to present a targeted survey of user interface design and data visualization design research through the years to inform the design of a graphical user interface for performing this configuration task.

Résumé

Les instruments de musique numériques, que l'on définit ici comme des systèmes musicaux interactifs contenant un mécanisme de contrôle ainsi qu'un mécanisme de production sonore, constituent de puissants outils pour l'analyse des méthodes de performance ainsi que pour la transformation et la redéfinition de la la performance musicale. Cependant, la nature éphémère des technologie digitales ainsi que la complexité du maintien de la configuration d'un instrument de musique numérique, comprenant des dizaines voire centaines de composantes discrètes interconnectées, présente un réel problème.

Même les systèmes musicaux acoustiques le plus complexes, comme le piano, sont assez robustes pour résister à l'usure quotidienne sans l'intervention d'une main experte ayant une connaissance aiguisée des matériaux et de la mécanique de l'instrument. Par ailleurs, ces instruments sont assez standards pour que tout luthier qualifié soit en mesure d'intervenir sur l'instrument en cas de besoin. A l'inverse, les instruments de musique numériques présentent souvent des configurations propres a chaque performance (cette configurabilité étant une des vertus des instruments digitaux), incluent divers type de système hardware et software, et ne peuvent souvent être configurés de manière fiable que par leur créateur.

Le problème grandit lorsque le nombre de capteurs utilisés pour la partie contrôle d'un instrument augmente et que l'interaction entre les sous-blocs contrôle et production sonore se complexifient. Cette relation entre le mécanisme de contrôle et de production sonore est appelée "mapping" de l'instrument. Le "mapping" d'un instrument se rapporte aux aspects les plus configurables définis par le logiciel (par opposition au éléments hardware), et mais aussi faisant partie les caractéristiques les plus importantes de l'instrument. Dans le cas d'un instrument de musique numérique, la capacité a agir facilement sur la configuration devient alors un enjeux de liberté d'expression artistique, en plus d'une condition de pérennisation.

Ce travail de thèse se construit sur la base de deux projets menés au sein du laboratoire IDMIL (Input Devices and Musical Interaction Lab), le "Digital Orchestra" et "Libmapper", et traite du problème de construction d'un système/interface pour la configuration de système musicaux complexes sans connaissances expertes en programmation. L'objectif est de présenter un sondage ciblé sur la recherche en conception d'interface utilisateur et visualisation de données, dans le but d'apporter des éléments nouveaux a la conception d'interfaces destinées aux taches de configuration d'instruments digitaux.

Acknowledgments

Many thanks to my supervisor, Professor Marcelo M. Wanderley, for his abundance of available time for his students and helping me integrate my personal work with the work of others at the Input Devices and Music Interaction Laboratory (IDMIL).

I am grateful to have been in the presence of all the students and professors of Music Technology at McGill University. Never have I been part of a group of people who were so eager to offer their time to others who needed help, regardless of whether they had something to gain from it. Special thanks to Joseph Malloch and Stephen Sinclair, without whose help and hours of work on the Mapper Tools project I would never have been able to even contemplate this research project.

I would never have survived the Montreal winters if it was not for Vincent Freour, a boundless source of energy who pulled me out into the sunlight even when I was determined to remain in my cave.

Finally, infinite gratitude to my parents Pandu and Padma who have always and will continue to support me, no matter how seemingly useless and lost I may appear through my many journeys.

Contents

List of Figures

List of Tables

List of Acronyms

HCI	Human-Computer Interaction
UID	User Interface Design
DMI	Digital Musical Instrument
EMI	Electronic Musical Instrument
OSC	Open Sound Control
IDMIL	Input Devices and Music Interaction Laboratory

Chapter 1

Introduction

> "Most principles of design should be greeted with some skepticism, for word authority can dominate our vision, and we may come to see only through the lenses of word authority rather than with our own eyes.
>
> What is to be sought in designs for the display of information is the clear portrayal of complexity. Not the complication of the simple; rather the task of the designer is to give visual access to the subtle and the difficult - that is,
>
> the revelation of the complex." - Edward R. Tufte, 2001 [3]

1.1 Project Overview

The goal of this research project is to apply a thorough understanding of a specific problem and a selection of theoretical principles in the areas of user interface design, data visualization, and human cognition to design a user interface that allows a user to solve this problem as effectively as possible. The problem addressed here is creating a mapping between a large number of signal sources and destinations distributed over many devices on a network to be used in a musical scenario.

This research is an offshoot of a branch of research that began with the McGill Digital Orchestra Project [4]. The spirit and intent of the Digital Orchestra Project is understood as follows.

> "Although designers of Digital Musical Instruments (DMI) are interested in creating useful, flexible, and creatively-inspiring interfaces and sounds, this process

> often depends on the vision and insight of a single individual. The McGill Digital Orchestra project instead brings together research-creators and researchers in performance, composition and music technology to work collaboratively in creating tools for live performance with digital technology." [5]

The research project described in this thesis is a rethink of one particular aspect of the Digital Orchestra Project that is especially important.

> "In the process of creating instruments for this project, we have found ourselves faced with the unique challenge of mapping new instruments in collaboration with experienced performers, as well as with composers tasked with writing pieces for these instruments. Because this ambitious project has taken on these three main challenges of the digital performance medium simultaneously, we have found ourselves in need of tools to help optimize the process. Specifically, mapping the various streams of controller output to the input parameters of synthesis engines has presented us with situations where both ease of use and flexibility were both [sic] of the utmost importance. We needed to be able to modify connections between data streams during precious engineer-composer-performer meeting time, while minimizing wasted minutes "reprogramming" our signal processing routines." [5]

This portion of the Digital Orchestra Project is now referred to, at the Input Devices and Music Interaction Laboratory (IDMIL) at McGill University, as the *Mapping Tools* project and includes the *Digital Orchestra Toolbox* developed during the Digital Orchestra Project. The current manifestation of this research into simplifying DMI mapping is *Libmapper*, a C library that implements the *Mapper protocol* that is used to create mappings and the *Maxmapper* graphical user interface (GUI) that is used to interface with the functionality that Libmapper provides. This GUI is called Maxmapper because it is implemented using the Max/MSP development environment and to differentiate it from *Vizmapper*, the name of the alternative GUI developed through the research presented in this thesis.

The Maxmapper has been used by many musicians and composers and has proven its ability to simplify the DMI mapping task for non-programmers. However, the interface is typically used in situations where there are a small number of DMIs each with a small number of signals. Examining the interface in Figure 1.1, it is clear that in a scenario

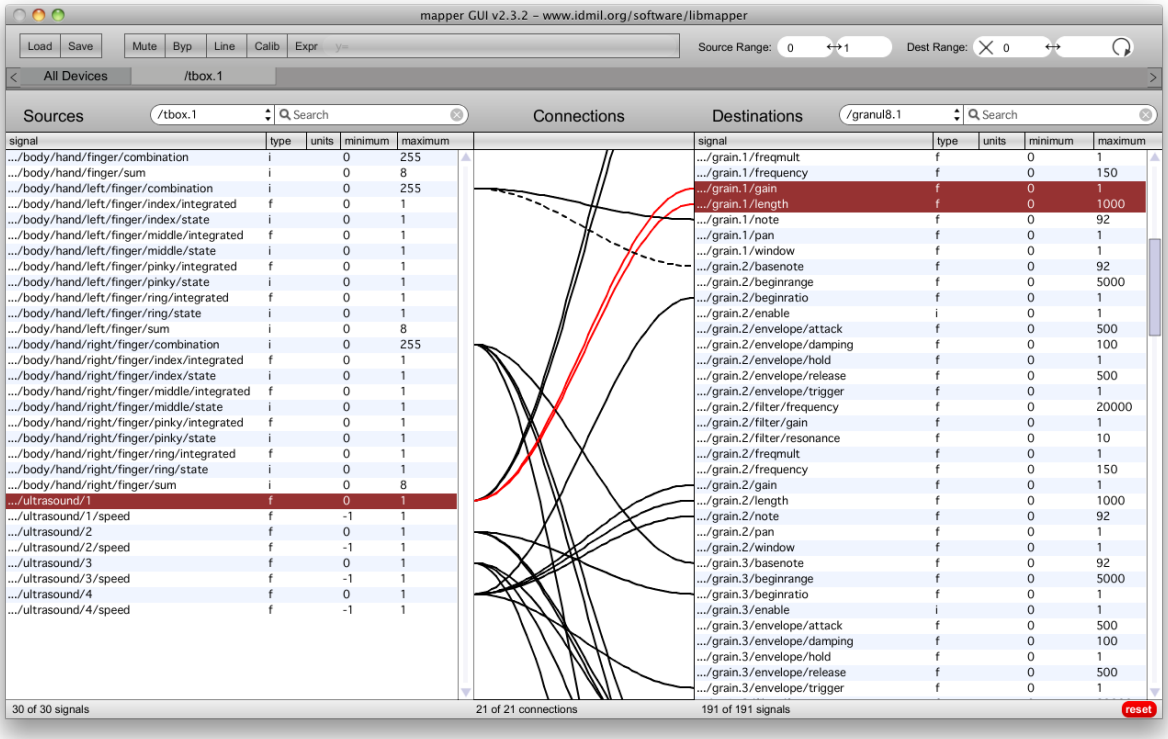


Fig. 1.1 The Maxmapper GUI

with an order of magnitude more signals, the way that the data is *visualized* will cease to provide the user with any understanding about the relational structure of a Mapper network or what configuration to use for a large network. Since the primary utility of a computer (in the sense of a being something that performs computations) is dealing with large bases of data and operating large numbers of things, this is a significant shortcoming of the user interface. After all, people performed computations before computers! We use computers because they can perform operations faster and in larger quantities than a human.

Computer systems engineers often talk about the ability of a system to *scale*, meaning the ability of a system to gracefully handle an increasing number of users, machines, data stores, etc. without significant structural modifications or loss of utility. In some sense, what is happening is that the user interface and data visualization of Maxmapper does not *scale* from DMIs to a larger sensor network. The hope is that the research in this thesis contributes some insights into alleviating this interface problem and produces a successful

alternative interface for configuring Mapper-based musical sensor networks.

1.2 Thesis Structure

This thesis is structured as follows:

Chapter 2 introduces the concept of mapping by providing a theoretical grounding of the concept as understood in mathematics and computer science and explains the more targeted concept of mappings in DMIs.

Chapter 3 examines the history of computer user interfaces as context for understanding a few principles discovered in the disciplines of user interface design, data visualization design, and cognitive science.

Chapter 4 uses these principles to design and implement a user interface for configuring mappings.

Chapter 5 concludes the thesis and presents hopes for future work on this topic.

Chapter 2

Mapping

Mapping is a concept that is useful in a number of disciplines. Within the discipline of designing digital musical instruments, mapping refers to the relationship between sensor outputs and sound synthesis algorithm inputs in a DMI. It has been shown that the choice of mapping is a primary factor in the design of DMIs [6]. Designers must have a clear sense of the space of possible configurations for a given set of control sensors and sound synthesizers, if they are to devise mapping strategies that effectively translate their design intentions into a functioning system.

It follows from this foundational premise that the methods a designer's tools use to frame the space of possible configurations greatly influences the final design of the DMI. For instance, when dealing with a musical system with large numbers of sensors and synthesis algorithms with many inputs, it is not a trivial task to decide what configuration out of the thousands or tens of thousands (or even more) of possible configurations to use. It is not practical to test every possible mapping. Instead, it is important that the tools a designer uses allow them to develop an integrative understanding of the relationships between the inputs and outputs in a network and quickly rule out large numbers of configurations. The first step towards this goal is acquiring a thorough understanding of the concept of mapping.

2.1 Mappings in Theory

The term *mapping* is used in mathematics, computer science, and related technical fields to encapsulate the concept of a *function* and other related concepts under one abstraction.

Like most technical terms used in multiple technical disciplines, the meaning of *mapping* varies according to the context of the field that it is used. However in the case of mapping, the variation is subtle enough that a cursory understanding of the way mapping is used in a couple of contexts outside of DMIs might illuminate why it is a necessary and valuable concept to encapsulate in the context of new musical interfaces and help understand what is required for an interface designed specifically for mapping.

2.1.1 Mappings in Mathematics

In the mathematics context, the use of the term *mapping* varies subtly according to the subdiscipline within mathematics in which it is used and often between specific mathematicians; however it is acceptable to use the word as a synonym for *function*. A function in mathematics describes an associative relationship between two sets of numbers. One set of numbers is referred to as a *domain* or *set of inputs* and the other set is referred to as a *codomain*, *range*, or *set of outputs*. Mathematicians often say that a function "maps" a domain to or onto a range, hence the reason for using the term *mapping* as a synonym for *function* [7].

To avoid confusion in later chapters, note that this the opposite of how outputs and inputs are referred to in a DMI mapping. The set of output signals is actually the domain of a DMI mapping and the set of input signals is the codomain of a DMI mapping.

Strictly speaking in mathematics, a function can only represent a *one-to-one mapping* or a *many-to-one mapping*. It is helpful to know that a property of both types of mapping is that every element in the input (domain) set is associated by the function with one and only one element in the output (range) set. A corollary statement about both one-to-one and many-to-one mappings is that every element in the output set might be associated to one element (one-to-one) or many elements (many-to-one) in the input set. This language should become more clear as the usage of the terms *one-to-one* and *many-to-one* are explored in the remainder of the chapter.

The following are examples of equations that represent functions:

$$y = f(x) = x + 1 \tag{2.1}$$

$$z = g(x) = x^2 \tag{2.2}$$

In addition to both being functions, equation 2.1 is a one-to-one mapping and equation 2.2 is a many-to-one mapping. Basic types of functions like equations 2.1 and 2.2 often use the set of all real numbers (roughly, defined as any number that can be expressed as a potentially infinite string of digits - in the case of irrational numbers like π - in decimal notation) as both the input set and the output set. Since the set of all real numbers is infinitely large, the association from one input element to one output element is described by a mathematical transformation that when applied to any element of the input set (the set of all real numbers) produces the element of the output set (the set of all real numbers) that the input element is mapped to. This is convenient because that means the function can be written very concisely in one line as opposed to two infinitely long columns with lines drawn between elements that are mapped to each other (not very convenient).

Equation 2.1 is the canonical notation for representing a mapping/function labeled f where the input element is labeled x and the output element is labeled y. Equation 2.2 represents a mapping/function labeled g where the input element is labeled x and the output element is labeled z.

To calculate the mapping that is implied by these two mathematical transformations, substitute any real number (since the set of all real numbers is the input set) for the symbol x. By this logic, equation 2.1 represents the mapping where every real number x is mapped to the real number y that is the value of $x + 1$. Equation 2.2 represents the mapping where every real number x is mapped to the real number z that is the value of x^2. Equation 2.2 is a many-to-one mapping because it maps both the positive and negative of every input number to the square of that number (the square of any real number is always positive), unlike equation 2.1 which maps every input number to an output number that no other input number is mapped to.

2.1.2 Cardinality in Mathematics

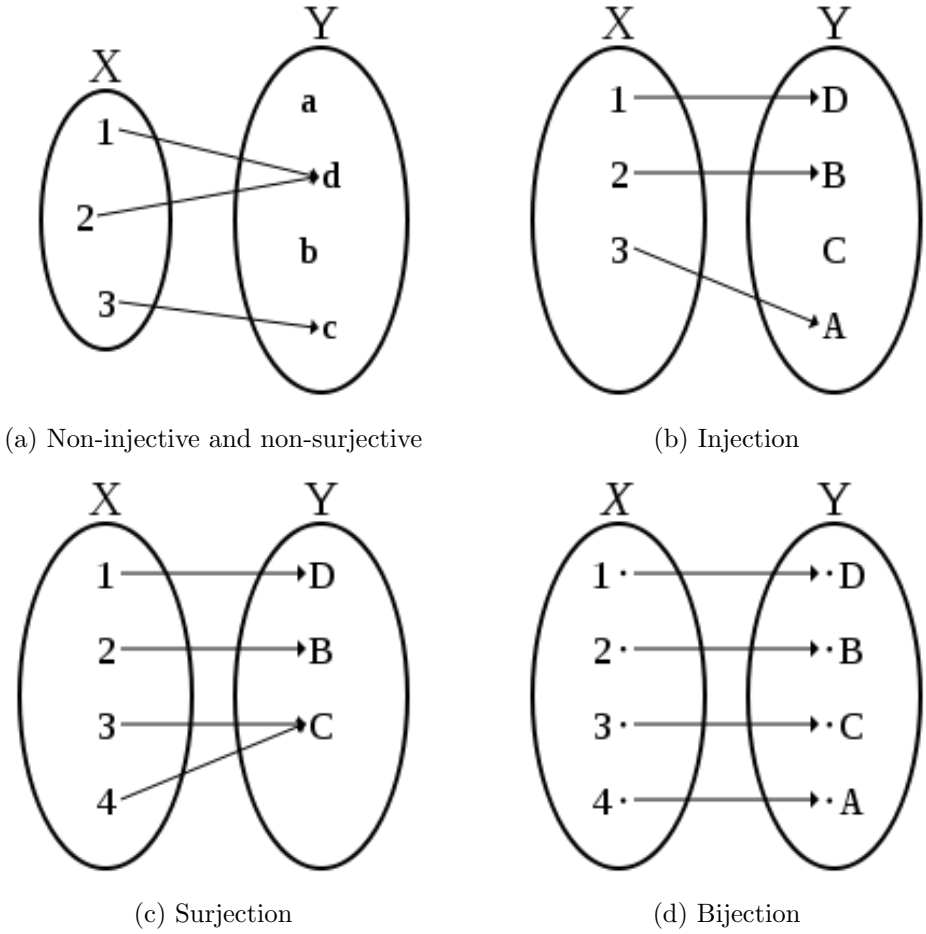
There is more to be said about the concept of a one-to-one mapping and two related concepts, one-to-many mappings and many-to-one mappings. The framework for discussing different types of mappings is provided in one form by the mathematical concept of *cardinality*.

First, three additional concepts should be enumerated and elaborated upon. These concepts are *injection*, *surjection*, and *bijection*.

An *injection* is a function that meets the condition that every element of the function's codomain (output) set is associated to no more than one element in the functions domain (input) set. Here, domain and codomain are used in the same way as in section 2.1.1.

A *surjection* is a function that meets the condition that every element in the codomain set is associated to at least one element in the domain set.

A *bijection* is a function that is both an *injection* and a *surjection*.



(a) Non-injective and non-surjective

(b) Injection

(c) Surjection

(d) Bijection

Fig. 2.1 Four types of functions

Figure 2.1 summarizes the possible types of functions using these distinctions.

Within mathematics, cardinality is a concept that allows one to reason about the relative sizes of different sets. It is particularly useful when comparing infinite sets. It allows one to reason correctly, for example, that the infinite size of the set of all real numbers is larger

than the infinite size of the set of all positive integers (natural numbers). Yes, infinity can be proven to be larger than infinity!

Two sets are defined to have the same *cardinality* if there exists a bijection between the two sets. It follows from the definition of injection that if there exists an injection from set A to set B, but no bijection, then the cardinality of B is greater than the cardinality of A [8].

As mentioned before, these mathematical concepts are useful for describing one-to-one mappings and many-to-one mappings. In this terminology, injections and bijections are one-to-one mappings, whereas surjections and functions that are neither injections or surjections are many-to-one mappings.

2.1.3 Mappings in Computer Science

In the computer science context, a mapping is an abstract data type or concrete data structure, commonly referred to as an *associative array* or *dictionary* [9].

Abstractly and simply, the one and only way to store data of any kind (the value "0", the value "1", some text, an image, a sound, a recorded signal) in a computer of any kind, such that one can reliably retrieve the data later, is to put the data in some "bucket" that has some "label". The words "bucket" and "label", as used here, are not well-defined in computer science. But they help to unobtrusively make the point that if the data has no label that is associated with it or equivalently if the label has no unique memory location associated with it, then practically, it cannot be retrieved. In the scenario of a computer processor accessing internal memory, *bucket* and *label* could be replaced by *memory location* and *memory address* respectively.

As an example from the music information retrieval context, if a librarian wants to be able to store a large amount of sheet music such that it can be found later, the librarian needs some system to catalog or index the material such that a piece of sheet music can be referred to by some label. In the case of a library, this label is often a call number, the title of the piece, the name of the composer, etc.

Just like a reference library can use many different labeling systems to catalog the sheet music stored within its walls so that it can be retrieved with minimum headaches, a computer system can internally utilize many different abstract data types or concrete data structures. In both cases, the best choice is dependent on the material or data being

cataloged or indexed. A mapping as a data structure in computer science parlance then, can be understood as a particular type of system for labeling data.

In particular, a mapping is a labeling system that is composed of a set of labels (keys) and a set of pieces of data (values), where every label in the first set is associated with one or many of the pieces of data in the second set. Then this structure encompassing the two sets and the list of associations between the two sets is given a higher level label that refers to the mapping as a whole. This allows one to construct a mapping of mappings, which lends itself nicely to hierarchical labeling systems.

This reframing of the term in the computer science context, as opposed to the mathematical context, focuses on one implication of the concept of mapping. One can specify multiple, unique mappings that each operate on the same collection of labels and the same collection of pieces of data. Similarly, one can specify multiple, unique functions that each operate on the same input set and the same output set.

So in a computer system, one can create multiple mappings that each associate the same set of labels with the same set of data differently. This characteristic of mappings is one of the primary reasons why mappings are a useful frame for understanding the potential of digital musical instruments.

2.1.4 Cardinality in Computer Science

Within computer science, cardinality is a concept used in relational database systems [10].

A relational database can be seen as a very robust and flexible *data labeling system*, in reference to the discussion in Section 2.1.3. A relational database is defined as a set of relations, where a relation is defined as a set of lists of attributes. Each list in the set has the same attributes as every other list in the set. An attribute defines a property that can take any value constrained by the set of values that are defined for each particular attribute. A relation is usually described by a table as in Tables 2.1, 2.2, and 2.3. Together these three relations define one database.

In a relational database, the cardinality of an attribute is the number of instances of that attribute that can be associated with a given set of instances for other attributes. In a database as in DMI mapping, it is largely sufficient to distinguish between a cardinality of *one* and a cardinality of *many*.

The simple database composed of the relations in Tables 2.1, 2.2, and 2.3 provides an

Student ID : Integer	Name : String	Thesis Topic : String
00001	Alice	4d Maps
00002	Bob	Anti-Gravity Toaster
00003	Carl	Invisible Clothing
00004	Doug	Personal Babel Fish

Table 2.1 Example of a database defined by 3 relations - Student Relation

Faculty ID : Integer	Name : String
0001	Ohyes Hedid
0002	Ohno Hedidnt

Table 2.2 Faculty Relation

Pairing ID : Integer	Faculty ID : Integer	Student ID : Integer
00000001	0001	00004
00000002	0002	00002
00000003	0001	00003
00000004	0001	00001

Table 2.3 Pairing Relation

example to reason about the cardinality of database relations. If every student (represented by their ID) has one advisor (also represented by their ID) but every advisor has many students, then the relation is one-to-many from Advisor ID to Student ID. Since the horizontal ordering of attributes has no meaning in a relation, we can also reason that the relation from Student ID to Advisor ID is the reverse, many-to-one. Finally, every student has one thesis topic and each thesis topic is the topic of one student so the relation is one-to-one from Student ID to Thesis Topic.

These mathematical and computer science concepts provide a solid foundation for understanding the various nuances of mapping abstractly and concretely. This background will be necessary to create a well-informed user interface for creating and modifying DMI mappings.

2.2 Mappings in Digital Musical Instruments

In an acoustic musical instrument, the equivalents of the "control mechanism" and "sound generation mechanism", as they are called in a DMI, are *intrinsically coupled* or bound to each other because the physical material (e.g. wood, metal, horse hair) of the instrument that forms the control mechanism also forms the sound generation mechanism. Making a decision that alters the control mechanism will significantly or subtly alter the sound generation mechanism. Replacing the horse hair of a violin bow with artificial fiber will result in a perceived difference in tone to the trained ear.

In a DMI, the control mechanism is composed of sensors typically embedded in structural materials that form the shape of the instrument and connected to a microcontroller that provides the interface for accessing the sensor signals. The physical gestures of the performer are not coupled to the sound generation through purely mechanical means, but through electrical means. Each sensor, whether it senses acceleration, orientation, touch, etc. produces an electrical signal that correlates to some physical measurement. This analog electrical signal is converted to a digital signal through some kind of analog to digital signal conversion (perhaps within the control mechanism on an integrated microcontroller) and then this digital signal is processed by a computer and sent to the sound generation mechanism, which is a piece of software that produces audio samples.

The following illustrative example makes clear that the control mechanism and sound generation mechanism are not *intrinsically coupled* in a DMI as they are in an acoustic

musical instrument.

If one takes a control mechanism with embedded touch sensors, records an output signal produced by a performer with the control mechanism on a hypothetical signal recorder, and replaces the connection between the control mechanism and the sound generation mechanism with a connection between the signal recorder and the sound generation mechanism, it is clear that the active control mechanism of the DMI has changed. The control mechanism is now the signal recorder. The original control mechanism with the touch sensors is no longer part of the DMI system because it is disconnected from the DMI. But this change results in no change within the sound generation mechanism. The sound is generated by software that produces audio samples and thus is not affected by a change to the physical hardware of the control mechanism as long as the control mechanism produces an output signal that is identical. If the performer presses play on the recorder it will be impossible to tell the difference between the initial DMI and the modified DMI purely from the sound generated by the DMI.

In physical terms, when two formerly intrinsically coupled aspects of system are free to become uncoupled, a new degree of freedom is introduced. One imprecise way to understand why this happens is that the "signal" (used loosely) controlling the sound generation is no longer all the vibrations in the physical body of the instrument (in an acoustic musical instrument), but a discrete electrical signal that travels through a very specific electrical channel (in a DMI).

In the context of DMI research this new degree of freedom is the freedom to choose a mapping [6]. Mapping as used in this context is very similar to how the term is used in both the mathematics and computer science context. Each context provides an understanding that is well suited to manipulate one of two possible conceptual layers in a mapping system. These two contexts have also been referred to as a *systems* point of view and a *functional* point of view respectively [11].

1. The set of sensors embedded in the control mechanism outputs a set of signals, which we will refer to as *output signals*. The sound generation mechanism outputs sound that is controlled through a set of *input signals* defined by the software that is responsible for the sound generation. Therefore, for any given control mechanism and sound generation mechanism there is a choice of how to associate the two sets of signals (output signals from the control mechanism and input signals from the sound

generation mechanism) with each other, much like an associative array in the computer science context. Importantly, this is a choice that is not independently available when constructing, for example, a violin because the two sets of signals (if this distinction between control mechanism and sound generation mechanism is imagined in a violin) in a violin are physically coupled and inseparable. It is not a choice that is explicitly made when constructing an acoustic instrument.

2. Furthermore, there is a second, lower level form of mapping when dealing with a specific pair of associated signals. Because it is unlikely that any given signal output by a control device is calibrated to a particular input signal in a sound generation device that one might associate with the given output signal, we require a second set of relations beyond a binary associated/not associated. Any given output signal might be capable of outputting values that are outside the set of values that a particular input handles or simply be outputting a different data type altogether. In electronics, this is called *signal conditioning* and can be handled completely in the analog components that produce the signals. If this signal conditioning is not handled in the analog hardware and is modified dynamically in the software of the DMI, after analog to digital conversion of the signal, then this signal conditioning is handled by the mapping system. In this case, a mapping in the functional, mathematical sense of the word is necessary to transform the output signal so that the values reliably fall within the set of acceptable input values. Performing signal conditioning as part of the DMI mapping, as opposed to as part of the control mechanism, has the benefit of not requiring any change in the analog electronics of the DMI, if the associative mapping layer changes.

A prototypical mapping system can specify not only the associations between two sets of signals (pairs of connected signals), but associate a function with each pair of connected signals that specifies how to transform/condition the signal before copying the value to an input. This function might, in fact, be simply $y = x$ in the case where no calibration is needed, however the point is that it is not practical to assume that this will always be an acceptable mapping function between an output and input. Therefore, signal conditioning might be useful to include as a second layer in a mapping system.

This physical decoupling between the control mechanism, the mapping system, and the sound generation mechanism creates a modularity between the three components that can

be exploited to simplify the DMI development and maintenance process.

This model of a DMI, as being composed of three components, is simply one way to decompose and understand a DMI. There may be concern that the definition of the mapping system as an association of sensor signals and sound generation signals with a functional transformation defining a conditioning between two associated signals is overly simplistic.

After all, sensor signals can undergo an arbitrarily complex interpretation and translation process that involves multiple layers of transformations, external memory systems, machine learning algorithms, etc. in some DMIs. However, because there is generally a correlation between the complexity of a system and the interface required to configure the system, the model that is chosen must necessarily be a simplifying model. In any case, other types of processing can be encapsulated in additional devices that provide both output and input signals and act as additional layers of arbitrary processing that do not fit into this particular model of mapping.

The goal here is not to create another system that approaches the high complexity of a complete programming language. The only way to interface with a more complete (and complex) model of the mapping system in any hypothetical DMI would be to increase the complexity of the interface, however that would very quickly increase the technical knowledge required to operate the interface for the mapping system. The research done as part of the McGill Digital Orchestra Project [4], showed that this particular model of a DMI strikes a good balance between the creation of a large diversity of DMIs and maintaining a simple process for configuring the mapping system of a collection of DMIs, though it is doubtless that this is not the final solution to the larger problem.

2.3 A Dedicated Mapping System: Libmapper

The creation, evolution, and stabilization of the design of a DMI is inherently a tightly iterative process. A violin luthier must be comfortable enough playing a violin to develop an aesthetic sense of what effect the countless decisions that are made in the process of building a violin have on the experiential quality of the instrument that they are creating. Whether a particular decision is the right one can only truly be evaluated after the decision is made and someone plays the violin.

Similarly, a team of people building a DMI will often make a construction decision, evaluate the effect of that decision, make some changes, and repeat. This process is often

referred to as *iterative development* and occurs in any serious design process whether or not it is ever made explicit [12].

Because such iterative development is so beneficial to the development of DMIs, people often use various platforms and frameworks as common foundational building blocks for such systems. These environments often take care of implementing higher level functionality for controlling and generating audio in realtime or interfacing with hardware. Thus, they allow one to quickly test ideas for a DMI and focus on experimenting with the aspects of the control mechanism and sound generation that are unique to the particular project before investing too much time and money into custom parts without assurance that the ideas are viable in a basic sense. For the control hardware, there are platforms like Arduino. Coding environments like Max/MSP and code libraries like STK simplify the development of synthesizer modules for sound generation. In an effort to provide a similar foundation for creating mapping systems, the Digital Orchestra Toolbox was created as part of the McGill Digital Orchestra project and includes components representing the common subroutines used in a mapping system. More recently, this functionality has been reimplemented as a C library called Libmapper at IDMIL, a lab at McGill University. Libmapper has several characteristics which make it a desirable system for configuring mappings.

One is that a mapping between control and sound generation can be modified without recompiling any code, restarting any system, or reloading any script. Typically, if a mapping is specified in software through a Max/MSP program, C program, etc. then the part of the code that specifies the mapping must be modified and recompiled or reinterpreted before the new mapping becomes active. Any DMI that embeds Libmapper in its control and sound generation software can have its mapping modified simply by being sent specific messages as specified by the mapping protocol.

Another useful characteristic is that, in the case of a local network with many different control mechanisms and sound generation mechanisms available, no central device is needed to facilitate communication between devices. If one component on the network experiences some difficulties, the other devices will still be able to keep the mappings between the remaining devices operating.

Lastly, and of particular relevance to the primary topic of this paper, is that the current state of mappings between various devices on the network can be viewed and modified by any graphical user interface that embeds Libmapper. In Libmapper parlance, these applications can register as *monitors*. Because of this capability, multiple members of a

team can be viewing and modifying the same mappings for multiple DMIs using different computers and using different graphical user interfaces.

A network with Libmapper-enabled devices and monitors is a *Mapper network*.

2.3.1 Open Sound Control Protocol

Seen from a bird's eye conceptual level, Libmapper is an implementation of a communication protocol that was created for specifying mappings over a distributed network [13].

A communications protocol is a rigorous and formal description of a message format and rules for exchanging messages adhering to the protocol. The term "protocol" usually refers to the formal description of the protocol and is distinct from the implementation of the protocol. A protocol without an implementation is analogically like a constitution without a government. If two devices adhere to the same protocol then they are able to reliably communicate with each other because they can, in a sense, speak the same language. The Mapper protocol is itself defined in terms of the Open Sound Control (OSC) protocol. This is a common practice for constructing increasingly specific protocols. The Internet is essentially a particular stack of protocols that are layered on top of each other and specify how devices and routers are to be addressed, open connections with each other, and interpret messages.

To say that one protocol is layered on top of another is to say that the higher level protocol is defined in terms of the structure and conventions of the lower level protocol. In the case of Libmapper, the protocol layer that the OSC protocol is implemented on top of is called the User Datagram Protocol (UDP). The other common protocol that can be used at this conceptual layer of the Internet protocol stack is the Transmission Control Protocol (TCP). The well-known Hypertext Transfer Protocol (HTTP) is layered on top of TCP. A web browser implements HTTP so that it can fetch and display web pages from remote systems that speak the same protocol. As a protocol, OSC in not defined in terms of a particular Transport protocol (the name given to the layer in the protocol that UDP and TCP occupy - HTTP occupies the layer on top of the Transport layer, known as the Application layer). This is different from HTTP, which is specifically a TCP-based protocol.

Libmapper then allows one to configure a mapping relationship between any collection of devices (control devices, sound generation devices, and anything else) by embedding

Libmapper into the code for each device and creating an instance of Libmapper upon startup of each device, thus allowing any system on a local network to easily implement the Mapper protocol without in-depth knowledge of the protocol.

The specification of the OSC protocol can be found on the web [14]. In the opinion of the creators of the protocol [15], OSC is a "weak" form of protocol because it does not define certain necessary aspects of any form of data transmission. It does not define the patterns of command and response, error handling or receipt acknowledgement - only the format of the messages that are sent. These aspects are left as decisions to be made by particular implementations of the protocol or a protocol stacked on top of OSC.

The complete definition of the OSC protocol is not important for the scope of this thesis, however the following aspects of the protocol should give a sufficient sense of what is defined:

a unit of transmission is a *OSC Packet*, which consists of a contiguous block of binary data that is the contents of the packet and a number reflecting the number of bytes of data in the contents

the size of an *OSC Packet* is always a multiple of 4 bytes

the contents of an *OSC Packet* is an *OSC Message*

an *OSC Message* consists of an *OSC Address Pattern* followed by an *OSC Type Tag String* followed by zero or more *OSC Arguments*

an *OSC Address Pattern* is a URL-like string strictly beginning with the '/' character

an *OSC Type Tag String* is a string strictly beginning with the ',' character followed a number of characters corresponding to the number of *OSC Arguments* each of which specifies the OSC defined data type of each argument

2.3.2 Mapper Protocol

The semantics for the common set of OSC messages that the *Mapper protocol* (the name that will be used to describe the specification implemented by Libmapper in this paper) is intended to be as useful as possible for configuring mappings between output and input signals without assuming too much about what form the control or sound generation will

take. In the Mapper protocol, there is a concept of *source device* and *destination device*. In a DMI, the control mechanism will often be announced as a source device and the sound generation mechanism will be announced as a destination device.

The set of messages proposed by the Mapper protocol allows each device participating in the composite distributed system to *announce* its presence, *discover*, *describe* itself, *link* to another device, *connect* one of its signals (also referred to as datastreams) to a signal on a device it is linked to, and *condition or transform* a signal with a basic mathematical function [13]. These last two pieces of functionality are the two possible levels of mapping that can be present in the prototypical mapping system referenced in Section 2.2.

The messages that are used to perform specific operations on the network have their own Mapper syntax distinct from OSC syntax. This syntax is used to improve readability and is transparently translated into OSC syntax by individual Libmapper instances before transmission over the network.

Announcement

Each device in the distributed system announces itself with a port and unique name that it would like to be addressed by. In the case that other devices on the local network have already reserved a particular port or unique name, Libmapper uses a collision detection algorithm to systematically try other ports and/or modify the requested name with an appended unique ordinal number. This allows a set of identical control or sound generation devices to each be programmed to request the same collection of OSC names for their signals without worrying about the identical devices being addressed by the same names.

The device is also responsible for declaring the signals that can send as output or receive as input. These signals are addressed by hierarchical OSC Address Patterns with the device name that the signal is associated with as the root of the address pattern. Therefore a device named `tstick` with an assigned ordinal of `2` would be addressed as `/tstick.2` and an accelerometer signal on the device could be addressed as `/tstick.2/accel/1/x`. A second signal might be addressed as `/tstick.2/accel/1/y` and a third signal, `/tstick.2/accel/2/x`. The fact that the name of every signal is formatted as a hierarchical namespace will be useful when deciding on the best visualization for this data.

Discovery

After a device appears on the network, announces itself, and receives a unique name and port number on the local network through the port and name allocation algorithm, it sends the following message to request that all Libmapper-compatible devices describe themselves:

```
/who
```

Other devices are obliged by the protocol to respond with (for example):

```
/registered /tstick.1 @inputs 1 @outputs 52 @class /tstick
  @IP 192.168.0.3 @port 8001
/registered /granul8.1 @inputs 80 @outputs 0 @class /granul8
  @IP 192.168.0.4 @port 8000
```

In this way any device or monitor (in the case of a graphical user interface) can request the names, IP addresses, and UDP ports of any devices on the network.

Device Linking

To create a direct network link between two devices, the Mapper protocol specifies the following message:

```
/link /tstick.1 /granul8.1
```

Libmapper creates a router in software, which can be thought of as a special virtual device distinct from source devices and destination devices, for each pair of linked devices. In Mapper protocol terminology, a link is created between a source device and a destination device. Every router is tasked with the signal conditioning and message transformation for any pair of connected signals involving the two devices in the source/destination link pair that the specific router is created by Libmapper to manage. This Libmapper design decision allows each device acting as a source in a link to handle the traffic and processing of the links that it is involved in, thus distributing the work load over all source devices as opposed to assuming a single central router handling all traffic between all linked devices in the system.

When the Libmapper instance in the source device successfully creates a router for the link and completes initialization of the connection after receiving a `/link` message, it responds with:

```
/linked /tstick.1 /granul8.1
```

Two devices are unlinked by sending the following message:

```
/unlink /tstick.1 /granul8.1
```

The source device then destroys the router handling this link after receiving a `/unlink` message, and responds with:

```
/unlinked /tstick.1 /granul8.1
```

Signal Connection

Once a router has been created between a source and destination by requesting a link, specific output signals on the source device can be connected to specific input signals on the destination device. For example, once a T-Stick [16] (`/tstick.1`) has been linked to the Granul8 granular synthesizer (`/granul8.1`), the individual signals output by the touch sensors, accelerometers, etc. in the T-Stick can be routed to specific inputs in the granular synthesizer. Without specifying connections between output signals and input signals, gestures performed with the control source device will have no affect on the audio samples produced by the sound generation destination device and the DMI will produce no sound.

In the Mapper protocol, this is accomplished by sending a message similar to these:

```
/connect /tstick.1/raw/pressure /granul8.1/gain
/connect /tstick.1/raw/pressure /granul8.1/gain
  @scaling expression @expression x*10 @clipping minimum 0
```

The router managing the link between `/tstick.1` and `/granul8.1` records all the relevant information communicated by the `/connect` message, sets up the address translation and transformation required to connect the output signal to its associated input signal, and responds with:

```
/connected /tstick.1/raw/pressure /granul8.1/gain
/properties /tstick.1/raw/pressure /granul8.1/gain
  @scaling expression
/properties /tstick.1/raw/pressure /granul8.1/gain
  @expression x*10
/properties /tstick.1/raw/pressure /granul8.1/gain
  @clipping minimum 0
```

A signal pair is disconnected by sending:

```
/disconnect /tstick.1/raw/pressure /granul8.1/gain
```

The router responds with:

```
/disconnected /tstick.1/raw/pressure /granul8.1/gain
```

The properties of a connection can be queried with:

```
/connection/properties/get /tstick.1/raw/pressure /granul8.1/gain
```

The syntax of the message for modifying properties of a connection is similar to creating a connection with `/connect` replaced by `/connection/modify`.

It may be noticed from this syntax that the Mapper protocol supports only *one-to-one* and *one-to-many* mappings between sets of signals. An output signal can be connected to one or many different input signals. However, the syntax does not provide a mechanism for specifying how to handle one input signal receiving a signal from multiple output signals. The added complexity of handling a *many-to-one* or *many-to-many* mapping maybe introduced into the Mapper protocol at a future time.

These special Mapper messages as defined by their message headers and syntax comprise what is referred to in this paper as the *Mapper protocol*. By sending messages from this set of messages recognized by Libmapper it is possible to configure mappings between an arbitrarily large collection of devices on a local network with relatively little work and without understanding OSC.

2.4 Sensor Networks

A *sensor network* is a term used in electrical engineering literature to refer to a system that employs a (typically) large number of distributed sensors that are connected to each other through autonomous, intermediary nodes capable of processing and communicating the data obtained from the particular sensors it is attached to and relaying that information to other nodes on the network. If the nodes communicate with each other through some form of wireless radio then the network can be referred to as a *wireless sensor network*. The key functional difference being that in the wireless scenario, the nodes are not tethered to a particular physical location and are free to move provided they remain within radio range. Libmapper can, in principle, with few to no modifications operate in either scenario and the phrase *sensor network* will be used to encapsulate both.

The concept of a sensor network is useful because it strikes a good balance of specificity and abstraction when describing a particular network topology and implementation. Sensor networks, particularly wireless sensor networks, are an active field of research in a diverse array of research fields including defense, environmental monitoring, energy/heat systems, and structural engineering. In these and related situations, the function of the sensor network is to sense, measure, and gather information from the environment and/or actors in the environment and use the capability of distributed processing nodes to make decisions about where the information should be routed [17]. The majority of research applications do not focus on the use of these datastreams to output sound, visuals, or control actuators in realtime, however the definition of a sensor network does not preclude the nodes in the network from being connected to actuators as well as sensors.

2.5 Digital Musical Instruments as Localized Sensor Networks

A DMI can be understood as a particular form of sensor network that is particularly localized and stable. Particularly, if all of the sensors, processing nodes (perhaps each running an instance of Libmapper), and sound generation mechanism are all housed in a single physical object. Typically, no academic would refer to such a DMI as a sensor network, however it would technically fall under the working definition used in this paper. A sensor network may include functionality that compensates for unreliable sensors and devices that enter and leave the network relatively often, especially for a large, distributed network,

which is not necessary in a DMI with comparatively few and robust sensors. The larger the network, the more effort must be made to handle sensor and device failure. Hardware failure and reliability is a topic outside of the scope of this thesis, but is an important practical distinction between systems that are typically termed *DMIs* and systems termed *sensor networks*.

Framing a DMI as a sensor network affords one the ability to imagine what form a more *sensor network-like* DMI might take. Additionally, because the task of mapping is fundamentally seen as a task of network topology configuration in this thesis, it is a better term to use when focusing on the mapping layer of a musical system. A single DMI might consist of many spatially-distributed control devices that each output signals connected to the inputs of a sound generation device. Or an orchestra of DMIs might be understood to be one musical sensor network that together produce one collection of sound. The argument is that understanding what is being built in terms of a sensor network, as opposed to individual DMIs that may or may not be played together, better reflects the peculiarities of what a DMI fundamentally is and why the task of building a DMI is so different from building an acoustic musical instrument.

Additionally, perhaps such a reframing might better work *with* the peculiarities of a DMI and lessen the extent to which design teams work against the inate characteristics of computers attempting to make DMIs in the image of acoustic musical instruments. After all, as previously discussed, this fundamental difference between a DMI and acoustic musical instrument is what introduces the task of mapping as unique to DMIs in the first place. For convenience, the term *musical sensor network* (MSN) will be used in the remainder of this paper to reference this particular framing of the DMI concept in contexts that might be helped by this alternative concept. However, the two terms can usually be used interchangeably without being inaccurate.

2.6 Usability and Graphical User Interfaces

Imagine a MSN comprised of more than ten Libmapper-enabled devices, some of them are control devices and some are sound generation devices. Each device might output, or receive as inputs, around ten signals. So to create a mapping layer over this distributed network of devices, one might send hundreds of Mapper protocol messages to link, unlink, connect, disconnect, etc. various pairs of devices and signals as one attempts to determine through

trial and error the specific set of connections and functional transformations that best fit the particular sensors, devices, context, and design intentions of the people constructing the system.

It may be that it is unclear what control device ought to be linked to what sound generation device, much less what signals within the control device to connect to what particular signals within the sound generation device. When experimenting with a mapping layer for a specific MSN it is necessary to view the current network mapping, modify the current mapping, test the new mapping, evaluate its effectiveness and then repeat the process until the mapping is satisfactory.

Any tools and interfaces that speed up this iterative process and reduce the amount of Libmapper-specific/Mapper protocol knowledge required to efficiently explore the possibilities of a particular MSN improves the usability of Libmapper and allows for a wider diversity of skill sets and roles in a broader performance project involving instrumentalists, composers, directors, and engineers to participate in the mapping task that is so crucial to the character of a musical system.

Advances in user interface design are, historically, a need driven by the specifics of the nature of the system that the user requires an interface to manipulate. Similarly, data visualization design is driven by the need of a user to develop understanding of a system from a complex set of data about the system so that they can correctly manipulate the system to match their intentions.

2.7 Summary

Mapping is a theoretical concept that is applicable to many different scenarios. However, the basic elements of the concept tend to remain intact throughout its usage in different disciplines. Understanding the differences and similarities between its usage in mathematics, computer science, and DMIs allows one to develop an understanding of the concept, in a way that minimizes the dependency on domain specific language.

Since the goal is to design a tool that allows a person without excessive technical knowledge to configure mappings in musical systems, this understanding is important for determining how the concept of mapping can be simplified by ignoring non-essential aspects in the context of DMIs. The result of this research process is the, arguably, simpler concept of MSNs to reframe DMIs in terms that emphasize the connections between elements of

the network that make up a musical system.

In this way, this chapter has been a foundation for understanding the interface and visualization design decisions that are made in Chapter 4 as coupled to the requirements and need for a mapping system. The next chapter presents a foundation for understanding the decisions in Chapter 4 as coupled to what is known about what type of visualizations make a system easy to understand and what type of interfaces are effective for manipulating systems.

Chapter 3

User Interface and Data Visualization Design

The bounds of the discipline of *human-computer interaction* vary depending on who is asked. At present, people who practice the discipline seem to agree that human-computer interaction (HCI) involves "the design, implementation and evaluation of interactive systems in the context of the user's task and work" [18].

Within the broad domain of interactive systems, different subdisciplines within human-computer interaction (HCI) focus on different aspects of the systems. Designing a user interface for manipulating a Mapper network is an interesting task because the design problem is a composite of two interrelated subproblems. The first is visualizing the current state of the Mapper network in such a way as to allow a user to quickly reason about other possible states of the network. Solving this first problem effectively helps with the second problem, which is creating an interface that allows a user to determine what actions within the interface should be taken to move the network to a different, more desirable state.

Conveniently there are two subdisciplines within HCI that task themselves with analyzing these two problems: *data visualization design* and *user interface design*. Literature in these fields have much to say on the topic of how to proceed with the design of an interface and this chapter will focus on the findings that are most relevant to Vizmapper. To better understand the broader context of HCI, this chapter begins with a history of user interfaces in relation to the parallel history of general purpose computers.

3.1 Short History of User Interfaces

The term *user interface* is a byproduct of historical developments in computing and makes best sense in the context of modern computing culture. The use of *user interface* as a distinct concept was not needed when the *user* was certain to be a professional engineer or programmer involved in the creation of the program and/or the machine that was to run the program; at the very least, they were comfortable interacting with the computer using the same means as the original programmer and did not require an explicitly designed interface to augment the usability of the system [19]. Since *user* is such a vague term, *computer interface* is often used to refer to the part of the system that is meant by the term *user interface*. This part of the system is the computer's interface to the world - whether the *user* is assumed to be a programmer, a musician, another machine, the larger environment, etc.

The concerns of user interface designers have changed considerably since people first started interacting with programmable digital computers for processing, storing, and retrieving information. Understanding the context and nature of the interdependent effect that fundamental developments in computing and their accompanying user interfaces have on each other helps one appreciate the impact that a seemingly unimportant thing like user interface can have on the development of the basic functionality of the computer - for example, the function of using a computer for creating mappings between a distributed network of devices.

3.1.1 Punch cards

The oldest computer interfaces largely depend on punch cards and paper tape as control mechanisms and paper printers for output and program feedback. On machines used during the 1950s and 1960s, in order to input a program and a dataset into a computer, first one prepares a deck of punched cards with a typewriter-like machine, then one feeds the deck to the computer [20]. Some machines can also mount reels of magnetic tape that store oft-used or previously generated datasets or precompiled utility software. On these early machines, a particular program being run from a deck of punch cards is assumed to have control of all the available resources of the computer. There would be no other resident software running on the machine handling auxiliary tasks like on modern computers. Any hardware devices required by a particular program like punch card readers and line printers

have to be handled completely by the user program contained in the deck of cards fed to the machine.

Operating systems development grew out of the growing complexity of computer systems involving many cooperating devices, which themselves mirrored growth of the complexity of the programs that were written for them. It makes little sense for a user of a machine to write basic operating software for interfacing with the same peripheral devices as every other user of the computer every time they run a program on such a machine. Since any user of a computer during the 1960s needed to enter a queue to reserve time to run their program on a machine they shared with their entire organization, any time wasted debugging low-level interactions with input and output devices is a major setback [21]. Consequently, machines were developed that included libraries of code to take care of low-level control and provide easier access to input and output devices as always resident services. This collection of software that is permanently resident on a machine became known as a *batch monitor* [22]. A user's program could link to these preloaded libraries without including the operating logic explicitly within their own program code. This shift is often cited as the genesis of the modern operating system.

In addition to providing access to input/output devices, the monitor provides services to perform error checking on user submitted programs and for generating useful feedback to the user concerning the progress of execution of a user program. The idea of generating "useful" feedback, and what this might mean, can be understood as the first instance of an explicitly designed user interface and the birth of user interface design [23].

3.1.2 Command-line

Command-line interfaces are the next step in the evolution of computer interfaces and the link between batch systems of the 1950s and what would be recognized as a *graphical user interface* (GUI) today. As processors developed further and resulted in the reduction of the amount of time needed to execute a basic operation on data, it became possible to interact with the computer more granularly with a series of requests, execute programs pseudo-instantaneously, and receive responses expressed as specially formatted strings of text using a specialized vocabulary. Requests could be completed in seconds, no longer hours and days, and it made sense for the user to simply wait for the request to be completed before entering in the next request. This is as opposed to sequencing a stack of commands

in the form of a batch request and waiting for all the commands to be completed or for one of the commands to fail. Instead, the user can change their mind about the structure of commands in the composite program in response to feedback from earlier commands. This introduces the possibility for software to explore a set of possibilities with the guidance of the user and allows for a type of interactivity not possible with the batch systems of the 1950s.

Although the earliest command-line systems borrow typewriter-like teletypes (as used for telegraph transmission) as the input and output mechanisms, by the 1970s, video display terminals are used with computers for providing text feedback on a virtual canvas of pixels that can be rapidly and reversibly modified (unlike teletype printers) and a program can display an interface that could be called visual as well as just textual [23]. This allows programmers to create the first computer games and text editors that rely on this capability of video display terminals.

3.1.3 Graphical

Further reduction of the amount of time a processor needs to execute individual operations results in the ability for the computer to communicate with multiple input/output devices in realtime. Typical devices that a modern computer program expects to have access to through the operating system include a color monitor wherein each pixel in the monitor has a separate referenceable address, a graphics card to help with processing of 2d/3d graphics operations, a mouse, a keyboard, and a sound card connected to audio speakers.

Much of the common grammar of all the popular graphical user interfaces in-use today are derivatives of two particular projects. The first is called NLS/Augment (NLS stands for oN-Line System) and was designed by Douglas Engelbart and his team at the Stanford Research Institute. During a famous, public demonstration of the system in 1968 (often affectionately referred to as "The Mother of All Demos"), Engelbart proceeded to demonstrate the use of a computer mouse, a graphical display with multiple windows, and hyperlinks among many other notable advancements in human-computer interaction.

The second groundbreaking project, the Xerox Alto, came from the Xerox Palo Alto Research Center (PARC) in 1973. It is perhaps the first computer designed from inception to be dedicated to the use of a single person, hence the introduction of the term *personal computer*. Although the monitor displays only black and white pixels, the graphical user

interface of the operating system contains buttons, windows, scrollbars, sliders, and many of the logical GUI components that are standard components of any program with a GUI. It is followed at PARC by the Xerox Star in 1981, which takes GUIs a step further towards their current incarnations and introduces the ability to share resources across a local area network [24].

It is primarily the ideas that these two projects consolidate and introduce to the wider world that eventually become the impetus for the recognition of HCI and, of more precise relevance to this paper, user interface design as important areas of study and relevant to the broader computer science community.

3.1.4 Five Foci of Interface Development

It is clear from history that, in some sense, computing machines have always had user interfaces. What changes is the the level of attention that any particular layer in the computing interface receives during various periods of history. Grudin [19] introduces a helpful framework for understanding this change that takes place during the second half of the 20th century as movement through five different foci of interface design.

1. Initially, during and before the days of punch cards, most users of computers are electrical engineers primarily concerned with working directly with the hardware (flipping switches, replacing vacuum tubes, wiring ports and circuits). Conceptually, the computer interface is located at the hardware itself.

2. Afterwards, the primitive operating systems and batch monitors place the majority of focus on the task of programming and the environment in which commands are executed and requests are made and begins shifting focus away from the hardware. The development of ever higher-level languages and environments eventually replaces the need to be familiar with hardware particulars. At this point, the primary interface in these systems is at the level of the software code and programming.

3. As machines begin to perform their operations more quickly, systems become interactive and users are not expected to be able to interact with the computer through computer code, the interface grows into the canonical monitor, keyboard, and mouse of the 1990s and 2000s. It makes sense for designers of computer systems to become more concerned with issues of perception and motor control.

4. In recent years it has become practical to interact with the computer in a more conversational/realtime manner and system designers focus on the entire process of a user interacting with the system over an extended period of time, from initial exposure to practiced expert interaction. The focus of the interface designer becomes less trained on issues of ergonomics and visual perception and more trained on deeper cognitive issues like learning and problem solving. The interface now encompasses the proclivities of the mind.

5. Impressively, Grudin predicts the advent of social networking in what he labels *groupware*, the fifth focus of interface development. When a single system serves a large group of interacting people (setting this scenario apart from the earliest batch systems where different users rarely interacted) as opposed to an individual, the user interface becomes entangled with the dynamics of interaction between the members of the group.

Ultimately, Grudin cites these observations within the context of the framework of five foci to support the assertion that the history of the computer is one of the computer extending its interface with the world outward from its hardware implementation and deep into the intangible aspects of human behavior and a broader understanding of how we accomplish tasks in everyday life [19].

3.2 Task Analysis

Task analysis is a class of techniques used in HCI to guide the design of user interfaces. One of the central assumptions of user interface design and HCI is that different tasks require different interfaces. This is mirrored in the history of user interfaces. As the users and tasks performed with the computer change, so does the interface. A corollary of this premise is that thinking carefully about the nature of the tasks that are meant to be accomplished through the interface results in better design decisions. Crystal and Ellington performed an in-depth comparative analysis of the dominant techniques and provide a good introduction to the motivations of task analysis:

> "Practitioners and researchers routinely advocate building user-centered systems which enable people to reach their goals, take account of natural human

> limitations, and generally are intuitive, efficient and pleasurable to use (Preece, Rogers and Sharp, 2002). Central to the design of such systems is a clear understanding of what users actually want to do: What are their tasks? What is the nature of those tasks? Many techniques have been proposed to help answer these questions. Task analysis techniques are particularly important because they enable rigorous, structured characterizations of user activity. They provide a framework for the investigation of existing practices to facilitate the design of complex systems.
>
> Task analysis is especially valuable in the context of human-computer interaction (HCI). User interfaces must be specified at an extremely low level (e.g. in terms of particular interaction styles and widgets), while still mapping effectively to users high-level tasks. Computer interfaces are often highly inflexible (when compared to interacting with a physical environment or another person). This inflexibility magnifies the impact of interface design problems, making the close integration of task structure and interface support especially crucial." [25]

Broadly, task analysis consists of the observational and heuristic analysis of the physical, mental, and contextual requirements for performing a specific task. As such, even in its most rigorous and quantitative forms, task analysis typically involves less quantitative methods like discourse analysis, contextual inquiry, and video analysis.

The roots of scientific task analysis go back to 1911 when Frederick Taylor published *The Principles of Scientific Management* [25]. Taylor was interested in improving manufacturing productivity and incorporating understanding of human factors into work methods. Known commonly as Taylorism, he argued that managers should rigorously systematize the organization of workers based on empirical evidence. Of course, it is more accurate to refer to Taylor's discipline as something like job design; however, the relevant point here is that effort was being made to examine the relative efficacy of performing a task using different methods. The psychological component of such job design is first examined by Harvard Business School between 1927 and 1932 at the Western Electric Hawthorne Plant. The studies essentially conclude that the psychology of individuals with the workplace contribute significantly to what workers produce and expect from their jobs.

It soon becomes commonplace for industrial engineers to incorporate analyses of production methods to improve interaction between humans and machines as the efficiency

of machines became as important as the efficiency of humans in production methods. As computers become a typical machine that humans interact with in the workplace and the power and flexibility of computers as tools expands, human-computer interaction (HCI) evolves into its own discipline and broadens its scope.

This increased flexibility means that computers become entangled in new areas of human behavior like music. Techniques are developed to deal with this greater scope and complexity required of task analysis and each technique focuses on different aspects and contributes different insights into the nature of a human task.

An important point that should be extracted from this history, regarding the use of HCI techniques like task analysis to analyze artistic systems, is that HCI and thus user interface and data visualization design, bear the fingerprints of a discipline that ultimately derives from the standpoint of increasing productivity and improving job performance. Therefore, the principles advanced by HCI should always be evaluated (and potentially ignored) in the broader context of particular scenarios [26]. Despite this reality, task analysis is the primary means of analyzing the mapping task in Chapter 4 and the mindset of this analytical technique is the basis of the process used to arrive at the final design of Vizmapper for this thesis.

3.3 Graphical Perception and Data Visualization

> "Graphics is the visual means of resolving logical problems."

This idea from Bertin [27], who is one of the first to provide a theoretical foundation to data visualization, summarizes the hope for a dedicated interface for configuring a Mapper network. It is in this context that *data visualization* or *information visualization* is understood to be "the computer-assisted use of visual processing to gain understanding" by Card [28]. The task of mapping is a logical problem, as well as an artistic one. It is a logical problem in the sense that one cannot make connections haphazardly between any pair of input and output signals and expect the network mapping that is produced to be interesting as a DMI or meet the goals of the design team. One must still infer, through some form of reasoning, the suitability of a particular signal connection in the context of the particular devices on the network, the nature of the signals that the devices produce, and the broader artistic intentions of the design team.

It could be that one output signal generates floating-point decimal values and one input signal accepts only integer values. Or if a specific output signal tends to generate a static signal regardless of the gestures or environmental dynamics sensed by the control mechanism of the MSN, it would make for a very dull and non-dynamic performance to connect this output signal to the input signal controlling the pitch of an audio synthesizer. Of course, as previously stated, one does not typically make use of metrics like productivity or performance in an artistic context, so the logical problem is different. But even an artistic project has goals and intentions, though they are considerably harder to define in words. Visualizing a Mapper network effectively within a user interface will likely help a team of people (as long as there is agreement about the artistic and other intentions of the scenario that the MSN is to be used in), resolve these logical problems involving network topology and signal transformation more effectively.

Graphics are thought to have at least two distinct uses. The first is communicating a set of information. The second is acting as a medium for graphical processing, defined as using the perception and manipulation of graphical objects to understand the information [28]. Any graphical interface that is monitoring a Mapper network will likely incorporate both uses of graphics. The interface needs to effectively communicate the current state of the network, including the namespaces and properties of all registered devices and signals and the set of signal connections and transformations between outputs and inputs. Also, the interface must allow the user to manipulate the graphics to connect and disconnect signals and apply functional transformations to the signal data.

It is useful to categorize all possible data into three types, as distinguished by their function [27][28]:

Nominal - data values that are understood as either = or NOT = to each other - examples are the set of religions, colors, and professions

Ordinal - data values that can be understood using a greater-than or less-than relation in some way - an example is the set of the days of the week

Quantitative - data values that can be manipulated with arithmetic - an example is the population of a city

Note that each successive functional type is a special case of the previous types. All quantitative data is necessarily ordinal data and nominal data. All ordinal data is neces-

sarily nominal data. Conversely, only some nominal data is ordinal and only some ordinal data is quantitative.

Nominal	Ordinal	Quantitative
Position	Position	Position
Color Hue	Density	Length
Texture	Color Saturation	Angle
Connection	Color Hue	Slope
Containment	Texture	Area
Density	Connection	Volume
Color Saturation	Containment	Density
Shape	Length	Color Saturation
Length	Angle	Color Hue
Angle	Slope	Texture
Slope	Area	Connection
Area	Volume	Containment
Volume	Shape	Shape

Table 3.1 Ranking of perceptual tasks for each fundamental data type (N.B. black background indicates non-applicability of the task for that data type) [2]

Depending on the functional type of the data being visualized, certain forms of visual differentiation are more effective than others. Card et al. [28] use the distinction between *automatic processing* and *controlled processing* capacities in human visual processing to inform an analysis of visualization techniques. Automatic processing works on visual properties like position and color and is highly parallelized, but lacks the complexity of highly structured visual information. Controlled processing notably works on text and can deal with more complex information, but is not very parallelized because it requires conscious attention [29]. This is why it is easier to assess a large amount of data displayed as graphics that take advantage of automatic processing than words and numbers that require controlled processing.

It is common in the literature to decompose any information graphic into *marks* (points, lines, areas, surfaces, or volumes), the *automatically processed graphical properties* of those marks (position, color, size, shape, etc...), and the *controlled processing graphical properties* of those marks (if the mark has external semantic meaning as language or numbers).

A well-cited study by Cleveland and McGill [30] forms the basis of a theory about the relative accuracy with which people infer quantitative information among the base automatically processed perceptual tasks that can be employed in a graphical presentation of data. This theory is expanded upon by Mackinlay [2] to form the ranking displayed in Table 3.1. To clarify one point of possible ambiguity - color hue, color saturation and density are equivalent to the common hue, saturation, value (HSV) representation of color. This theory is useful for deciding how to map the available, perceived graphical properties to the variable components of a data set because it allows one to rank the effectiveness of possible graphical languages. Mackinlay communicates the importance of such a ranking through the common sense *Principle of Importance Ordering*:

> "Encode more important information more effectively." [2]

Of course, this presumes a method for analyzing the different components in a data set and characterizing the relative importance of the different variables present in a data set.

The most extensive theoretical framework for the display of graphical information is presented in a book called the *Semiology of Graphics* by Bertin [27]. He describes a three stage process for analyzing the data in preparation for designing a visualization.

1. The first stage is determining the number of *components* or *variables* present in the data. A component is defined as a concept in the data that varies across a number of *elements* particular to the component. For example, the component of "output signals present on a Mapper network" varies across the set of output signals labeled by their declared network name. Similarly one could define another component encapsulating the variation in the available input signals.

2. The second stage is determining to *length* of each component in the data set. The length is defined as the number of elements which are uniquely identifiable in the component. If the component is "the states in the United States", the length of the component is 50. Bertin's concept of *length* is very similar to the concept of *cardinality* explored in Chapter 2. The cardinality of a component is important for choosing the perceptual graphical property to map to a component in a data set because a high cardinality component requires a mapping to a perceptual task that is able to be finely differentiated whereas a lower cardinality component can be

mapped to a task that people tend to differentiate more coarsely. Since each graphical property of a particular mark can only be used to represent one component for clarity, it is important not to waste a high resolution perceptual task on a low cardinality component.

3. The third stage is determining the functional type of the component as previously defined (nominal, ordinal, or quantitative). Bertin calls this property of a component the *level of organization*.

This analysis can then be used in combination with Table 3.1 to make sure the data of each component is transcribed with a visual variable possessing the same cardinality and level of organization and thus contributing to an efficient visualization.

This theoretical framework will be used in Chapter 4 to weigh possible visualization choices for Vizmapper. The final aspect of data visualization researched is the idea of *clutter* and how it modifies the effectiveness of a visualization. First a definition:

> "A good visualization clearly reveals structure within the data and thus can help the viewer to better identify patterns and detect outliers. Clutter, on the other hand, is characterized by crowded and disordered visual entities that obscure the structure in visual displays. In other words, clutter is the opposite of structure; it corresponds to all the factors that interfere with the process of finding structures. Clutter is certainly undesirable since it hinders viewers understanding of the content of the displays." [31]

The idea of clutter is not very precise, however it neatly summarizes what ought to be kept in mind at all times when designing a visualization. The purpose of a visualization is allowing a user to see patterns that would require much more time to see using tables of numbers or text. If a visualization can be changed to reduce the number of visual entities or the sensation of crowding without changing what is being communicated, then this change is a win. It is important not to forget the central purpose of a visualization and not get lost in pretty colors and shapes.

3.4 Recall and Recognition

There are certain aspects of cognitive functionality that are well-studied and easily applied to interface design. These provide useful theoretical guides for designing efficient user

interfaces. The remainder of this chapter outlines those hypotheses that are most relevant to the Vizmapper interface.

It is common to understand human information retrieval in terms of two types of retrieval, recall and recognition. In the recall process, information is reproduced from memory. In the recognition process, a piece of information induces the knowledge that the information is not new; it has been seen before and the information acts as a cue [18]. In this respect, recognition is a less complex cognitive activity. Recall can be assisted with cues that allow an individual to speed access to the information in their memory. One proven cue is the use of categories.

Modern understandings of recognition and recall attribute the relative effectiveness of the two processes to the way in which perception and memory are interconnected [32].

> "Activating a memory consists of reactivating the same pattern of neural activity that occurred when the memory was formed. Somehow the brain distinguishes initial activations of neural patterns from reactivations - perhaps by measuring the relative ease with which the pattern was reactivated. New perceptions very similar to the original ones reactivate the same patterns of neurons, resulting in *recognition* if the reactivated perception reaches awareness. In the absence of a similar perception, stimulation from activity in other parts of the brain can also reactivate a pattern of neural activity, which if it reaches awareness results in *recall*." [32]

HCI practitioners have long known about this interesting byproduct of the evolution of the brain as the creators of the Xerox Star point out in explaining their design decisions for the system.

> "Traditional computer systems require users to remember and type a great deal just to control the system. This impedes learning and retention, especially by casual users. Star's designers favored an approach emphasizing recognition over recall, seeing and pointing over remembering and typing." [24]

This basic understanding of recall and recognition provides an easily applicable rule for building an effective interface.

3.5 Hierarchical Structures

Hierarchies and taxonomies (classification schemes arranged hierarchically and used often in science) are widely used to reduce the complexity of a data set (like the set of all the animals in world is organized by kingdom, phylum, class, etc...) by dividing up the data set into smaller subsets and relating the subsets to the larger superset. It is believed that this general process plays a central role in recall, recognition, and problem solving [33][34][35].

The basis of this theory stems from a famous paper by George A. Miller called *The Magical Number Seven* [33]. Central to Miller's two central hypotheses is the common concept of the *bit*. Briefly, one bit of information is defined as the amount of information that is required to make a decision between two equally likely alternatives [33]. It is precisely the amount of information that is required to know whether a flipped coin is heads or tails. Two bits is the amount of information needed to decide between 4 equally likely alternatives. Three bits is the amount of information needed to decide between 8 equally likely alternatives and so on...

The first hypothesis is that a person's ability to distinguish between elements of a set, regardless of what sense is perceiving the elements, is roughly limited to between 2.3 and 2.8 bits (this is not *exactly* what Miller says), which corresponds to 7 plus or minus two elements. This is referred to as the *span of absolute judgement*. Note well, that this is only true of elements that vary along one perceived dimension. For instance, human faces are perceived among multiple visual property dimensions. Otherwise, we would never be able to distinguish between the faces of more than 10 people! Also, there are well-known exceptions to this rule. A musical example is the ability of a person with *absolute pitch* to accurately distinguish between around 60 different pitches [33]. The exact number of bits understood to be the average span of absolute judgement has changed as our understanding of the brain has improved since 1956, but what has not changed is that the number is lower than one might expect.

The second related hypothesis has to do with the *span of immediate memory*. Immediate memory is the abstract model of the temporary memory store that one uses during conscious reasoning. In this case, the relevant parameter is no longer a bit, but a *chunk*. The number of chunks that humans can hold in immediate memory and reason about is also about 7. A chunk can represent a similarly low, "magic" number of bits (two to three) in immediate memory. However, a chunk can also represent a similar number of chunks instead of bits,

which allows one to reason about a large number of bits of information as long as they are hierarchically organized such that lower level chunks are represented by higher level chunks.

> "In the jargon of communication theory, this process would be called *recoding*. The input is given in a code that contains many chunks with few bits per chunk. The operator recodes the input into another code that contains fewer chunks with more bits per chunk. There are many ways to do this recoding, but probably the simplest is to group the input events, apply a new name to the group, and then remember the new name rather than the original input events." [33]

Given this, it is not altogether surprising that the following is also known. For a given set of unknown associative connections between a set of elements, presenting the connections as a graphical hierarchy aids learning the structure of connections *much* more than a list of connections [34]. This fact is immediately applicable for visualizing a Mapper network.

In summary, if an interface expects a user to make decisions involving a set of high cardinality, there must be a form of hierarchical organization of the elements of the set. Otherwise, the user will not be able to proceed very effectively or accurately.

3.6 Filtering and Information Seeking

When dealing with a data set with a high cardinality for one component or another, filtering becomes a very attractive feature for a user interface to have. This is especially true when the goal of a user interface or data visualization is not only to improve the ability for a user to extrapolate high level integrative knowledge about the set of data, but to focus on and select specific data for manipulation, as required by the Vizmapper interface. This focus is related to data visualization, but is referred to as *visual information seeking* (VIS) to distinguish it from data visualization.

Filtering, like hierarchical organization, provides a mechanism for reducing the perceived complexity of a data set. Unlike hierarchical organization, which preserves the overall cardinality of the data set, filtering temporarily restricts the visible set to a subset within the complete set that meet the parameters of the filter. Filters can be implemented graphically by using sliders, drop down menus, and other common graphical widgets or use textual

queries. Hierarchical organization can itself be used as a filtering mechanism by acting as predefined set of salient queries (for example, to restrict the set of all animals to mammals from a typical hierarchical biological classification).

Most of this understanding of filters in the context of VIS comes from a paper by Ahlberg and Schneiderman [36]. The relevance of filters to the problem of mapping in a large MSN is best summarized as follows:

> "The key to these principles is understanding the enormous capacity for human visual information processing. By presenting information visually and allowing dynamic user control through direct manipulation principles, it is possible to traverse large information spaces and facilitate comprehension with reduced anxiety." [36]

The principle of *tight coupling* in a user interface is of particular interest when including filtering capabilities in an interface because it makes clear the distinction between filtering and searching. Tight coupling applies to many aspects of a user interface and broadly refers to maintaining tightly coupled consistency between the state of a data visualization and any user actions that are supposed to change the state of the system. If it is possible for a user to break this consistency between the visual state of the system and the actual state of the system the user will likely be confused.

Typically, when searching a database, the system state begins by displaying an empty set of data because no query has been made. Once a query is executed than the interface displays a subset of data (in graphical or textual form) that corresponds to the elements of the entire data set that match the search query. Conversely, when filtering a data set, the initial state of the system with an empty query is to display the entire set of available data and to reduce the amount of information on the screen as the query becomes more specific. The difference is admittedly subtle. However, when a user is introduced to a large data set for the first time, the filtering convention is much less confusing because it is much more disorienting to be presented with a empty set of data than it is to be presented with too much data [36]. In the former case, it is more difficult for a user to determine what the first action to take should be.

3.7 Summary

Often when dealing with modern computers systems, it is easy to forget all of the tiny, interrelated decisions that have influenced the interfaces that are used by people when interacting with computers. Yet, as the history of computers illustrates, it has a direct influence on how people end up using computers. As the advent of DMIs is a relatively recent happening in the longer history of digital computers, it makes sense to appreciate how specific decisions about computer interfaces broadened the use of computers and how similar decisions about the interface of a mapping tool might encourage a similar broadening of the use of computers in music.

Human-computer interaction practitioners and cognitive scientists provide us with several methodological tools to guide such a design of a graphical user interface. However, the purpose of introducing these aspects of cognition and effective design that others have discovered is not to provide a set of *rules* to govern the decisions that are made in the Vizmapper design. Many of these connections between interfaces and cognition are not rigorously provable in the sense that the principles of physics or electrical engineering are arguably provable. One of goals of the introduction of this thesis, and specifically the initial quotation, is to make this clear from the outset.

However, this is the minimum due diligence that ought to be applied when making a good faith attempt to *improve* how someone accomplishes a task and encourage the proper mindset when performing the task. The history and research reviewed in this chapter provide a solid theoretical foundation for tackling the broad problem of designing a user interface, and the specific problem of visualizing and manipulating a large data set through a graphical user interface.

Chapter 4

Vizmapper

With the benefit of the research and context that is presented in Chapters 2 and 3, it is now possible to embark on an informed analysis of the mapping task for an MSN and to make informed decisions about the implementation of the system.

The graphical user interface to Libmapper that is the result of these design choices is called Vizmapper.

The task of mapping is understood, from Chapter 2, to be creating a set of associations and functional transformations between a set of signals being output by a set of devices (usually with sensors) and a set of inputs made available by various devices (usually with audio synthesis modules) capable of receiving signals.

The context for the mapping task is understood to be one where many programmers, engineers, composers, and/or musicians are interested in experimenting with interactive musical systems for use in a creative (as opposed to productivity or analysis) context.

4.1 Task Analysis of DMI Mapping

The section on task analysis in Chapter 3 concluded that, as a technique, it is best-suited to understanding how to design interfaces that maximize productivity as opposed to channeling creative expression. This may or may not present a problem depending on whether we choose the characterize the use of computers, in the context of performing the task of mapping, as an act of expression or an act of productivity. Atau Tanaka offers a line of thinking that applies to tools and instruments, but may be of use in attempting to resolve this problem [37].

> "A tool can be improved to be more efficient, can take on new features to help in realizing its task, and can even take on other, new tasks not part of the original design specification. In the ideal case, a tool expands the limits of what it can do. It should be easy to use, and be accessible to [sic] wide range of naive users. Limitations or defaults are seen as aspects that can be improved upon.
>
> A musical instrument's raison-d'etre, on the other hand, is not at all utilitarian. It is not meant to carry out a single defined task as a tool is. Instead, a musical instrument often changes context, withstanding changes of musical style played on it while maintaining its identity. A tool gets better as it attains perfection in realizing its tasks. The evolution of an instrument is less driven by practical concerns, and is motivated instead by the quality of sound the instrument produces. In this regard, it is not so necessary for an instrument to be perfect as much as it is important for it to display distinguishing characteristics, or "personality". What might be considered imperfections or limitations from the perspective of tool design often contribute to a "personality" of a musical instrument.
>
> Computers are generalist machines with which tools are programmed. By itself, a computer is a tabula rasa, full of potential, but without specific inherent orientation. Software applications endow the computer with specific capabilities. It is with such a machine that we seek to create instruments with which we can establish a profound musical rapport.
>
> The input device is the gateway through which the user accesses the computer software's functionality. As a generalist device, generalized input devices like the keyboard or mouse allow the manipulation of a variety of different software tools. Music software can be written to give musically specific capabilities to the computer. Input devices can be built to exploit the specific capabilities of this software. On this general platform, then, we begin to build a specialized system, each component becoming part of the total instrument description."

This line of thinking suggests that the extent to which Vizmapper is not a tool limits the extent to which the principles of user interface and data visualization design ought to play a role in the design process. HCI is much better suited to evaluating more objective

notions like utility, tasks, and functionality than more subjective notions like personality, quality of sound, and musical rapport.

It is clear from Tanaka's definitions that Vizmapper serves as a tool rather than an instrument. However, the fact that Vizmapper is specifically a tool for accomplishing the task of creating and modifying mappings *within a musical instrument* suggests that the design must be treated more subtly in this particular context. As the purpose of Vizmapper is partly to make the connections between components in a musical instrument more malleable and more susceptible to experimentation for groups of non-programmers, the task bears some resemblance to a non-utilitarian task. It is reasonable to assume that if the evolution of an instrument is motivated by the quality of the sound that the instrument produces, then similarly the evolution of a mapping interface is motivated by the quality of the mappings and DMIs that the interface produces. This reality should be evaluated in parallel with the understanding that configuration of a mapping is a somewhat utilitarian task; either the interface allows a team to configure the mapping efficiently or it does not.

To see this, the complex task of designing a musical instrument is deconstructed using an analysis process called *hierarchical task analysis* [38].

Wanderley and Depalle distill the last decade of thinking into understanding the abstract components of digital musical instruments by decomposing a DMI into 3 main components [39]:

The physical interface containing the sensors, actuators, and physical body of the instrument.

The software synthesis system which creates both the sonic output of the instrument and any visual, haptic and/or vibrotactile feedback.

The mapping system in which connections are made between parameters of the physical interface and those of the synthesis system.

The design of these 3 components can be regarded as the 3 main subtasks of the overall task of designing a DMI. Each of these subtasks are themselves composed of more granular tasks, thus forming the beginnings of a hierarchical structure representing an analysis of the overall task.

Designing the physical interface

choose sensors that are capable of detecting the desired physical phenomena or gestures

choose actuators that are capable of inducing the desired physical phenomena or feedback

choose sensors that are made available as outputs for connections to the inputs of the synthesis system

choose actuators that are made available as inputs for connections from the outputs of the synthesis system

choose structural/aesthetic materials that combined with the sensors and actuators will form the composite form of the physical interface

design the shape, look, feel of the composite physical interface

Designing the software synthesis system

choose a mechanism/algorithm for performing sound synthesis in software

choose a mechanism/algorithm for generating visual, haptic, and/or vibrotactile feedback

choose parameters of sound synthesis that are made available as inputs for connections from the outputs of the physical interface and/or other components of the composite synthesis system

choose parameters of visual, haptic, and/or vibrotactile feedback synthesis that are made available as inputs for connections from the outputs of the physical interface and/or other components of the composite synthesis system

choose parameters of sound, visual, haptic, and/or vibrotactile synthesis that are made available as outputs for connections to the physical interface and/or other components of the synthesis system

Designing the mapping

choose a subset from the complete set of available outputs to start connections from

choose a subset from the complete set of available inputs to connect to specific outputs

create functional transformation mappings that specify how (if at all) to modify source signals from outputs to generate the desired destination signals for the inputs

This hierarchical model of DMI design generalizes well to anything from a self-contained handheld instrument to a massively distributed system spread over a large physical environment. It also works regardless of whether it is an individual or a team that is engaged in the instrument design. This decomposition also makes more clear why perhaps the mapping system is deserving of a dedicated system and user interface to manipulate.

A good analogy is the choice of what collection of LEGO blocks to use as opposed to how to put the LEGO blocks together once the blocks are chosen. Procedurally, one chooses what types of blocks to use before deciding how to put the blocks together. Similarly, it makes sense to make certain choices about what pieces will be used for the physical interface and the software synthesis system before deciding how the mapping system will put the two collections of inputs/outputs together. It is difficult to map between two sets of signals when the sets of signals are not already defined. Of course, this model laying out the necessary choices to be made says nothing about *what* choices should be made. That process is entirely dependent on the artistic intentions of the design team and the goal of a mapping tool is only to allow the choices that are made in the mapping subtask to be implemented as efficiently as possible.

4.2 Implementation

As a piece of software, there are a limited number of options for implementing the system while ensuring that the system can be reliably used on the variety of systems that a team designing a musical system is likely to be using in the present and future. As of the time that this system has been implemented, the most common form factors for computers are desktops, laptops, tablets, and smartphones. Each form factor is typically loaded with one of a small collection of operating systems that are widely available and easy to find expert knowledge about.

Desktops/Laptops Windows (XP, Vista, 7), Mac OSX, Linux (Ubuntu, Debian, etc.)

Tablets/Smartphones Android, iOS

This is not an exhaustive list, however the point is that it is not practical (especially in a research context) to develop and maintain several versions of a software system that will work on a wide variety of operating systems and devices. However, the fact is that people will use whatever system they have available and the variety of devices that an artist or engineer is likely to use to interact with a sensor network is increasing. To design a practical system that is meant to work in a collaborative context, it is crucial to adapt to this reality.

Before beginning development on a software system, it is necessary to choose a development environment and a deployment environment. There are any number of very stable *development* environments that enable a software developer to write a single collection of programmer code and translate the code into executables that can run on Windows, OSX, and Linux machines. These choices effect the developer of the software and not the user of the software. The choice of *deployment* environment is much more consequential to the user because it effects how they will actually run the executable. For an explanation of technical terms pertaining to this chapter review the appendix.

There are three common methods for a user to run software on their chosen machine.

1. The first method is for the user to run a compiled machine code executable, which means that the developer has written code and translated the code with a compiler into machine code that can be executed without any further translation on the user's specific machine and operating system. If there is no available compiled executable for a specific machine architecture, the user can obtain the programmer code and compile it into an executable that works on their specific machine, as long as a compiler exists for their machine. This is how many video games are deployed.

2. The first method is an option when the software is developed in a *compiled language* environment. There are also environments that use *interpreted languages* or *script languages*. An interpreted language differs from a compiled language because unlike a compiled language that uses the process of translating the program code into machine code with a compiler and producing a machine code executable to run the program, an interpreted language uses the process of interpreting the program code line by line directly when the program is run. The interpreter is responsible for interfacing with the machine for the interpreted code when the executable is run. In a interpreted language the programmer code and the executable are often the same file. Conversely,

a compiled executable interfaces directly with the machine because it has already been converted into machine code.

A web application is essentially an interpreted executable because a web browser that runs HTML, CSS, and Javascript downloads the programmer code and uses an internal interpreter to execute the program. It does not download a compiled executable. Distributing software as an interpreted script executable to be run in an interpreter is attractive because the burden for implementing the interpreter for various different machine architectures is moved to the developer of the programming language and the developer can count on a user being able to run their program as long as the interpreter for the language is available for the user's machine. The developer can write *machine-independent* code.

3. Environments like Java use a third strategy and run software on *virtual machines*. A virtual machine language is somewhat like a hybrid of a compiled language and interpreted language because a virtual machine language is compiled, but it is compiled to an intermediate machine code that is not the machine code of the actual physical machine that is running the program. It is compiled to the "machine code" of a software virtual machine that must be implemented for all machines that one would like to run the program on. This is useful because the user can be given a single file that is not programmer code like a script executable, but the developer can still count on the file being able to run on any machine that has the virtual machine available on the system.

Assuming the intent is for multiple team members to be running the Vizmapper interface to enable simultaneous work on the mapping of a network and that there is presumably a local network available to allow the various devices on the sensor network to communicate with each other, suggests Vizmapper ought to be implemented as a web application.

A web application is typically distributed as a two part software system where one part is called the *client* and the other part is called the *server*. One server can serve the web application to a large number of clients. This is the source of another advantage of web applications. A web client runs in a web browser and therefore can be run on any machine that includes a modern web browser, which includes every operating system listed at the beginning of this section. All modern web browsers provide an increasingly identical deployment environment. Since Libmapper is compiled code, the server software can take

care of interfacing with the library and communicating through the Mapper protocol with the rest of the Mapper network. Despite the fact that the server is more restricted, in terms of what deployment environment it can be run on because of the Libmapper dependency, the server need only run on one machine. Also, a web application is much easier to deploy on a collection of heterogenous machines. At this point in time, a web application reduces the likelihood that the team will be forced to huddle around the chosen few machines that can run the client, even if all machines cannot run a server with a more restrictive deployment environment.

Stephen Sinclair, the developer of Libmapper and a member of IDMIL, also developed a basic server written in Python (a popular interpreted language) that interfaces with Libmapper and provides a convenient interface for a web browser client to indirectly speak the Mapper protocol to the local network of devices by communicating with the server. This web server is called Webmapper. Together, Webmapper/Vizmapper provides a server/client web application that can be used to configure a Mapper network.

4.3 Application of User Interface Design Principles

If the user of Vizmapper is a human user configuring an MSN, then Vizmapper is the "user" of Webmapper. Webmapper is then the "user" of Libmapper. Libmapper provides an interface for Webmapper to speak the Mapper protocol and Webmapper provides an interface for Vizmapper to indirectly speak the Mapper protocol. Now the task remains to decide how the Vizmapper interface will present the signals and devices present on the network to the human user and allow the user to configure mappings between these signals and devices through Webmapper.

As a reminder, the subtask of *Designing the mapping* was further broken down as follows:

choose a collection of available outputs to start connections from

choose a collection of available inputs to connect to specific outputs

create mappings that specify how (if at all) to modify source signals from outputs to generate the desired destination signals for the inputs

From the hierarchical task analysis of designing a DMI, it can be argued that the task of designing a mapping is a less complex task than the task of designing a physical interface or designing an audio synthesis system. This lack of complexity in the task is one of the reasons why it is possible to even contemplate performing the other task through a single interface. Building an interface for the task of designing a physical interface or designing an audio synthesis system is less manageable in this way.

Instead, the complexity of the task lies in the complexity of the Mapper network and what the task of *choosing* actually entails. Since the focus is now on the mapping task, it is helpful to see if it is possible to be even more granular in the hierarchical decomposition of the task.

choose a collection of available outputs to start connections from

- view the complete set of output signals
- find an output signal to connect to an input signal
- select the output signal

choose a collection of available inputs to connect to specific outputs

- view the complete set of input signals
- find an input signal to connect to an output signal
- select the input signal

create mappings that specify how (if at all) to modify source signals from outputs to generate the desired destination signals for the inputs

- define a functional transformation/signal conditioning mapping between the output signal and input signal
- connect the signals

This clarifies the individual steps implicit in the overall task and suggests how to distribute the steps needed in the interface. Since the screen space on a monitor is limited and the most important data about the Mapper network are the names of the signals and the mappings between them, devoting as much space to the visualization of the signals and their connections is a priority.

The first decision made is to separate the viewing and browsing of the MSN from the editing of mappings. Vizmapper treats these two sets of subtasks as two different modes. Entering viewing and browsing mode means that it is not possible to modify the mappings of the signals being browsed. Entering editing mode, focuses the view on a subset of signals from the complete set and means that it is not possible to browse the larger set of signals. This decision is made because viewing and browsing does not actually effect the Mapper network, only editing a mapping effects the network. Preventing both modes from being accessed on the same screen makes the conceptual separation between the modes clear and reduces the likelihood of accidental modifications.

When it comes to viewing and browsing, since the interface is being designed with larger MSNs in mind, there should be a way to filter the complete set of signals to a subset of signals. As discussed in Chapter 3, this can be accomplished by pointing and clicking to utilize recognition processes or it can be accomplished by typing to encourage the use of recall processes. Although finding a specific signal through name recognition is less straining then remembering the name of a specific signal and typing it, it is not necessarily a simple decision. In the case where a user has been working with the same Mapper network for an extended period of time, it may become tedious to point, click, and browse their way to a specific signal even though they know the exact name of the signal they are looking for. In this case, the physical process involved to select the signal becomes more consciously cumbersome than the cognitive process involved in recognition or recall.

A single text input box, for entering a text matching filter query, occupies very little vertical space and adds functionality that is helpful for this type of user. The user can type in a portion of the complete OSC name of a signal and limit the view to only the signals that match the text in the input box. In order to display the union of more than one text query (show the signals that match the first text query OR second query OR third query, etc.), the user uses a space character to delimit the separate filter queries.

Typically, interfaces that allow the user to browse data using text queries allow the user to compose more complex queries (like ANDs or NOTs in addition to ORs). However, this functionality seems to deviate from the scope of the interface and add syntax complexity without a substantial benefit to the central task. Since the user is only searching within a set of names of signals and not a set of large text documents (like one might do with a database or the internet), it is likely this power is not required by most users. Additional features and complexity are avoided in Vizmapper, unless the functionality that is added

is significant to the mapping task as it has been analyzed.

4.4 Application of Data Visualization Design Principles

Following the graphical perception research in Section 3.3, before making decisions about the visualization of the network, the data set of a Mapper network will be analyzed using the 3 stage process of Bertin [27].

This involves determining the *components* in the data, the *length* of each component, and the *level of organization* (nominal, ordinal, or quantitative).

There are 3 easily distinguishable sets of data that a user concerns themselves with when mapping signals.

1. set of signals on the Mapper network, which are defined as OSC names that are part of a hierarchical namespace

2. set of associative connections between pairs of signals, each connection is defined by a source and a destination (the source of the data is typically an output signal and the destination is typically an input signal)

3. set of function transformations that are paired with each associative connection, which are defined using a function type, a mathematical function, and an allowable range of values for the input of the function (actually an output signal in Mapper terminology) and output of the function (actually an input signal in Mapper terminology)

It is useful to note that the most complex elements are the elements of the set of functional transformations. The least complex elements are the OSC names of the signals. This is taken simply from the number of variables that must be defined to fully define each element.

So the data set can be decomposed into *3* components, but it might also be decomposed into *4*. The first component, the set of signals, can be split into one component for output signals and one component for input signals. The question is whether one decomposition is more effective than the other. If it is assumed that the source of a connection will always be an output signal of a device and the destination of a connection will always be an input signal of a device, it makes sense to use *4* because it will always be necessary to select a source and a destination to form any connection and it will be efficient to graphically

distinguish between the 2 types of signal. Therefore, there are *4* components in the data set. The first component of the previous decomposition can be further split into:

1. set of output signals on the Mapper network, which are defined as OSC names that are part of a hierarchical namespace

2. set of input signals on the Mapper network, which are defined as OSC names that are part of a hierarchical namespace

The length of the output signals component and the input signals component are assumed to be *large* and within an order of magnitude of each other. There might be many more output signals than input signals or vice versa, however since there will be roughly one control mechanism device with output signals for each sound generation mechanism device with input signals in a scenario with DMIs, this is a reasonable assumption.

The length of the set of associative connections is bounded by the length of the set of input signals. This is because the Mapper protocol only handles *one-to-many* and *one-to-one* mappings as explained in Chapter 2. Therefore, there can be, at most, one connection for every input signal.

The length of the set of function transformations is equal to the length of the set of associative connections because every connection is paired with a functional transformation governing data transmission over the connection. This is not strictly true, however it is simpler to treat those connections with no active transformation as utilizing the function of $x = y$ with no bounded ranges (also known as a bypass) to indicate the lack of any signal conditioning.

The level of organization of all four components is *nominal*. None of the components have elements that can be ordered or manipulated with arithmetic.

So the conclusion of a Bertin-like analysis of a typical Mapper network data set is the following:

Components - four components

1. output signals
2. input signals
3. connections

4. functional transformations

Length - all four components are of approximately equivalent length

Level of organization - all four components are nominal

This analysis comes with one caveat in light of Chapters 2 and 3. There is one level of organization that Bertin does not address - hierarchical organization. This is a level of organization that seems relevant, especially given the importance of hierarchical organization in human cognition, yet is not directly addressed by the nominal/ordinal/quantitative categorization scheme. Due to the nature of OSC namespaces, the set of output signals and the set of input signals can be grouped hierarchically. This suggests a type of relation between signals that is not simply = or NOT =.

The problem of visualizing an abstract data set that has both hierarchical relationships between elements and connections between elements as components is problem that has been tackled in data visualization literature [1]. Holten refers to the hierarchically organized component as a set of *inclusion relations* and the set of connections as a set of *adjacency relations*. One example of a data set exhibiting a structure similar to a Mapper network data set is an academic citation network where publications are the lowest level elements of the hierarchy and departments and universities are represented by higher level groupings in the hierarchy. Connections between publications would represent one publication citing another [1].

Since we are dealing with nominal data with some hierarchical organization, let us focus on the most discerning perceptual tasks for nominal data using the ranking presented in Section 3.3. This reduced set of ranked perceptual tasks is shown in Table 4.1.

Nominal
Position
Color Hue
Texture
Connection
Containment
Density

Table 4.1 Ranking of top perceptual tasks for discerning between nominal data [2]

Inspired by Holten's own exploration of visualizing similarly structured data, the decision is made to represent signals with circles occupying different points in space. The heuristic reason for this decision is that the shape of a circle lends itself to a symmetrical and aesthetically pleasing presentation of the data and makes for an easy target for the user to click (more on this shortly). There are 4 components and it is important to distinguish between these components, first and foremost. Since position is already being used to distinguish between signals and at least one output signal and one input signal will need to be displayed for the connection and functional transformation components to be relevant, it makes sense to place a constraint on the *position* of signals that forces the output signals to be confined to the left side of the display and input signals to be confined to the right.

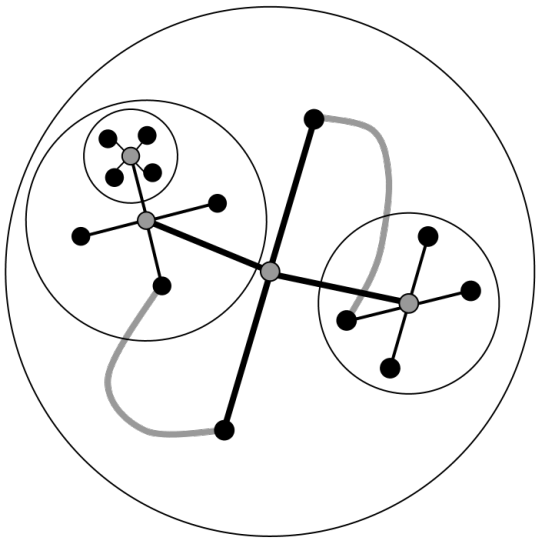


Fig. 4.1 Example of a balloon tree visualization with dark lines for inclusion relations and light lines for the adjacency relations - inspired by [1]

The decision is made to use *containment* to represent the inclusion relations between hierarchically organized elements. This is taken from the use of a *balloon tree* visualization by Holten in his own research. An example of a balloon tree is shown in Figure 4.1. Examining the figure, notice that *connection*, *containment*, and *density* are used to communicate specific relationships between elements. Again, there are two components being displayed: a set of inclusion relations and a set of adjacency relations. In this example, there are only two adjacency relations shown in the figure.

Connection is used to communicate both adjacency and inclusion. The curved, lighter

lines represent adjacency relations between terminal elements (for example, citations between publications). The straight, dark lines represent inclusion relations between different levels in the hierarchy. Density and shape are used to distinguish between the two. Density is also used to distinguish between terminal elements in the hierarchy (dark dots) and intermediate elements in the hierarchy (light dots, for example, representing departments and universities).

Containment and position are also used to communicate relations between the elements. Enclosing circles differentiate between levels in the hierarchy and different branches of the hierarchy on the same level. The center of every circle is the position of the intermediate elements defining what the circle represents (for example, department on one level of hierarchy, university on another level) and the elements positioned radially around the centers are the child elements (either terminal or lower level, but still intermediate, elements). Notice that terminal elements can either be placed directly on the first level of the hierarchy or on deeper levels depending on the specific inclusion relations of the data set (perhaps there are independent publications that are not associated with any particular organization in the data set).

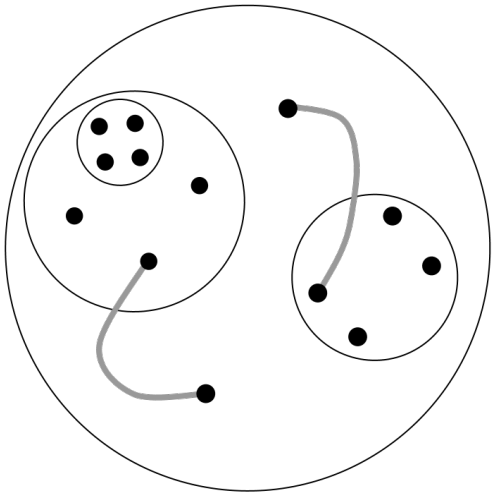


Fig. 4.2 Modified balloon tree visualization

The balloon tree used in Figure 4.1 is not as uncluttered as it might be. The lines used to communicate inclusion relations and the lighter dots placed at the center of every circle actually display the same data as the containing circles. This seems to be unnecessary

visual redundancy. Figure 4.2 effectively communicates the same data set as 4.1 with no loss of information and less clutter. The missing visual elements are implied.

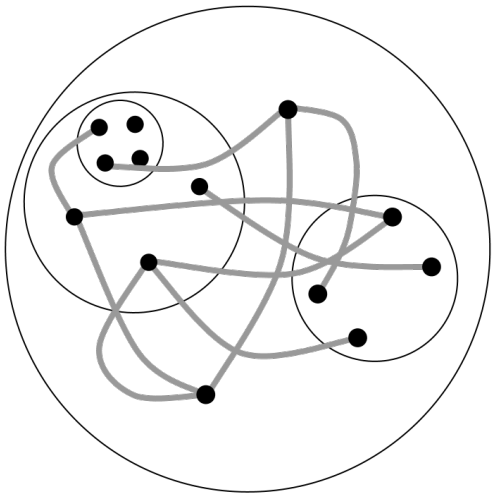
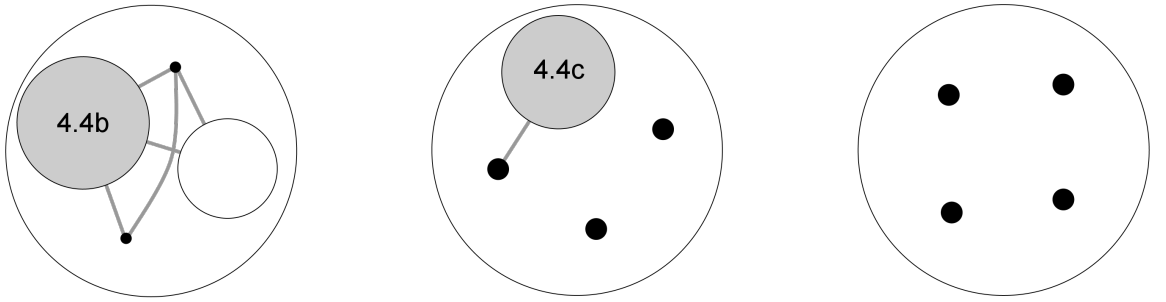


Fig. 4.3 Cluttered balloon tree visualization

When the number of elements in the data set grows large and there are many adjacency relations between the elements, there is a different type of clutter that arises involving the lines representing the adjacency relations like in Figure 4.3. This is a type of clutter that is more difficult to reduce. One can be more specific about how the connections are drawn, using bezier curves and bundling to organize the connections like one might bundle wires between mixers. This is the method that Holten uses [1]. In future work, using Holten's connection clustering algorithm in Vizmapper would be a net benefit, however it is not included at the time of this thesis.

As an alternative to Holten's method, it may be possible to use a different graphical property entirely. However, it is difficult to conceive of a graphical property that communicates connections between signals better than the graphical property of connection. One potential way out is realizing that there is no need to treat the visualization as static. It is already possible to use a text-matching filter to modify the visualization by typing within the interface as described in Section 4.3. Perhaps, it would make sense to allow the user to filter the visualization by pointing and clicking as well.

Figure 4.4a only displays details of the hierarchy at the top level of the current view. It is clear that this alternative bubble tree leaves out information present in the other visual-



(a) Top level (b) Branch of the second level (c) Branch on the third level

Fig. 4.4 Interactive balloon tree visualization of hierarchy

izations, even though it is supposed to visualize the same data set as Figure 4.3. However, it displays all the adjacency relations between groups on single level of the hierarchy and displays the inclusion relation between the root of the hierarchy and the first level of the hierarchy. If the user clicks on gray circle in Figure 4.4a representing a branch of the hierarchy on the second level, then the interface can modify the visualization to Figure 4.4b. The user can then click on the gray circle in Figure 4.4b to proceed into the deepest level (which in Figure 4.3 is the third level) in the whole hierarchy shown in Figure 4.4c.

However, this interactive visualization has lost information about the adjacency relations between the current level of the hierarchy and a higher level of the hierarchy. It only displays adjacency relations between the groups on the same level. Therefore, even though Figure 4.4a shows the gray circle having adjacency relations between all the other groups and elements on the top level of the hierarchy, Figure 4.4b loses this information completely and there is no way to recover exactly what terminal elements are connected to each other.

Fortunately, a decision made in Section 4.3 allows this problem to be circumvented. The problem of lost information in this scenario is a result of all of the elements indiscriminately belonging to the same component. Since the decision is made in Vizmapper to separate the potential sources of a connection from the potential destinations of a connection and any given signal can only be an output (source) or input (destination), this problem is nonexistent.

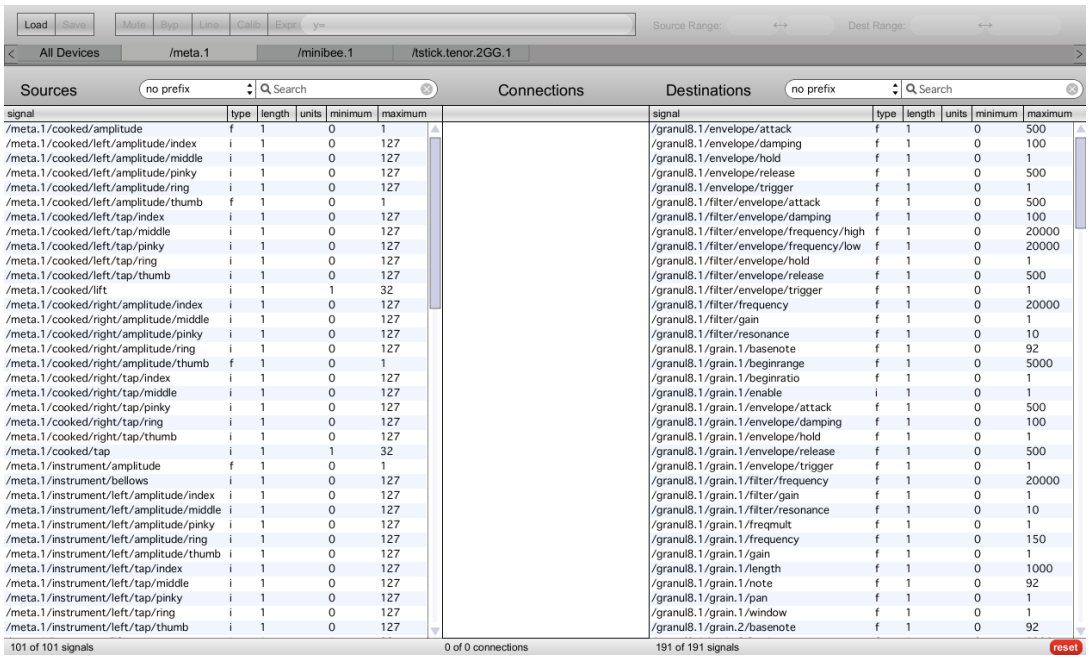


Fig. 4.5 Example Mapper network displayed in Maxmapper - Meta-Instrument and Granul8

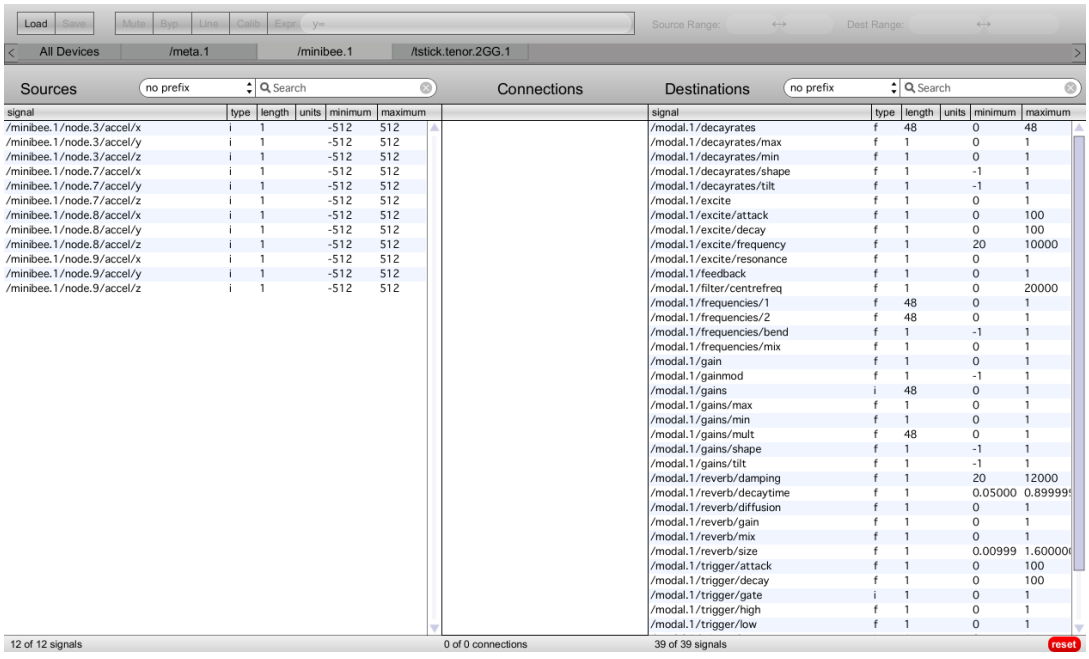


Fig. 4.6 Example Mapper network displayed in Maxmapper - Minibees and Modal

4.5 Vizmapper

The best way to see this is to see the visualization used in Vizmapper. The example Mapper network that will be used in the remainder of this chapter features Serge de Laubier's Meta-Instrument [40] and a collection of four wireless Minibee transmitters, each with a 3-axis accelerometer, as the control devices. The synthesizer devices are the Granul8 granular synthesizer and a modal synthesizer, both developed at IDMIL and implemented using Max/MSP.

Figure 4.5 shows Maxmapper displaying the Meta-Instrument output signals in the *Source* column and the Granul8 input signals in the *Destination* column. As shown at the bottom of the interface, the Meta-Instrument declares 101 output signals and the Granul8 synthesizer declares 191 input signals. Figure 4.6 shows the 4 Minibees with a total of 12 output signals (3 signals each) and the modal synthesizer with 39 signals. The idea is that Vizmapper presents a different view of this large system, which makes more clear the relationships between signals.

Figures 4.7 through 4.11 show screenshots of a Figure 4.4-style branch traversal down the Meta-Instrument in Vizmapper. Every highlighted cluster in these figures indicate a user click on the cluster to transition to the next screen in the interface. This series of screenshots shows the implementation of the interactive, balloon tree visualization arrived at in Figure 4.4.

The inclusion relations between all signals on the Mapper network are inferred by Vizmapper from the OSC namespace of the Mapper network. This places a premium on a wisely constructed OSC namespace for the signals that reduces the number of signals at any given level of a branch of the hierarchy. The OSC namespace of the Mapper-enabled Meta-Instrument is an example of a well-constructed namespace. As explained in Chapter 3, this type of visual hierarchical organization reduces the number of elements that a user has to decide between on any given level of a branch of the hierarchy and should mean a more effective interface. Notice the use of whether the circle is filled or not to represent whether the circle represents a terminal signal or a cluster of signals at an intermediate level of one branch of the complete hierarchy. *Cluster* and *branch* will be used interchangeably to refer to a specific position in the network hierarchy. Additionally, the Vizmapper visualization of the network displays two levels of the hierarchy at any given point, giving the user more context then the one level view used in Figure 4.4.

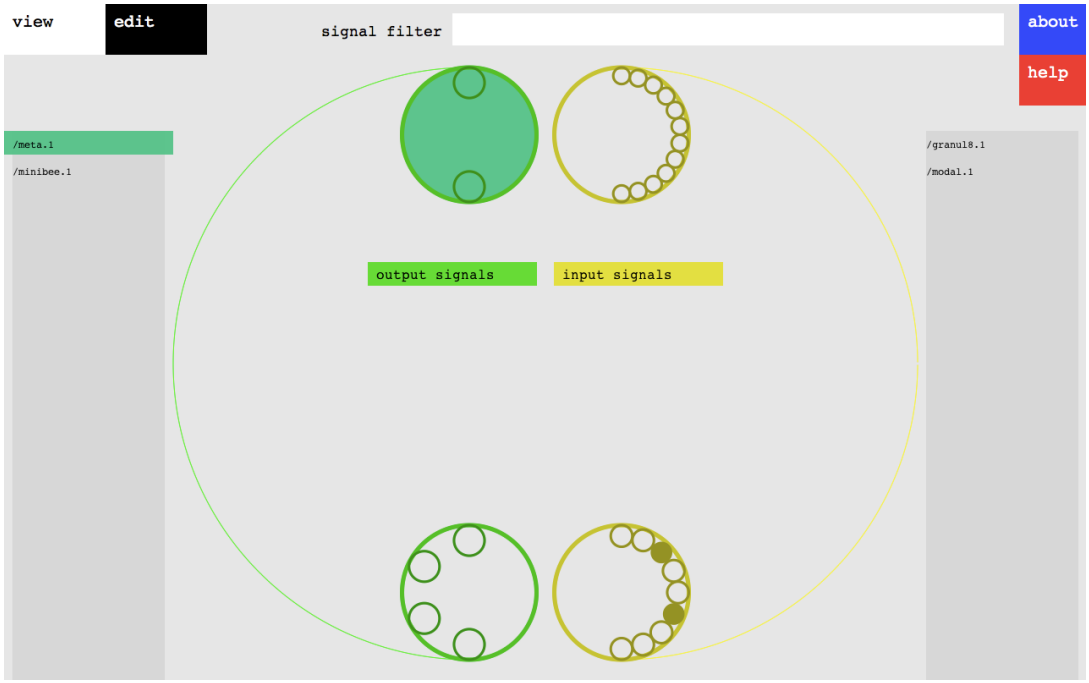


Fig. 4.7 Top level of the Mapper network signal hierarchy with the Meta-Instrument highlighted

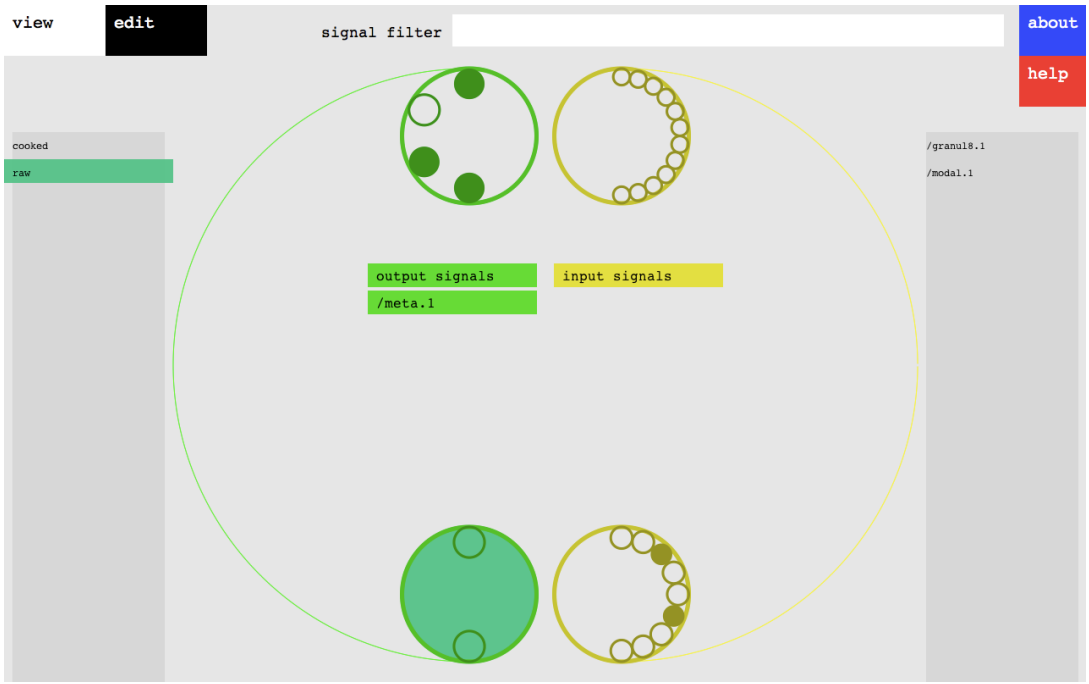


Fig. 4.8 Top level of the Meta-Instrument with the **raw** signal cluster highlighted

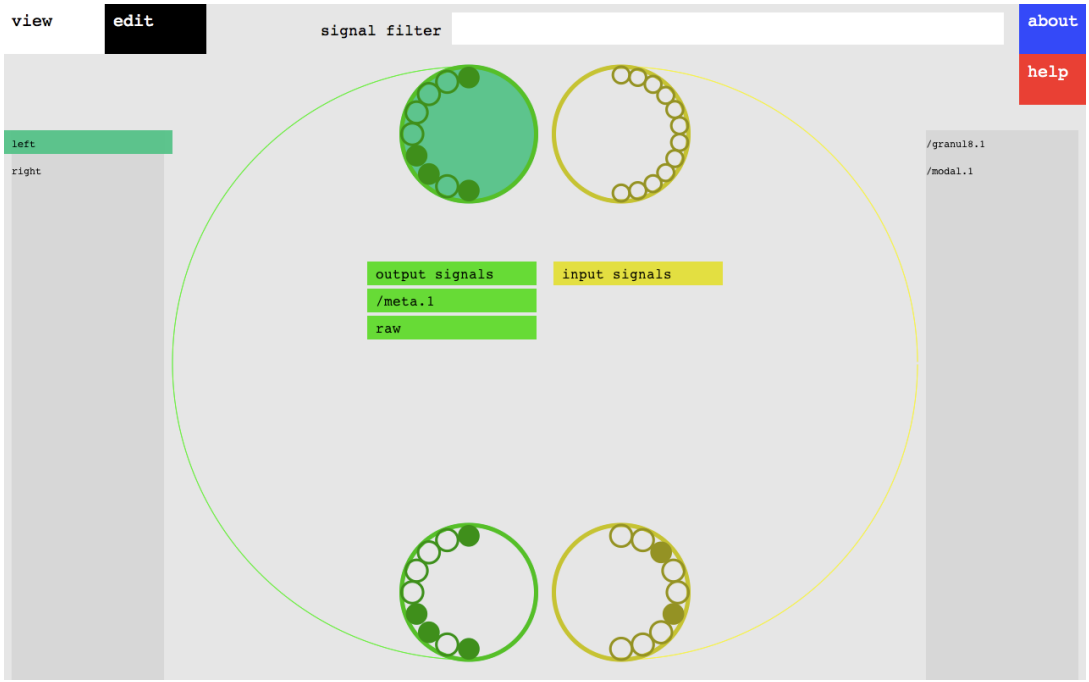


Fig. 4.9 **/meta.1/raw** branch with **left** signal cluster highlighted

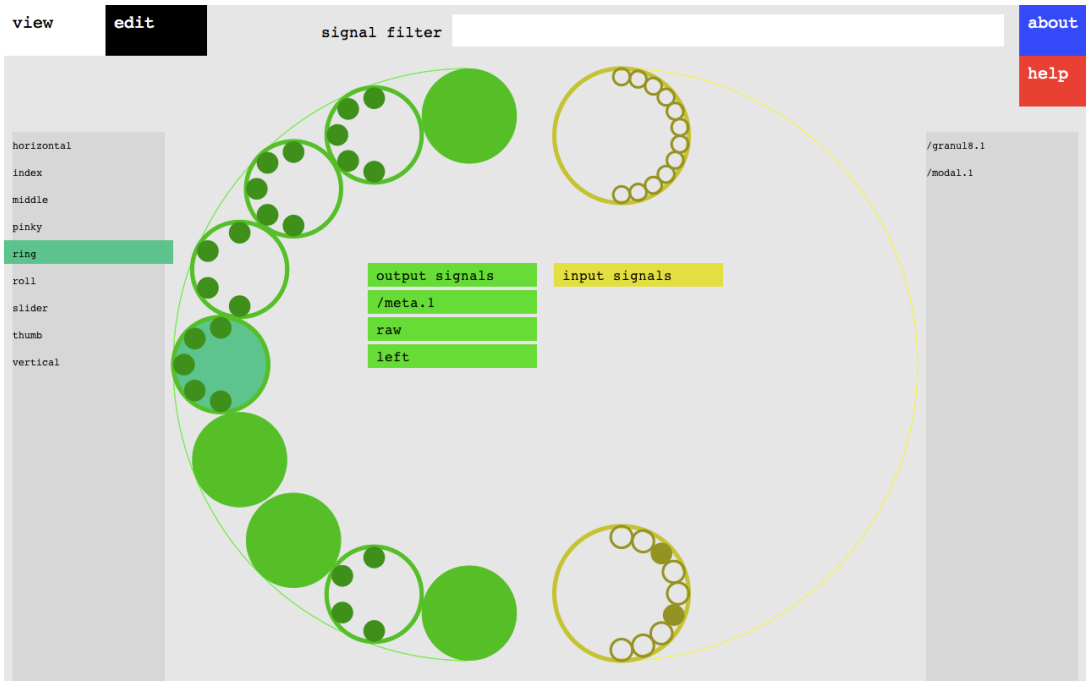


Fig. 4.10 **/meta.1/raw/left** branch with **ring** finger cluster highlighted

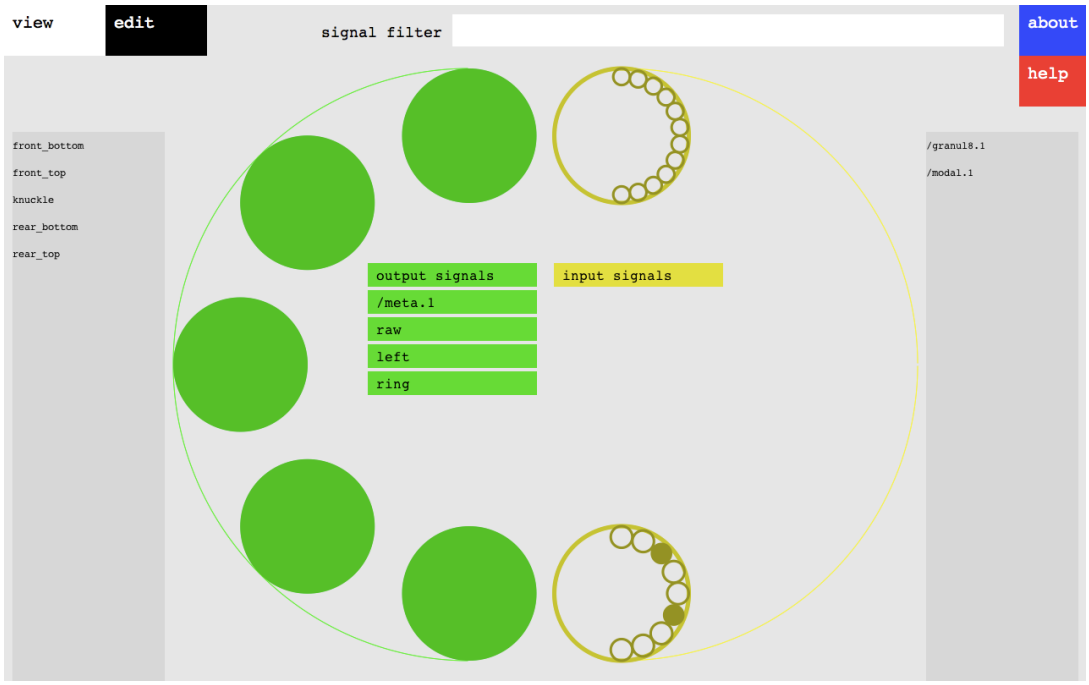


Fig. 4.11 **/meta.1/raw/left/ring** branch with 5 signals in this cluster displayed

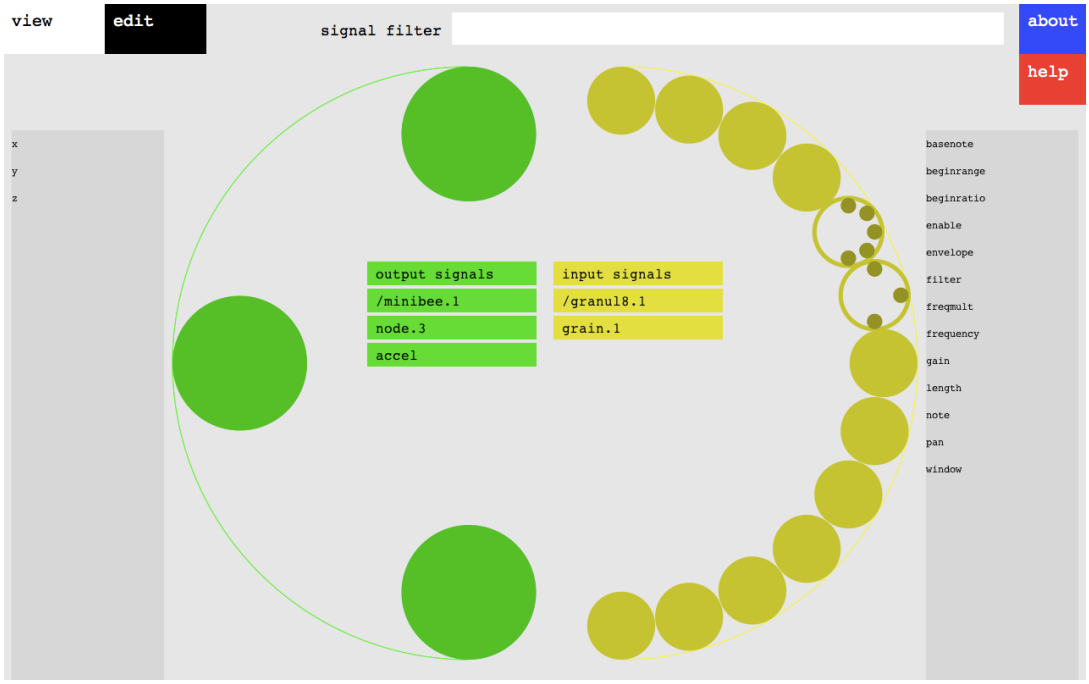


Fig. 4.12 **/minibee.1/node.3/accel** branch of output signals and **/granul8.1/grain.1** branch of input signals

The center of the screen displays two columns of buttons that display a trace of the path taken in the OSC namespace hierarchy for the output signals and for the input signals. Clicking on one of the buttons in a column returns the visualization of either the outputs signals or input signals to a previous position in the current traversal. Clicking on the always present *output signals* and *input signals* buttons returns the visualization of that set of signals to the first level of the hierarchy. The left and right columns of text display the names of the currently accessible groups and/or signals at the current level of the hierarchy. The hierarchical visualization and text-matching filter at the top of the screen work together by removing any groups or signals from the visualization whose paths do not match the text filter.

In order to make a connection, a user uses the *view mode* to navigate down to a level of both the output signal hierarchy and input signal hierarchy that have at least one terminal signal in each. This is because the current view of the network is frozen when the user switches to the *edit mode* by clicking the edit tab at the top left of the screen. Hierarchy traversal is limited to the view mode. Figure 4.12 shows an example traversal in view mode to prepare for connections between output signals in the `/minibee.1/node.3/accel` cluster and input signals in the `/granul8.1/grain.1` cluster.

Once the two signals that are to be connected are in view, the user switches to the *edit mode*, clicks on one output signal and one input signal, enters the pertinent information into the form on the left side of the screen to define a functional transformation (if necessary), and then clicks the *update* button to tell the Webmapper server to request a connection between the two selected signals. Any device linking that is necessary to create the needed Libmapper routers to manage the connection is done automatically without user intervention.

As of this thesis, Libmapper allows network monitors to specify a functional transformation with the several properties. Vizmapper provides an interface to a subset of these properties to simplify the choices of the user:

Mode - There are 5 modes that can be selected.

None - This acts as a default in Vizmapper. If the min and max range of the output and input signals are defined in the signal declaration then Libmapper will request a line transformation with the expression of the line chosen to cover both ranges linearly. Otherwise, it will request a bypass transformation.

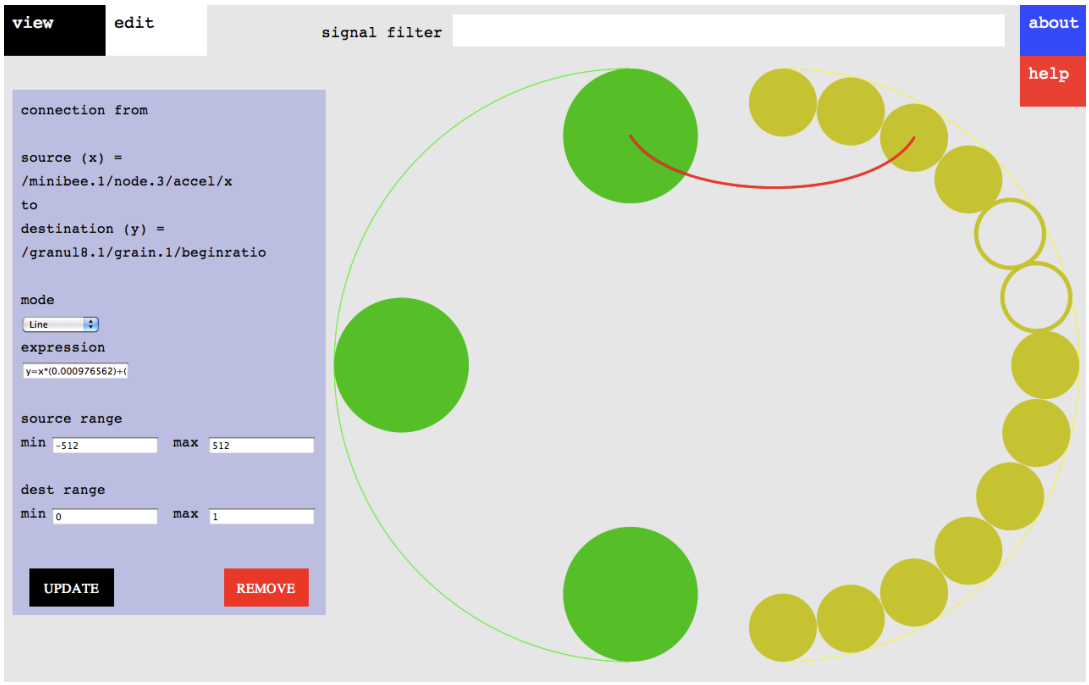


Fig. 4.13 Creation of new connection between **/minibee.1/node.3/accel/x** signal and **/granul8.1/grain.1/beginratio** signal in edit mode

Line - This mode applies a linear transformation defined by the specified expression, source range, and destination range.

Bypass - This mode represents a lack of functional transformation and applies no conditioning to the output signal before sending the data to the input signal.

Expression - This mode accepts an arbitrary single-variable expression to condition the signal.

Calibrate - If the input signal has a defined range, this mode monitors the data over the connection until a line or expression mode request is sent and constructs a transformation calibrated to the data transmitted over the connection during this time. If the destination signal has no defined range it defaults to a bypass.

Expression - The expression used in a line or expression transformation.

Source range - The range that the output signal data is restricted to.

Destination range - The range that the input signal data is restricted to.

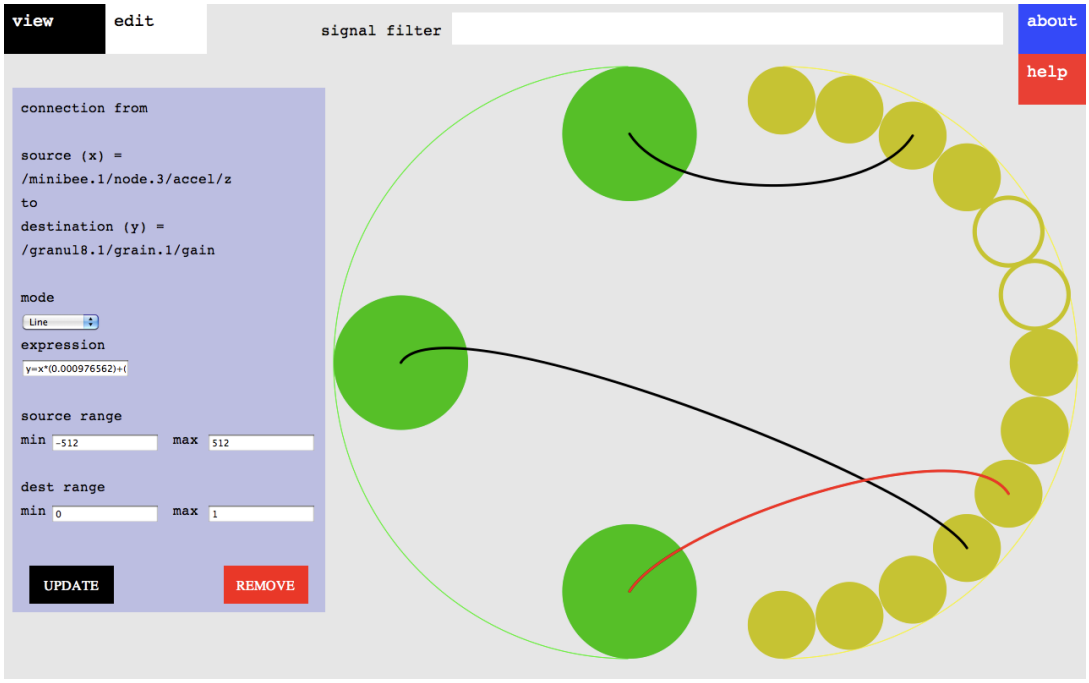


Fig. 4.14 Creation between two more pairs of signals in same branch as first connection

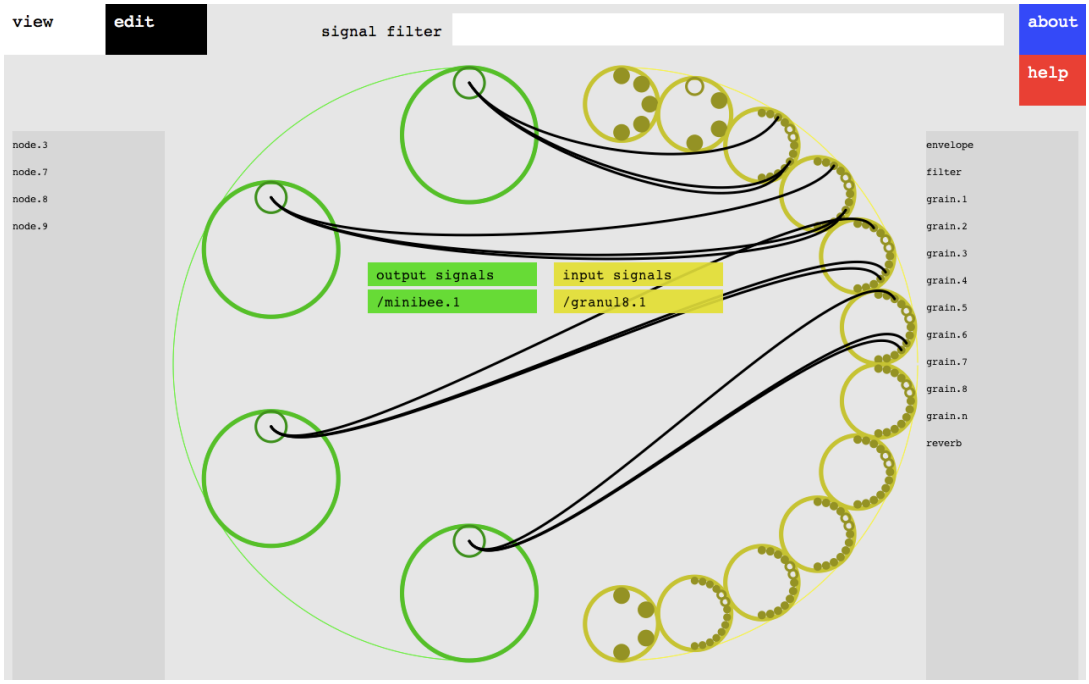


Fig. 4.15 View mode, showing additional connections between the remaining 3 Minibees and the next 3 available granular synth grains

The user can view the functional transformation properties of a connection at any time by clicking on the curved line that represents the connection while in edit mode. Clicking on a connection while in view mode does nothing.

To break a connection between two signals or view the active functional transformation over the a connection, the user clicks on a connection between two signals to select the connection. To finish breaking the connection, the user clicks the *remove* button.

Figure 4.13 shows the result of creating a default *None* connection between the `x` output signal in the `/minibee.1/node.3/accel` cluster and the `beginratio` input signal in the `/granul8.1/grain.1` cluster. The mode, expression, source range, and destination range have been automatically calculated by Libmapper to fit the ranges of the signals that were connected. After making two more connections between the `y` and `length` signals, and the `z` and `gain` signals, the interface looks like Figure 4.14.

The same 3 signal connections are made for each of 3 more pairs of clusters: `/minibee.1/node.7` and `/granul8.1/grain.2`, `/minibee.1/node.8` and `/granul8.1/grain.3`, `/minibee.1/node.9` and `/granul8.1/grain.4`. If the user navigates to the top level of `/minibee.1` and `/granul8.1` in view mode, the resulting display looks like Figure 4.15.

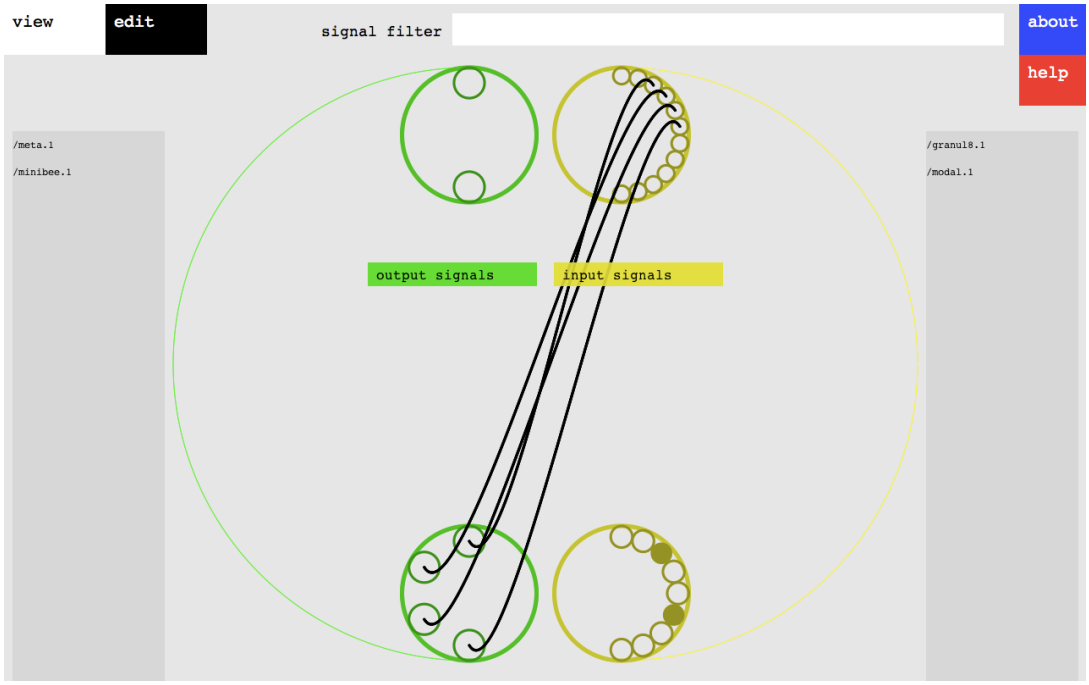


Fig. 4.16 Top level view of Mapper network signal hierarchy with the 12 new connections summarized as 4 visual connections

Anecdotally, after working with this particular Mapper network for some time, the visual patterns representing clusters of signals begin to be recognizable and it is possible to navigate through the network without paying too much attention to the textual labels of clusters and signals while in view mode. This is not the case when two clusters contain a similar arrangement of subclusters and signals, which is one shortcoming of this visualization. In future work, it may be fruitful to explore the idea of automatically generating a distinct visual pattern for each cluster and signal that guarantees elements in the same branch of the hierarchy are readily distinguishable from each other.

At the top level of the whole Mapper network, the amount of visual clutter is reduced because the 12 connections that were created are reduced to 4 visual connections (one curved line per Minibee node/Granul8 synth grain pair) as shown in Figure 4.16. Figure 4.17 shows the result of reducing the visual clutter further by using the signal filter functionality of the text input box at the top of the interface. This filtering mechanism can be used at any point in either the view or edit mode.

Comparing these screenshots of the Vizmapper with the Maxmapper display of the same Mapper network as shown in Figure 4.18, there is a clear difference in the structural knowledge of the network that each interface provides. A benefit of the Libmapper system is that it allows multiple monitors using different GUIs to operate on the same Mapper network at the same time. Therefore, different team members can use different interfaces depending on their personal preferences.

4.6 Summary

This chapter explained how the Vizmapper interface provides all the functionality necessary to create a mapping on a Mapper network. The central argument is that through the process of understanding the mapping task and researching user interface design principles, data visualization principles, and human cognitive processes, this thesis research has resulted in an effective interface for configuring mappings that scales to a MSN with many signals. The proof of this will be in the future use of the tool, as no user testing has been conducted, however past research into graphical user interfaces suggests that the decisions made in Vizmapper are significant steps in the right direction.

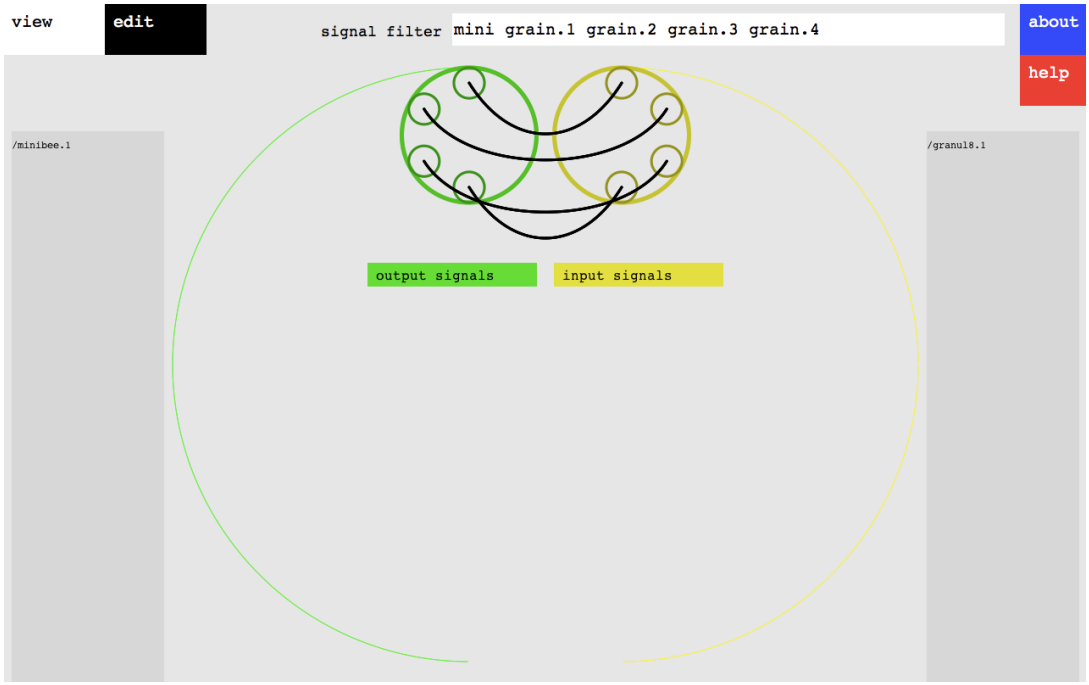


Fig. 4.17 Use of signal filter in top level view to focus on relevant signal clusters

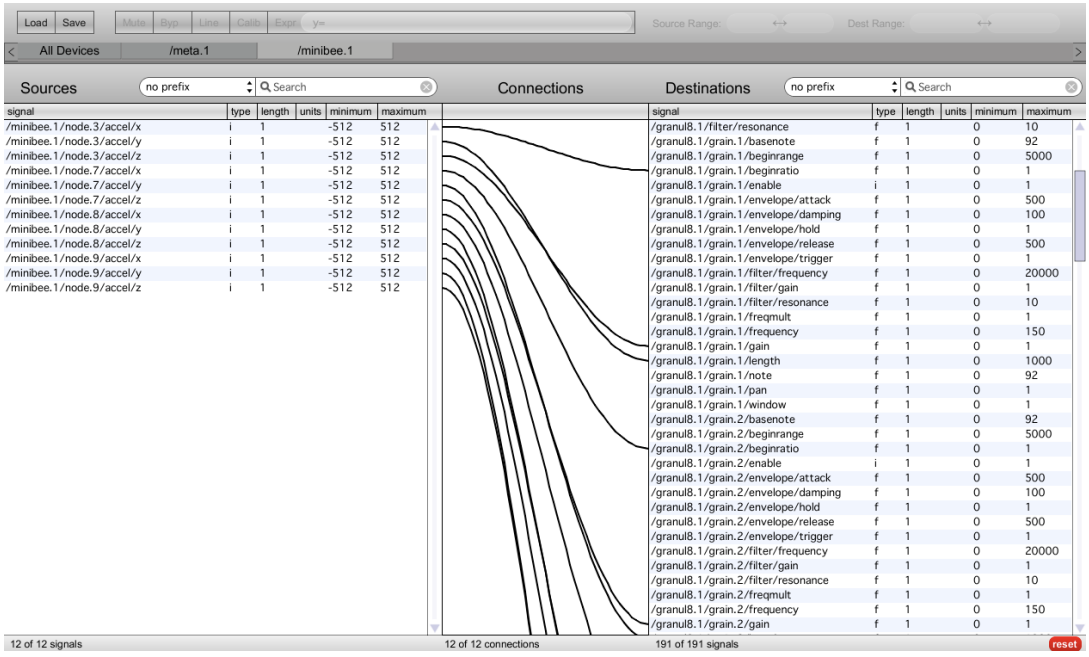


Fig. 4.18 Same connections shown in Maxmapper

Chapter 5

Conclusions and Further Research

This chapter concludes the research presented in this thesis. For a non-technical user, understanding the concept of mapping enough to develop an informed opinion about how to configure the mapping of a particular musical system can be a daunting task. However, it is the responsibility of a computer interface to bridge this gap and turn a computer into an intuitive mechanism for accomplishing specific tasks. The task of designing a mapping for an MSN is bounded and specific enough that proper use of graphics and interactivity can reduce the complexity of configuring a mapping for a large Mapper network. Vizmapper is the result of research into what HCI and cognitive science have to say about how a graphical user interface can accomplish this goal.

The base impulse behind this whole endeavor is the belief that the interface of a system is as important, arguably more important, then the internals of the system. The internals of a system are tasked with maintaining the stability, reliability, and functionality of the system, but the interface of the system to the broader set of external systems, users, and world is ultimately what gives a system purpose. The history of computer science and electrical engineering is largely a history of ignoring the interface or relegating the interface to a mere curiosity, simply the interest of the types of individuals who are too "artsy" to appreciate "real" computer science and programming languages. This is true of not only graphical user interfaces, but the interfaces of server-based application programming interfaces, code libraries, object oriented classes, and even the documentation (I admit to this shortcoming) that allows the external world to understand the "protocol" of the interface in plain, human language. The distinction between the *system* and the *interface*

is, at its heart, an artifact of the history of computers and the specific individuals who devoted their lives to the endeavor.

Noticeably absent in this thesis is any mention of user testing or experimentation to see whether the Vizmapper interface actually accomplishes what is intended. Such testing would certainly be expected of a larger project on this topic and would be helpful to refine the interface in future iterations. However, to steal a phrase, I greet the conventional understanding of the utility of user testing with skepticism. Sometimes an interface should be molded to the user and other times the user should be molded by the interface. That is the nature of compelling non-computational interfaces; it stands to reason that the same would be true of computational interfaces. The type of long term study that would be necessary to observe how interfaces affect the mindset of a user is not something that is often (if at all) performed. However, I think that such a study would be far more compelling and useful than the typical user testing that is performed.

There are two concrete improvements that should be made to the current Vizmapper interface in the future. The first is to use Holten's connection bundling algorithm [1] to reduce visual clutter when many connections are displayed simultaneously. The second is to use some form of visual hashing to automatically generate a distinct visual pattern for each cluster and signal so that visually identical signals and clusters can be readily distinguished between, as one grows familiar with their particular Mapper network. There is much work to be done to further improve interfaces to Libmapper if the Mapper Tools project is ever to be used in a wide context.

The tools and code developed as part of the Mapper Tools project, including Vizmapper, are freely available on the web. See the appendix for information about where to find these tools.

Appendix A

Resources

Mapping Tools project http://www.idmil.org/projects/mappingtools

Libmapper http://www.idmil.org/projects/libmapper

Webmapper http://github.com/radarsat1/webmapper

Vizmapper http://github.com/vijayrudraraju/vizmapper

Appendix B

Definitions

There are a number of terms that are used in chapter 4 that are helpful to define. The specifics of the definitions are debatable and are not cited because they are based on experience and may not even be terms that are widely used, however they remain helpful for understanding the rest of this chapter. Treat them as placeholders for their definitions.

Development environment a sofware/hardware environment that the developer uses to write programmer code and executables

Deployment environment a software/hardware environment that the user uses to run executables

Operating system a program that manages hardware resources and the computer processor - from the developer's point of view, reduces the complexity and risk of writting programmer code that interfaces with the physical hardware and computer processor by providing an easier to use set of operations that provide a higher level interface (called system calls) to the processor and other hardware - system calls prevent programmer code from generating machine code that executes low-level processor and hardware operations in such a way that would destablize the system

Progammer code human readable text that is written by the developer in a specific computer language and translated to produce an executable

Machine code machine readable code that is often produced by a compiler and requires no further translation to be understood by a physical computer processor

Executable a file or collection of files that the user can *run* without any further manual translation within their deployment environment - can be programmer code, machine code, or other forms of executable code

Machine a physical computer processor that provides a stable interface of machine operations that can be executed by machine code

Compiler a software program that takes programmer code as input and produces machine code or machine code executables as output

Interpreter a software program that takes programmer code as an executable to run a program

References

[1] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 741–748, September 2006.

[2] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics*, vol. 5, pp. 110–141, April 1986.

[3] E. R. Tufte, *The Visual Display of Quantitative Information*. Cheshire, Connecticut: Graphics Press, 2 ed., 2001.

[4] S. Ferguson and M. M. Wanderley, "The mcgill digital orchestra: An interdisciplinary project on digital musical instruments," *Journal of Interdisciplinary Music Studies*, vol. 4, pp. 17–35, Fall 2010.

[5] J. Malloch, S. Sinclair, and M. M. Wanderley, "From controller to sound: Tools for collaborative development of digital musical instruments," in *Proceedings of the International Computer Music Conference*, (Copenhagen, Denmark), ICMA, August 2007.

[6] A. Hunt, M. M. Wanderley, and M. Paradis, "The importance of parameter mapping in electronic instrument design," *Journal of New Music Research*, vol. 32, pp. 429–440, December 2003.

[7] P. V. O'Neil, *Beginning Partial Differential Equations*. Wiley-Interscience, 1999.

[8] M. Beck and R. Geoghegan, *The Art of Proof: Basic Training for Deeper Mathematics*. Springer, 2010.

[9] K. Mehlhorn and P. Sanders, *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008.

[10] J. Rumbaugh, "Relations as semantic constructs in an object-oriented language," in *Conference proceedings on Object-oriented programming systems, languages and applications*, OOPSLA '87, (New York, NY, USA), pp. 466–481, ACM, 1987.

[11] D. V. Nort, *Modular and Adaptive Control of Sound Processing*. PhD thesis, McGill University, Montreal, QC, Canada, January 2010.

[12] C. Larman and V. R. Basili, "Interative and incremental development: A brief history," *IEEE Computer*, vol. 36, pp. 47–56, June 2003.

[13] J. Malloch, S. Sinclair, and M. M. Wanderley, "A network-based framework for collaborative development and performance of digital musical instruments," in *Computer Music Modeling and Retrieval. Sense of Sounds* (R. Kronland-Martinet, S. Ystad, and K. Jensen, eds.), vol. 4969 of *Lecture Notes in Computer Science*, pp. 401–425, Springer Berlin / Heidelberg, 2009.

[14] M. Wright, "The open sound control 1.0 specification." `http://opensoundcontrol.org/spec-1_0`, March 2002.

[15] A. Freed and A. Schmeder, "Features and future of open sound control version 1.1 for nime," in *NIME*, 04/06/2009 2009.

[16] J. Malloch and M. M. Wanderley, "The t-stick: From musical interface to musical instrument," in *Proc. of the 2007 Conf. on New Interfaces for Musical Expression*, pp. 66–69, 2007.

[17] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, August 2008.

[18] A. J. Dix, J. E. Finlay, G. D. Abowd, and R. Beale, *Human-Computer Interaction*. Prentice Hall Europe, 1998.

[19] J. Grudin, "The computer reaches out: The historical continuity of interface design," in *CHI'90 Human Factors in Computing Systems*, pp. 261–268, April 1990.

[20] P. H. Nielson, "History of operating systems." `http://www.philscis.com/CIS1520/LectureonHistoryofOperatingSystems.pdf`, 2011.

[21] S. Haldar and A. A. Aravind, *Operating Systems*. Pearson Education India, 2010.

[22] R. Bjork, "A brief history of computer operating systems." `http://www.math-cs.gordon.edu/courses/cs322/lectures/history.html`, 2000.

[23] E. S. Raymond and R. W. Landley, "The art of unix usability." `http://www.catb.org/~esr/writings/taouu/html/index.html`, January 2008.

[24] J. Johnson, T. Roberts, W. Verplank, D. Smith, C. Irby, M. Beard, and K. Mackey, "The xerox star: A retrospective," *Computer*, vol. 22, pp. 11–26, September 1989.

[25] A. Crystal and B. Ellington, "Task analysis and human-computer interaction: approaches, techniques, and levels of analysis," in *Proceedings of the Tenth Americas Conference on Information Systems*, (New York, New York), Americas Conference on Information Systems, August 2004.

[26] M. M. Wanderley and N. Orio, "Evaluation of input devices for musical expression: Borrowing tools from hci," *Computer Music Journal*, vol. 26, no. 3, pp. 62–76, 2002.

[27] J. Bertin, *Semiology of Graphics*. The University of Wisconsin Press, 1983.

[28] S. K. Card and J. Mackinlay, "The structure of the information visualization design space," in *IEEE Symposium on Information Visualization*, (Los Alamitos, CA, USA), p. 92, IEEE Computer Society, 1997.

[29] R. M. Shiffrin and W. Schneider, "Controlled and automatic human information processing: Ii. perceptual learning, automatic attending, and a general theory," *Psychological Review*, vol. 84, pp. 127–190, March 1977.

[30] W. S. Cleveland and R. McGill, "Graphical perception: Theory, experimentation, and application to the development of graphical methods," *Journal of the American Statistical Association*, vol. 79, pp. 531–554, September 1984.

[31] W. Peng, M. O. Ward, and E. A. Rundensteiner, "Clutter reduction in multidimensional data visualization using dimension reordering," in *IEEE Symposium on Information Visualization*, pp. 89–96, 2004.

[32] J. Johnson, *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules*. Morgan Kaufmann, May 2010.

[33] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *The Psychological Review*, vol. 63, pp. 81–97, 1956.

[34] T. O. Nelson and E. E. Smith, "Acquisition and forgetting of hierarchically organized information in long-term memory," *Journal of Experimental Psychology*, vol. 95, pp. 388–396, October 1972.

[35] P. M. Wortman and L. D. Greenberg, "Coding, recoding, and decoding of hierarchical information in long-term memory," *Journal of Verbal Learning and Verbal Behavior*, vol. 10, pp. 234–243, June 1971.

[36] C. Ahlberg and B. Shneiderman, "Visual information seeking: tight coupling of dynamic query filters with starfield displays," in *Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence*, CHI '94, (New York, NY, USA), pp. 313–317, ACM, 1994.

[37] A. Tanaka, "Musical performance practice on sensor-based instruments," in *Trends in Gestural Control of Music* (M. M. Wanderley and M. Battier, eds.), pp. 389–406, IRCAM, 2000.

[38] J. Annett and K. Duncan, "Task analysis and training design," *Occupational Psychology*, vol. 41, pp. 211–221, 1967.

[39] M. M. Wanderley and P. Depalle, "Gestural control of sound synthesis," *Proceedings of the IEEE*, vol. 92, pp. 632–644, April 2004.

[40] S. de Laubier, "The meta-instrument," *Computer Music Journal*, vol. 22, pp. 25–29, Spring 1998.