A Flexible Tool for the Visualization and Manipulation of Musical Mapping Networks

Aaron Henry Krajeski

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

July 2013

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

2013/07/24

Abstract

This report describes the use of LaTeX to format a thesis. A number of topics are covered: content and organization of the thesis, LaTeX macros for controlling the thesis layout, formatting mathematical expressions, generating bibliographic references, importing figures and graphs, generating graphs in MATLAB, and formatting tables. The LaTeX macros used to format a thesis (and this document) are described.

Acknowledgments

Acknowledge this, asshole.

Preface

There are some things I should probably pre-face, certainly not reface.

Contents

List of Figures

various graphs of response time (discussion) screenshot of drawing screenshot of saving/loading screenshot of main view screenshot of grid view

List of Tables

List of Acronyms

DMI	Digital Musical Instrument
GUI	Graphical User Interface
IDMIL	Input Devices for Musical Interaction Laboratory
API	Application Programming Interface
SWIG	Simplified Wrapper and Interface Generator
OSC	Open Sound Control
MVC	Model View Controller
HTTP	HypterText Transfer Protocol
HTML	HyperText Markup Language
CSS	Cascading Style Sheet

Chapter 1

Introduction & Motivation

> "In order that our tools, and their uses, develop effectively: (A) we shall have to give still more attention to doing the approximately right, rather than the exactly wrong..." (Tuckey 1965)

Throughout the vast majority of human history the term "musical instrument" has signified both the physical object with which the musician interacted *and* the direct source of the sound created: a violin with vibrating strings, a reeded saxophone, a timpani with its membrane, etc. With the advent of electronic sound in the late 19th century, it became possible for interactive objects to be separated from the sound producing devices they control (Chadabe 2000). As technological development progressed, so did the capacity to divide musical instruments into independent parts. With digitization it is now not only possible to arbitrarily connect a control element to any sound synthesis dimension, but also to modify this association according to the whims of the user. Since mechanical linkages are no longer necessary in the design of musical instruments, control surfaces can, and often do, take on a variety of wild and arbitrary shapes and modes of interaction.[1] All that is necessary for this process is for control devices to output some kind of electronic signal that other, sound-producing instruments can accept. With no obvious means of implementation, the success or failure of these new digital musical instruments (DMIs) often depends on how artfully their output signals are "mapped" to synthesis parameters.

More and more frequently, the mapping itself becomes a part of the expressive element

[1] International Conference on New Interfaces for Musical Expression. [Online]. Available: `http://www.nime.org/`. Accessed June 23, 2013.

of a musical work (Hunt and Kirk 2000), as it associates itself with both composition and performance with certain DMIs. Thus is becomes necessary for mapping to be dynamic and interactive: sometimes poured over in composition studios, or sometimes edited mid-piece. Musicians are not necessarily computer programmers, thus ideally the act of mapping should not require computer expertise. This means that on top of the low-level layer of interactive mapping (simply instructing a machine to connect signals to others in specific ways), there needs to exist an interface to make such an activity easy, logical, intuitive and in line with the artistic process.

As the actual act of mapping is as expansive and nebulous as the instruments it hopes to assist, thus the design of such a mapping interface presents many interesting challenges. Due to the tremendously wide variety of possible use cases, several seemingly contradictory goals emerge: What is the best way to visually represent complex musical networks while simultaneously allowing for the user to easily manipulate them? How can systems with many devices and signals be well represented while still allowing in-depth control of small networks? How can an interface be transparent to non-technical users while still accommodating all possible functionality that advanced users may wish to use?

1.1 Context and Motivation

The world of digital musical instruments is still dominated by keyboard type input devices. Though many novel DMIs currently exist (and many more are being created) these devices are usually unique and often difficult to use without their creator being present (Cook 2009). Since mapping is such an important feature of DMIs, a means of transparently editing them could inspire more people to use novel musical controllers. In response to this challenge, libmapper, a tool for collaborative mapping, was created at the Input Devices and Music Interaction Laboratory (IDMIL).

In its most basic state, libmapper takes the form of an application programming interface (API). APIs are primarily a means for different pieces of computer software to communicate with one another. The only possible way to communicate directly with the libmapper API is through coded text. For example, figure 1.1 causes a synthesizer to announce itself and begin communicating with other devices on a libmapper-enabled network (Malloch et al. 2008).

This is obviously inaccessible to users who do not have the time or desire to read

```c
#include <mapper.h>
mapper_admin_init();
my_admin = mapper_admin_new("tester", MAPPER_DEVICE_SYNTH, 8000);
mapper_admin_input_add(my_admin, "/test/input","i"))
mapper_admin_input_add(my_admin, "/test/another_input","f"))

// Loop until port and identifier ordinal are allocated.
while ( !my_admin->port.locked || !my_admin->ordinal.locked )
{
   usleep(10000); // wait 10 ms
   mapper_admin_poll(my_admin);
}

for (;;)
{
   usleep(10000);
   mapper_admin_poll(my_admin);
}
```

Fig. 1.1 A sample of libmapper code

through documentation files, or those who have no knowledge of programming semantics. A steep learning curve is especially a problem for a network tool like libmapper: because it is primarily a means of communication between instruments, it can only be successful if it is widely adopted. A libmapper-enabled controller will only be useful if many high quality libmapper synthesizers exist. In turn, synthesizer makers will only have incentive to incorporate libmapper into their designs if there are already controllers that use the system.

An API can be contrasted with a graphical user interface (GUI), an interface that contains abstractions on top of the raw code. These abstractions can be features like buttons, menus, visual representations of data, etc. In general, GUIs are designed to be familiar to those who have used digital devices in the past, and thus easy to learn and use. Two GUIs have been created for libmapper (see section 2.4.4): a basic interface built in Max/MSP[2] and vizmapper (Rudraraju 2011), a more abstract representation of a libmapper network. Both of these GUIs have their strengths, yet neither adequately meets

[2]MAX: You make the machine that makes your music. [Online]. Available: `http://cycling74.com/`. Accessed June 17, 2013

the full range of possible use cases for libmapper. A more flexible approach is required if the GUI is to be usable in situations with hundreds of signals, transparent for systems with multi-leveled hierarchical devices, intuitive during performances where devices output light and haptic feedback as well as sound, and responsive for tasks where speed of manipulation is an absolutely necessity.

With such an interface in place, libmapper can greatly expand its user base. As a result, more controller and synthesizer designers may choose to incorporate libmapper into their devices, and in turn these devices will be easier to learn and use. Hopefully the end result will be greater adoption of non keyboard-based DMIs in the electronic music community.

1.2 Project Overview

The focus of this project is to create a graphical user interface for libmapper. This interface aims to be flexible and intuitive, simultaneously allowing for useful control of the full range of possible libmapper networks while also not intimidating non-technical users with complexity. The presupposed solution to this problem is to provide users with multiple independent modes of viewing and interacting with the network. Certain view modes can excel in providing precise control, while others can help users understand the structure of complex networks. The idea is to provide multiple imperfect solutions to an unsolvable problem, so that each can be "...approximately right, rather than exactly wrong" (Tuckey).

This project was structured in four major, non-sequential parts: a review of prior visualized mapping interfaces, the integration of presently available GUIs for libmapper, the extension of interface features and the collection of user feedback. Results of the research phase informed implementation and are presented here. Development began by updating a cross-platform implementation of the current Max/MSP-based GUI, while integrating functionality from vizmapper. New view modes were integrated into design while refining functionality of the previous ones. Throughout the design process, the GUI was provided to potential users who gave feedback on the strengths, weaknesses and potential avenues for improvement.

1.3 Thesis Overview

The remainder of this document is organized as follows. Chapter 2 outlines concepts necessary for providing context for this thesis project. A wide variety of domains inform the creation of a musical mapping interface. Special attention is paid to mapping theory, data visualization, relevant existing user interfaces, user centric design techniques and specifics of libmapper itself. Chapter 3 describes the design and implementation of the libmapperGUI. This chapter presents design decisions made, technical details of implementation and how the user-centric approach informed the process. Chapter 4 evaluates results, both on the empirical level of software performance as well as qualitative user feedback. Finally, Chapter 5 presents conclusions of the work and suggests further developments for the software.

1.4 Contributions

The contributions of this thesis are: the exploration of issues related to user interface design for musical mapping networks, the design and implementation of an interface for libmapper that aims to improve on usability and flexibility of the system, and this thesis document, which describes the research and development therein.

Chapter 2

Background

Dynamic mapping is becoming an increasingly important requirement for digital musical instruments. This chapter surveys currently available tools that allow for manipulation of musical and non-musical networks in real time. The first section presents a review of mapping itself, both from a theoretical and a musical standpoint. This portion also introduces the libmapper application programming interface. The second section reviews relevant work in the visual representation of information. The following portion describes applicable techniques in user interface design. Finally, a review of user interfaces for mapping is presented.

2.1 Mapping

At the most fundamental level, *mapping* is the act of associating two or more sets of information. Mappings can be mathematical, computational, linguistic (like translation), geographic, or even poetic[1]. Within the context of DMI design mapping is the relationship between sensor outputs and synthesis inputs. The entire character of a new instrument can be drastically altered though mapping, even while control surface and sound source are held constant (Hunt et al. 2003). As a result, the theoretical formalism of mapping becomes yet another necessary tool in the modern instrument designer's arsenal.

[1] What is metaphor if not the association of unlike things?

2.1.1 Mapping theory

Mapping as function and mapping cardinality

From the perspective of mathematics, the term *mapping* is very nearly synonymous with *function* (Halmos 1970), as both describe how one set of numbers corresponds with another. The first group is commonly referred to as the *domain* and the second as the *codomain* or *range*. An in-depth review of functions in mathematics is beyond the scope of this thesis, however a few fundamental examples will be useful for reference in section 2.1.2. The following are instances of two basic types of mathematical functions:

$$y = 2x - 1 \tag{2.1}$$

$$y = x^2 \tag{2.2}$$

Each function takes a single input value (x) and *maps* that number onto its range (y). The fact that each of these equations take in only a single number as input, and output a single number in turn, means they can be graphed in a two dimensional space. This is not necessarily the case, as functions can input and output lists of numbers (vectors). Mathematically they are not very interesting, but they represent two fundamentally different *kinds* of functions.

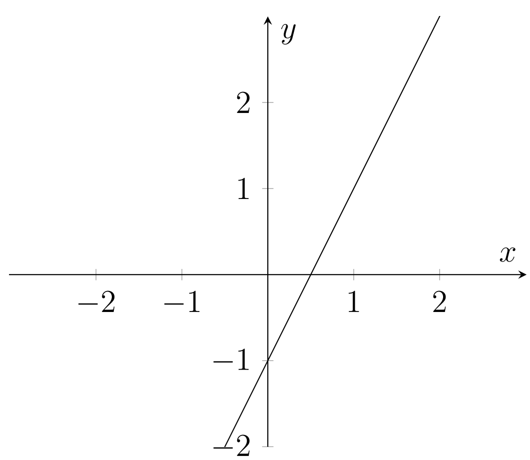


Fig. 2.1 The function described in equation 2.1, graphed in two dimensions.

For equation 2.1 each input value has *one and only one* corresponding output value. The same is true if the function is to be inverted, as each output value corresponds to only one input value. The range is simply a scaled and shifted version of the domain. The mapping's *one-to-one* nature can clearly be seen in figure 2.1. To mathematicians this is known as the mapping's *cardinality*.

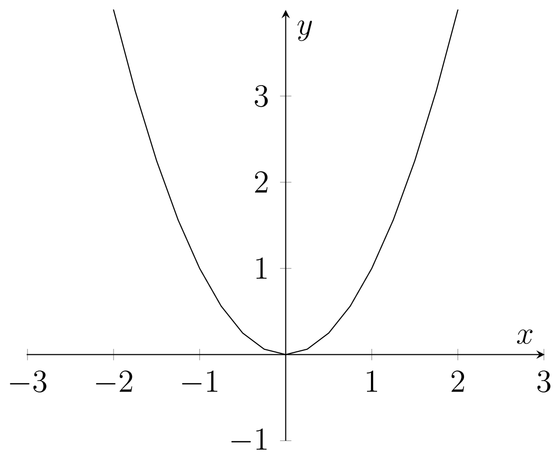


Fig. 2.2 Equation 2.2 projected on the Cartesian plane.

This is not the case for equation 2.2, for although each input has only one output, single positions in the codomain can have multiple corresponding inputs (e.g. both 3^2 *and* -3^2 are equal to 9). Thus we can consider equation 2.2 to be a classical example of a mapping with a cardinality *many-to-one*. In figure 2.2 the range of the function is wrapped back onto itself such that a horizontal line could intersect the curve twice.

Two more mapping cardinalities are relevant to instrument design, an example of each:

$$y = \pm\sqrt{x} \tag{2.3}$$

$$y = \pm\sqrt{1 - x^2} \tag{2.4}$$

They are not considered to be functions by mathematicians[2], but are nonetheless important for our purposes. In equation 2.3 a single input can result in multiple outputs (an input of 4 results in the output of *both* 2 and -2), yet each output has only a single input. This is simply the inverse function of equation 2.2, and is an example of a *one-to-many* mapping. On a graph of such a mapping, a *vertical* line may cross at multiple points. The final equation is that of a circle centered at the origin with a radius of one. This is a

[2]In mathematics, a true function can have no more than one output value for every input value.

many-to-many mapping, as both it an its inverse result in multiple outputs from a single input.

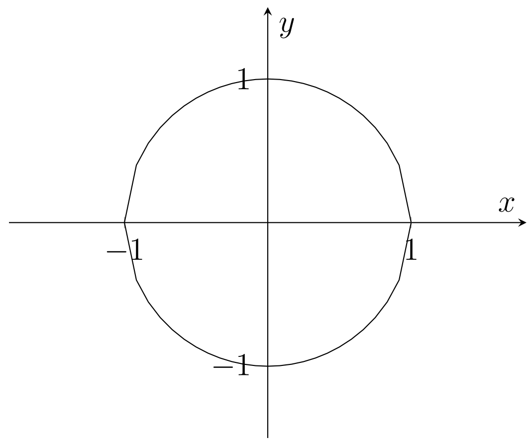


Fig. 2.3 Equation 2.4, a many-to-many mapping.

Though a graphical plane is the most common way for mathematicians to visualize two-dimensional functions, drawing the direct association between input and output will be more useful going forward. Figure 2.4 provides an illustration of such an approach. The astute reader will notice a striking similarity to the GUI view mode described in section 3.2.2 and these diagrams.

Mapping as association

In computer science, a mapping is less commonly referred to as a function and more usually called an *associative array* or a *dictionary*, though the word *map* is also used (Mehlhorn and Sanders 2008). This type of data structure is generally the most flexible way for computers store information. An associative array consists of key/value pairs, where the *value* is the data to be stored and the *key* is the reference to that data.

In table 2.1 the data is non-numeric and associations between keys and values are arbitrary (from a mathematical point of view). There obviously exists no distinct function that can transform a countries name into the name of its currency, thus the computer must explicitly remember the associations between the words in the form of a *hash table*. At the lowest level, computers store information on a vast array of zeros and ones, and the value "Kwanza" only arises through a non-trivial process of encoding and decoding. In order to retrieve it the computer *must* know where it can be found. The hash table takes the input

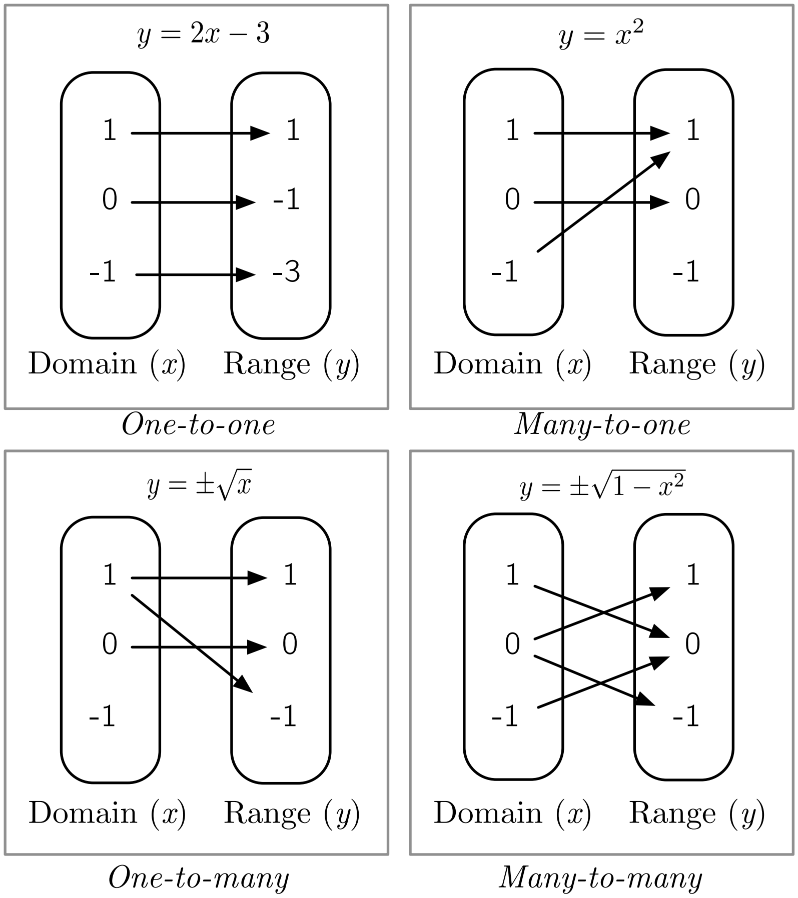


Fig. 2.4 The four mapping classes

of a key, finds the address for the value and returns it. In this way the hash table is literally the association between two sets of data and thus the mapping between them.

The four mapping classes outlined in the above section are not limited to the functional domain. The associative array in table 2.1 is another example of a many-to-one mapping, as many countries have the same currency name. In this vain a one-to-many mapping could be the same keys with values switched to "Former Monarchs" ("France" would map to both "Louis XVI" and "Napoleon III", etc), while a value of "Official Languages" would be a many-to-many mapping ("Canada" maps to both "English" and "French" while both "Canada" and "France" map to "French").

Though most applicably represented in computer science, data structures like associative arrays appear in many other fields. Library card catalogs (one-to-one), multilingual

Table 2.1 An example of key/value pairs (contries and currencies)

key	value
Canada	Dollar
France	Euro
Bahrain	Dinar
Germany	Euro
Angola	Kwanza
USA	Dollar

dictionaries (many-to-many) and address books (many-to-one) are all very straightforward instances of key/value pairs. In a library card catalog the call number even acts as a sort of hash table. In a large library, a book that is placed in the incorrect position on the shelves will likely be lost for a very long time. Thus the system must not only remember the keys (titles) and associated values (the books themselves) but also their positions in memory, their call numbers.

2.1.2 Mapping for digital musical instruments

With an acoustical musical instrument a musician must interact directly with the physical object that produces the sound. In this context, the concepts of "control surface," and "synthesis devices" are not very relevant, as they are intrinsically linked. In the case of an acoustic guitar the pick *could* be considered to be a sort of control device (as it is primarily used for instrumental interaction), with the strings and body a acting as the sound producing section. The problem with this type of approach is that changing the material of the pick, perhaps to give a different feel for the player, will also necessarily modify the sound produced. The same can be said for modifying nearly any aspect of an acoustic instrument: it will change both the control interface and the created sound. This coupling of parameters causes any concept of a *mapping layer* to be irrelevant.

As stated in the introduction, this is not the case for electronic instruments (Hunt et al. 2000). Electronic sensors transduce musical gestures into signals, which are in turn converted into auditory phenomena by amplifiers and speakers. Any arbitrary transformation can happen to the signals[3] in between these two phases. This flexibility is most obvious with outlandish novel instruments like (TODO: maybe pictures?), but is fundamentally true for

[3]Especially digital signals, which are remarkable for their robustness and mutability.

any electronic instrument. An electric guitar senses gesture with a magnetic pickup that transforms the signal of a vibrating string into an electronic signal, which is made audible by an amplifier. Though this can happen directly, more or less reproducing the sound of an acoustic guitar, it is also possible to greatly modify this signal before it is amplified, creating tones that may be unrecognizable as the original acoustic instrument.

The mapping layer

In response to the importance of this uncoupling of parameters, electronic instruments are often conceptualized as having three independent layers (Wanderley and Depalle 2004): (TODO diagram)

- The “gestural controller”: The device with which the musician interacts directly. It generally has sensors that collect gestural data and can provide haptic feedback. The generated signals are output into the mapping layer.
- The “sound generation unit”: This device receives input signals from the mapping layer and uses them to generate sound. This layer can contain melody generating algorithms, sound modifying effects, physical models of acoustical instruments or any other construct that is directly used to produce sound.
- The “mapping layer”: The abstract space that receives input signals from the gestural controller and outputs to the sound generation unit. These signals can be connected and modified independently of actions in the other two layers.

As can be seen above, the words “output” and “input” become ambiguous, whether one is speaking from the perspective of devices (control devices *output* signals that are *input* into the synthesis devices) or the perspective of the mapping layer (the mapping receives *input* from the controller which is *output* to the synthesizer). For the detailed analysis of mappings and mapping devices, this can obviously create confusion. To avoid this, signals arriving at the mapping layer from the control surfaces will be referred to as *source signals* and signals sent from the mapping layer to the sound generation units will be called *destination signals* for the remainder of this thesis. This follows the nomenclature described in Malloch et al. (2013) and the libmapper API in general.

Functional versus systems perspective on mapping

Both the more mathematical perspective of mapping as functions and the computer science standpoint of mapping as association are relevant to DMI design. These two concepts are often referred to as the *functional* and the *systems* points of view for mapping, respectively (Nort 2010).

Once two signals are connected, say the position of a knob and the cutoff frequency of a low-pass filter,[4] it is very possible that the raw numbers sent from the knob are not appropriate as input for the filter. It may be that the knob transmits numbers ranging from 0 - 127 (2^7) and the filter accepts numbers from 0 - 1023 (2^{10}). As a result the filter will always be more or less closed no matter how the user turns the knob. To account for this, the mapping needs to *scale* the source signal (by a factor of 8) to fit the destination range. This is a functional kind of mapping, analogous to section 2.1.1. The source signal may need to be transformed in many ways.

The other, higher-level perspective on mapping deals with the actual connection of source to destination signals. On any mapping network there can exist several devices, each with numerous signals. The act of associating devices with devices, signals with signals can drastically change the character of a DMI or group of DMIs. This is known as the systems perspective on mapping. It is necessary for libmapper and the GUI to be able to assist with both kinds of mappings.

Mapping strategies

For expressive musical networks, simple one-to-one mappings are often insufficient. Kvifte (2008) argues that it is extremely rare to find such associations in acoustic instruments, as the control parameters are usually tightly coupled with several acoustic dimensions. Interfaces with hundreds of knobs and sliders, each one connected to a single sound parameter have thus been found to "...hinder rather than help expressive musical behavior." (Kvifte) In practical experiments where mappings of varying complexity are compared, the most complex were generaly found to be the most expressive and useful (Hunt and Wanderley 2002). However, Goudeseune (2002) states that mappings need to be simple enough for the performer to comprehend them. Goudeseune argues for "...static mappings over dynamic,

[4] A standard synthesis parameter that controls the brightness of a sound, think of the difference between the vowel 'o' in 'food' (low cutoff) and the vowel 'a' in 'sad' (high cutoff).

and simple over complex" and proposes an algorithmic solution to compute them. These "interpolated mappings" are generated by associating single points in the source and destination spaces (i.e. certain performer gestures with certain sounds) and mathematically filling in the spaces between.

One proposed solution to the cognitive complexity of associating many source and destination signals is to create a second mapping layer (Hunt et al. 2000). Instead of dealing with raw sensor output, like acceleration and inclination, musicians can interact with more interesting gestural information such as "jab" or "left-arm swing." These "cooked" parameters are argued to be more meaningful and useful musical information than the raw signals. This approach is explored in Momeni and Henry (2006) for mapping both to audio and visual synthesis. The conventional wisdom that mappings need to be complex, yet transparent and meaningful all point to the necessity of a tool for the intuitive and expressive configuration of mappings.

2.1.3 libmapper

The McGill Digital Orchestra project[5] began in 2006 with the aim of helping researchers and performers in music technology to work collaboratively in creating hardware and software solutions for live performance with digital technology. The libmapper project began in response to the difficulty of creating dynamic musical mappings in a collaborative setting (Malloch et al. 2008). In its most basic state, libmapper is a library for connecting things. As described by its website:

> "libmapper is an open-source, cross-platform software library for declaring data signals on a shared network and enabling arbitrary connections to be made between them. libmapper creates a distributed mapping system/network, with no central points of failure, the potential for tight collaboration and easy parallelization of media synthesis. The main focus of libmapper development is to provide tools for creating and using systems for interactive control of media synthesis."[6]

[5] The McGill Digital Orchestra. [Online]. Available: `http://www.music.mcgill.ca/musictech/DigitalOrchestra/`. Accessed July 9, 2013

[6] libmapper: a library for connecting things. [Online]. Available: `libmapper.org`. Accessed June, 2013

Without libmapper, DMI designers are usually required to "hard-code" mappings into their designs. This has the disadvantage of being slow to modify, as it might be necessary to re-compile[7] code any time a change is made. If the DMI is built in a development environment like Max/MSP modifications can be more quickly implemented. Max/MSP is a "high-level" abstraction on top of machine readable code, so Max/MSP programs are prone to slowness and cross-compatibility issues, inhibiting collaboration (Place and Lossius 2006). In either implementation it is difficult for someone other than the original designer to modify mappings.

As a C[8] library, libmapper does not introduce many abstractions on top of the data and can work quickly. Any device that embeds libmapper in its code can communicate with other devices that have done the same. In a libmapper network devices communicate with one another directly, as opposed to through some centralized network device. This means that less data overall needs to be sent over the network, and failure of a single device (like the router) will not crash the entire system (Malloch et al. 2013), an especially dire situation during live performance.

Another advantage of libmapper, which is especially relevant to this project, is the ability to create an administrative device. These "monitors" can query libmapper devices for data, and thus collect data on the network overall. Monitors also are able to create, destroy and modify connections on the network. This allows for external visualization and control of a libmapper network.

Open sound control and libmapper syntax

Like any communication, communication between digital devices functions well only when the devices speak the same language. In the Internet age this becomes particularly relevant: the vast array of continuously connected devices, sending and requesting information would instantly collapse if every developer coded to his or her own idiosyncrasies. To prevent this, computer scientists make use of various communication "protocols" when creating software. Hypertext Transfer Protocol (HTTP) is the most famous example of such a system.

At its core, libmapper builds its on language on top of the Open Sound Control (OSC) protocol, as described by Wright and Freed (1997). OSC defines the format for messages

[7] A process in which human-readable code is translated into something the computer can understand. This can take anywhere from a few seconds to days.

[8] An extremely popular, multi-purpose programming language.

that are sent between sound producing devices (as implied by the name), but can also be used for related multimedia devices such as stage lights or vibrating motors. It provides means for flexible, high-resolution communication and was intended to replace MIDI[9], the 30-year-old standard for musical instrument communication.

OSC formats messages much like Internet URLs, arbitrary strings of characters separated by '/' characters. libmapper messages also take on this format, using the structure to expose hierarchy of signals:

- `tstick.1/raw/accelerometer/1/x`: The data for the 'x' dimension of the first accelerometer of the first instrument of class "tstick" on the network (see TODO for a description of the gestural controller T-Stick). Here the word "raw" denotes that no pre-processing has been applied to this signal.

- `tstick.1/raw/accelerometer/2/y`: A signal transmitting the data for the same instrument as above, but the 'y' dimension of the second accelerometer.

- `tstick.1/cooked/accelerometer/2/amplitude`: A "cooked" signal. All three dimensions of accelerometer 2 are combined to compute the overall acceleration of the point. These signals can also be cooked to expose angle and elevation as signals.

- `granul8.2/filter/evelope/frequency/low`: The data for the low-end cutoff for the shape of the filter for the instrument named "granul8.2" (a granular synthesizer, thus a destination device).

This structure of signal names aims to be semantically relevant, and allows a GUI to display hierarchical structure of networks. Any one of the above signals transmits not only the signal's value, but also metadata. Signal metadata usually includes data type, length (single number vs. vector), units like volts or meters per second, maximum value and minimum value. Designers can "tag" signals with any extra metadata they may wish to add, such as physical position, color or owner's name. In the GUI it is necessary to allow users to view and manipulate any arbitrary kind of signal metadata.

To make signal names as coherent and consistent as possible, libmapper makes use of the *Gesture Description Interchange Format* (GDIF) (Jensenius et al. 2006), which

[9]MIDI Manufacturers Association - The official source of information about MIDI. [Online]. Available: `www.midi.org`. Accessed July 11, 2013

provides a standard for motion capture data. Structures are given short, semantically relevant names. GDIF also provides a standard vocabulary for describing motion with dimensions such as "weight," "space," "time" and "flow." Though these standards are not enforced, as libmapper signals can be given any sort of names by their creators, most extant libmapper-enabled devices use them.

Structure of libmapper networks

In order to maintain internal consistency, libmapper introduces a naming convention of its own. At the heart of any libmapper network are *signals*, defined in Malloch, Sinclair, and Wanderley (2013) as:

> "Data organized into a time series. Conceptually a signal is continuous, however our use of the term signal will refer to discretized signals, without assumptions regarding sampling intervals."

Here Malloch et al. are referring to digital as opposed to analog signals (hence the use of the term "discretized"). Notice how signals are not necessarily numeric by this definition, though they will almost certainly will be going forward. Signals are the only information actually passed from control surfaces to synthesizers, all other data structures exist to organize and label them. *Source signals* is data entering libmapper from control surfaces while *destination signals* belong to synthesizers and receive data. A *connection* is a bridge between two signals. Once a connection is created within libmapper, a source signal begins sending its data to a destination signal. A single source signal can be connected to many destination signals (a one-to-many mapping), but at the time of the writing of this document single destination signals cannot receive input from many source signals (a many-to-one mapping). Justification for this lack of functionality is discussed in Malloch et al. (2013).

Devices are essentially groups of signals. A device often has some kind of physical entity that makes the grouping logical, e.g. the "T-Stick," which has many "child" signals. Within software the grouping is usually a discreet computer program. With development environments like Max/MSP users are free to group signals into devices however they may wish. As mentioned previously, libmapper devices do not send all signal data to some centralized router, and instead work directly with one another. In order to accomplish this devices must be explicitly *linked*. Figure 2.5 demonstrates instances of libmapper devices, signals, links and connections.

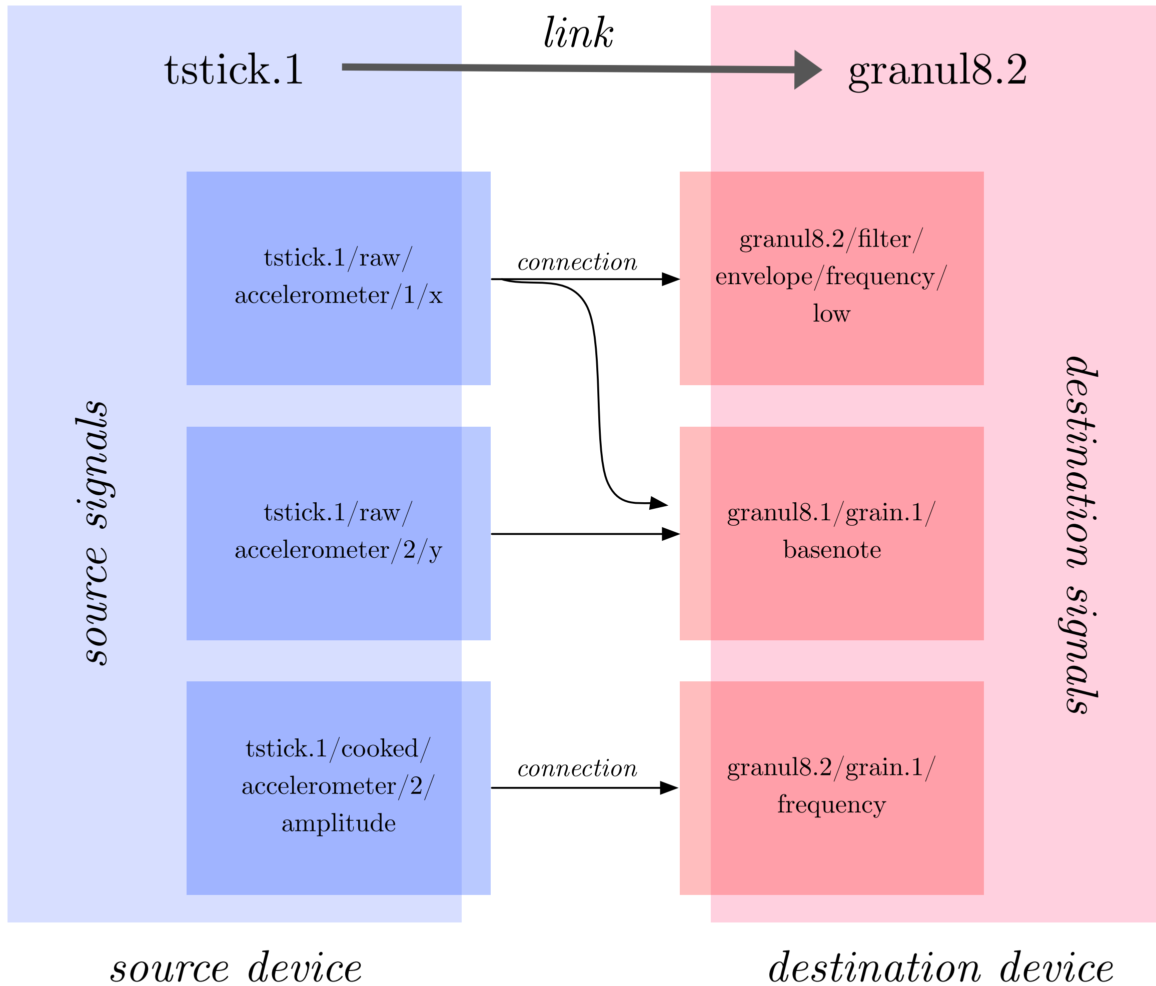


Fig. 2.5 A simple libmapper network

Devices and signals can carry a variety of *metadata*. Devices usually list the number of child signals they possess and their location on the network (IP address and port). As previously stated users can tag devices and signals with arbitrary metadata. Connections have a much more specific set of metadata.

Connection properties

Creation of links and connections is mapping from the systems perspective, but libmapper also allows for functional mapping through the modification of connections. This can be accomplished by altering certain properties possessed by every libmapper connection:

- **Expression**: A mathematical equation relating the source (x) to destination (y) values. An expression of $y = x$ will simply pass through source values, while an expression of $y = 3x + 2$, will apply a linear transformation to the source data (e.g. a value of 1 will be output as 5). libmapper supports a variety of expressions, including exponential functions, trigonometric relations, comparison operators, derivation and integration.

- **Range**: An array of four numbers containing the user-specified maximum and minimum values for both the source and destination signals.

- **Mode**: The type of connection, this influences the effect of the expression and range properties and can be one of four categories:

  - *Linear*: libmapper automatically scales the output such that it fits the destination range, based on the source range. For example, if a certain connection has a source range of $[0, 1]$, and a destination range of $[5, 10]$, libmapper will automatically apply an expression of $y = 5x + 5$, such that the minimum and maximum source values will correspond to the minimum and maximum destination values respectively. A source value that is outside of this source range will result in a destination value that is also outside of the range. In this mode the user cannot directly modify the expression.
  - *Calibration*: The same as the linear mode except the source range parameter is ignored. libmapper instead polls the source signal to find the source range directly.
  - *Bypass*: Source values are sent directly through to the destination signal, as would happen with an expression $y = x$.
  - *Expression*: The user is able to set the expression to any arbitrary relation.

- **Boundary**: What is to happen to data values when they extend beyond the destination range. There are four options:

  - *None*: Values are passed through unchanged.
  - *Clamp*: Values outside of the boundary are constrained to that value.
  - *Mute*: No values outside of the boundary are passed to the output.
  - *Wrap*: Values exceeding the maximum are "wrapped" back to the minimum bound and vice versa.
  - *Fold*: When the signal passes outside of the boundary, the value is inverted back onto the destination range.

- **Mute**: A true-or-false value muting and un-muting data sent over the connection.
- **Send as instance**: Not all signals on libmapper networks are unique and long lasting, a good example being a keypress on a keyboard. During the keypress, data like aftertouch and release can be sent, making it a bona fide signal. However, musicians constantly create and complete keypress events during performance with keyboard instruments. To maintain every keypress as a unique signal with unique metadata would be tremendously unhelpful for mapping. Also, forcing a user to map every keypress event individually would make live performance impossible.

  To support this, libmapper gives connections the *send as instance* property. Sending data as an instance means that libmapper treats the connected signals as instances of a general class of signals. New instances of a signal class will be treated like previous instances and do not need to be mapped individually.
- **Link scope**: Not a property of connections, but of links. By default links are "scoped" to notify destination devices of the creation and destruction of signal instances on linked source devices. For intermediate devices, ones that function as both source and destination, this may not be the desired behavior. If device `A` is linked to intermediate device `B`, which is in turn linked to device `C`, then `C` will not be notified of instance events on `A` with default link scope settings. The user can modify the scope of link `B` → `C` to include `A` if desired.

libmapper bindings

A final crucial libmapper feature is its multi-language *bindings*. The C language is "low-level," in that it is very procedural and does not allow for very abstract data structures. This makes it extremely flexible, but difficult and time consuming to use. To make libmapper more friendly for different kinds of developers, "bindings" have been created for the higher-level Python[10] and Java[11] programming languages. libmapper functions are bound to other languages using the Simplified Wrapper and Interface Generator (SWIG)[12]. SWIG automatically writes a kind of dictionary that interprets function calls from other languages to the original C. Automatically generated files sit in-between the controlling code and the original library.

Though concept of "mapping" itself is extremely abstract, the libmapper API places it into a concrete context. libmapper is not only means of organizing networks though the creation and destruction of links and connections, it is also a tool for customizing response by its support for modifying connection properties. In this way it can serve both the high-level systems perspective and the low-level functional view of mapping. Though designed for musical devices, the API's loose framework could readily be applied to any type of multimedia system. libmapper is an extremely powerful, flexible tool and requires a user interface that can elegantly deploy its full range of capabilities.

2.2 Data Visualization

The graphical user interface described in this thesis is a purely visual interface. No means of auditory or haptic response was implemented, or even seriously considered. Creating an auditory tool for controlling musical instruments is obviously problematic and most personal computers provide no means of produce haptic feedback. This obviously limits the usable dimensions, but also greatly simplifies the problem of how to best represent the tremendous variety of libmapper networks.

Fortunately graphic designers and statisticians have already deeply probed the problem of how to best display data visually. It is necessary here to briefly review some of this work, especially the techniques relevant to the creation of a libmapper GUI and visual systems from which inspiration was drawn.

[10]Python Programming Language - Official Website. [Online]. Available: `http://www.python.org/`. Accessed July 17, 2013

[11]java.com: Java + You. [Online]. Available: `java.com/en`. Accessed July 17, 2013

[12]Simplified Wrapper and Interface Generator. [Online]. Available: `http://www.swig.org/`. Accessed July 17, 2013

2.2.1 Graphical dimensions

The visual dimension can be broken down into many sub-dimensions. These dimensions are not fully separable, but doing so creates a useful paradigm for understanding and creating solutions for our visual problem. Bertin (1983) presents a simple vocabulary for categorizing graphical objects and relationships.

Table 2.2 Bertin's graphical relationships

Marks	Points, lines and areas
Positional	1-D, 2-D and 3-D
Temporal	Animation
Retinal	Color, shape, size, saturation, texture and orientation

Visual presentations use marks to encode information by way of their positional, temporal and retinal qualities. In table 2.2 *retinal* properties are so called because the eye is sensitive to them independently of their position. Though the third positional dimension is relevant and would be useful, it is currently beyond the scope of this research, not to mention the hardware on which the GUI is to run.

Cleveland and McGill (1984) expand on this vocabulary, enumerating further sub-dimensions of marks and retinal properties. An experiment is described in which subjects are asked the relative values of various visual objects (e.g. the first box is 50% larger than the box on the left), for various visual dimensions. From the data, they were able to create a ranking of visual dimensions for quantitative information. In figure 2.6, differences between objects are more accurately perceived when the difference is encoded using a variable higher up the chart. Note that variables like shape, texture and opacity are not included.

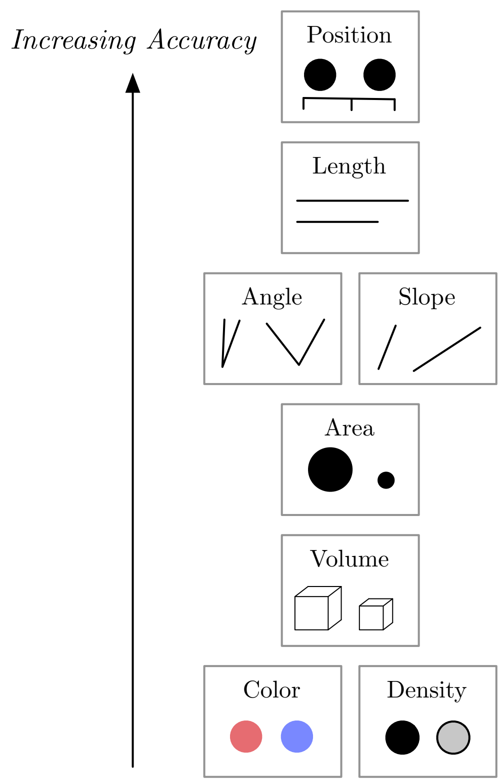


Fig. 2.6 Cleveland and McGill's rankings for quantitative perceptual tasks.

Mackinlay (1986) uses this ranking to expand into non-quantitative data sets. Nominal information is that in which elements can be understood to be similar or dissimilar to one another, yet have no definite order or value. libmapper uses nominal information in the form of device, signal, link and connection names, as well as connection modes and boundary conditions. Ordinal data fits between quantitative and nominal, ordinal items are understood to be greater than or less than one another, while having no definite numerical ratios. If multiple devices of the same class are present on the same libmapper network, libmapper will append ordinal numbers to the end of their device names (e.g. `tstick.1`, `tstick.2` and `tstick.3`).

Table 2.3 Mackinlay's Graphical Rankings

quantitative	**ordinal**	**nominal**
position	position	position
length	density	color hue
angle	color saturation	texture
slope	color hue	connection
area	texture	containment
volume	connection	density
density	containment	color saturation
color saturation	length	shape
color hue	angle	length
texture	slope	angle
connection	area	slope
containment	volume	area
shape	*shape*	volume

In table 2.3, items in italics are considered unsuitable by Mackinlay. Though position is the most accurate dimension for all types of data, dimensions like *length* differ widely. For the visualization of libmapper networks, it is often necessary to encode many dimensions of data onto a single mark. Devices, signals, connections and links all have a set of meta-data with quantitative, ordinal and nominative information. In the design of an effective GUI it will be necessary to properly associate high-accuracy visual dimensions to network properties that require them and reserve low-accuracy dimensions for those that do not. In this way the problem of this thesis conveniently becomes one of mapping: how can we best correlate visual dimensions with properties of libmapper networks?

2.2.2 Relevant visualization techniques and systems

Encoding Color

"Color" itself is a multi-dimensional phenomenon that does much to communicate information in modern user interfaces. Since color was an uncommon feature of computer displays at the time neither Bertin (1983) nor Cleveland and McGill (1984) explore its use in depth. Cleveland and McGill simply state that it is not good for encoding quantitative information. Mackinlay (1986) elaborates on this, breaking color into "hue" and "saturation," upgrading its use for ordinal and nominal data.

Tufte (1990) provides a definite procedure for incorporating color into evidence displays[13]. Techniques are gleaned from centuries-old map making and applied to computer interfaces. Principal rules, summarized and expanded from Imhof (1982), are found to be (paraphrased):

- *First rule*: Bright colors are painful when used uninterrupted over large areas or when placed adjacent to each other, but can be extremely powerful when used sparingly while accompanied by dull tones.

- *Second rule*: Light, bright colors produce unpleasant results when accompanied with the color white.

- *Third rule*: Background and base colors should be muted or neutral. For this reason, *gray* is regarded to be the most versatile of colors.

- *Fourth rule*: Two or more large, enclosed areas within a single display cause the image to fall apart. Unity can be maintained if the colors of one section are interspersed throughout the other. "All colors of the main theme should be scattered like islands in the background color."

Links and causal arrows

For the visualization of networks, the idea of a visual "connection" becomes very important. This linking action is usually accomplished by an arrow-like object in evidence displays. Tufte (2006) enumerates numerous guidelines for incorporating line-like objects

[13] Tufte's favorite term for data-driven graphics

into presentations. Again drawing inspiration from map making (an obvious inspiration for "mapping"), the use of differentiation among linking arrows is greatly emphasized: "Nouns name a specific something; arrows and links are too often non-specific, generic, identical, undifferentiated, and ambiguous." The use of many line properties, such as dashing, arrow-heads and color can better illustrate a variety of influences in a linked chart.

Tufte also cautions against using heavy line weights when unnecessary, as it effectively decreases display resolution. Thick lines are also more likely to create $1 + 1 = 3$ noise, or the effect of negative space acting as a display feature.

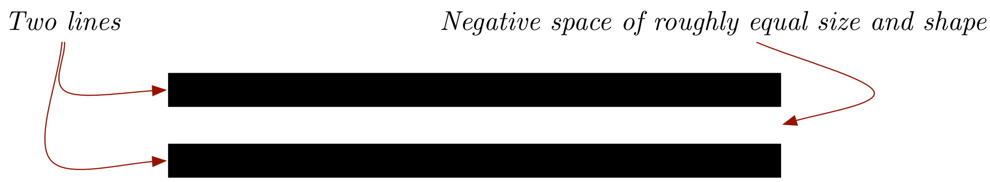


Fig. 2.7 An example of Tufte's $1 + 1 = 3$ noise

In figure 2.7 the negative space between the two lines appears as its own white line, as opposed to simply empty space. In displays with numerous or thick lines this can cause negative space to compete with informative features, attenuating the overall effectiveness of the display. $1 + 1 = 3$ noise plagues dense computer user interfaces, borders and other non-essential display features should be lightened, thinned and removed when at all possible.

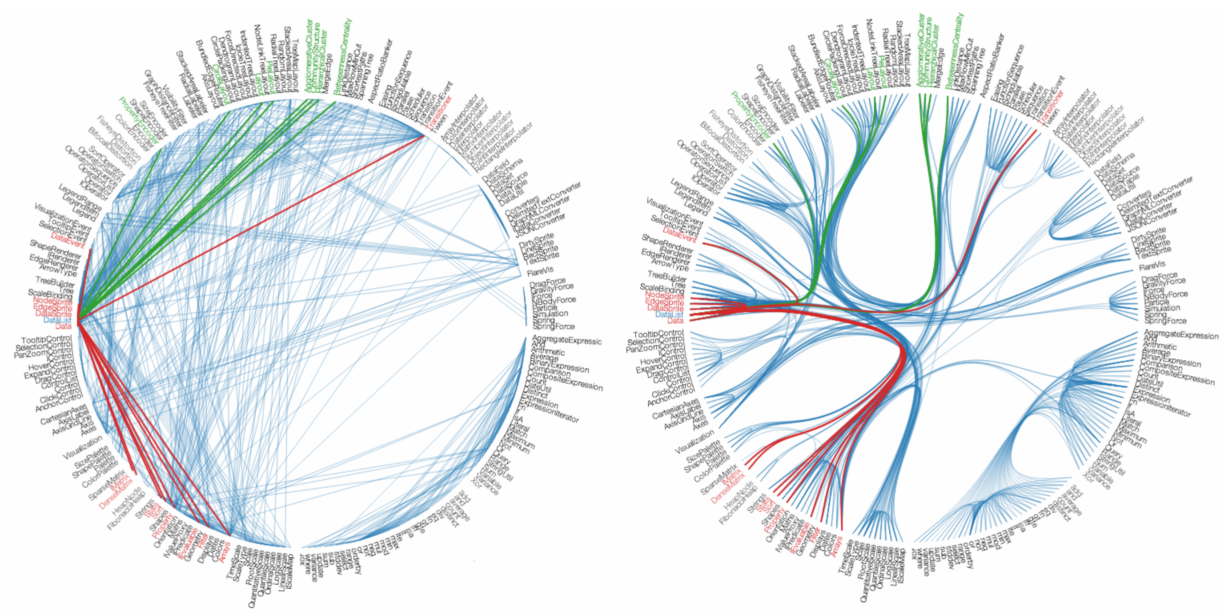
Hierarchical edge bundling

Of course, in diagrams with tremendous amounts of connections no amount of thinning and coloration can create an informative display. The technique of hierarchical edge bundling (Holten 2006) groups lines based on adjacency relationships. Displays take advantage of hierarchical information encoded within the dataset. Linking arrows are curved towards other arrows that are connected to related elements. Figure 2.8 demonstrates this effect for arbitrary data.[14]

In a libmapper system, this would mean that connections between signals on the same device will be pulled towards one another. If a hierarchical structure exists in the naming

[14]Images courtesy of: mbostock - The d3 visualization library. [Online]. Available: `https://github.com/mbostock/d3/wiki/Gallery`. Accessed July 24, 2013

convention, connections between related signals will experience an even stronger force between them. For example, the connections from signal `tstick.1/raw/accelerometer/1/x` will be bundled tightly with connections from signal `tstick.1/raw/accelerometer/1/y`, but less tightly to `tstick.1/raw/accelerometer/2/x`. Any of these connections will not be pulled at all towards connections from signals on other devices.



No hierarchical edge bundling With hierarchal edge bundling

Fig. 2.8 A dense interconnected network displayed with and without hierarchical edge bundling techniques

Braun

Braun is an application for visualizing OSC data flows on a scrolling graph (Bullock 2008). Users are presented with options to adjust what dimension is displayed on the y-axis, the x-axis being reserved for time. Multiple data flows can be viewed on the same set of axes, and time scales can be set arbitrarily, giving the users an overall impression of trends in OSC messages over their networks. An extremely simple visualization, it creates a sort of oscilloscope, but for network OSC data.

2.3 User Interface Design

2.3.1 A brief history of electronic user interfaces

2.3.2 Task analysis

2.3.3 Recall and recognition?

2.3.4 Collaborative network interfaces

MPG Care Package (Wolek 2010)

2.3.5 The model-view-controller architecture

MVC Krasner Pope (Krasner and Pope 1988)

2.3.6 User centric design

1. Organizational context (Kling 1977)
2. Usability testing (Corry et al. 1997)
3. Information professionals (Schulze 2001)

2.4 Relevant User Interfaces

1. Inclusive interconnections (Booth 2010)
2. Sense Stage (Baalman et al. 2010)

2.4.1 Junxion

Junxion (STEIM 2004)

2.4.2 Osculator

Osculator: mapping OSC stuff (Wildora 2012)

2.4.3 Other similar interfaces

1. mpgcarepackage?

2. Integra (Bullock et al. 2011)

3. Eaganmatrix: GRID VIEW! (Audio 2103)

4. Patchage: a linking, dragging, connecting interface (Robillard 2011)

2.4.4 Prior interfaces for libmapper

Vizmapper (Rudraraju 2011)

2.5 Summary

Music technology is an inherently interdisciplinary field

Chapter 3

Design & Implementation

Development of a graphical user interface for libmapper creates a unique challenge. Obviously such an interface is a practical tool, and should function as such, yet it also must work in concert with DMIs which are inherently designed for creative use. For the purposes of this project, the assumed solution to this innate paradox is to provide the user with multiple independent modes of control. This assumption based on experiences with prior user interfaces for libmapper: for each interface users reported excellent functionality for certain use cases, and poor functionality for others. libmapper itself is an extremely flexible API that makes few assumptions as to the network of devices and signals or how they are mapped. It is thus fitting that a GUI for libmapper would be equally as flexible. In lieu of a single perfect solution for network visualization and interactivity, providing users with various independent solutions provided a good compromise.

3.1 Design Background

Work on this project began with a moderately featured, little used GUI for libmapper known as "webmapper." Webmapper was created at IDMIL as an attempt to replace the Max/MSP GUI as the result of limitations described in section 2.4, the principle among which being the cross-platform incompatibility of Max/MSP. It was thought that a browser-based approach would greatly simplify the process of creating cross-compatibility with all major operating systems and even mobile devices.

Webmapper utilizes the Python bindings for libmapper by registering an administrative monitor to communicate with a libmapper network. The monitor can create and modify

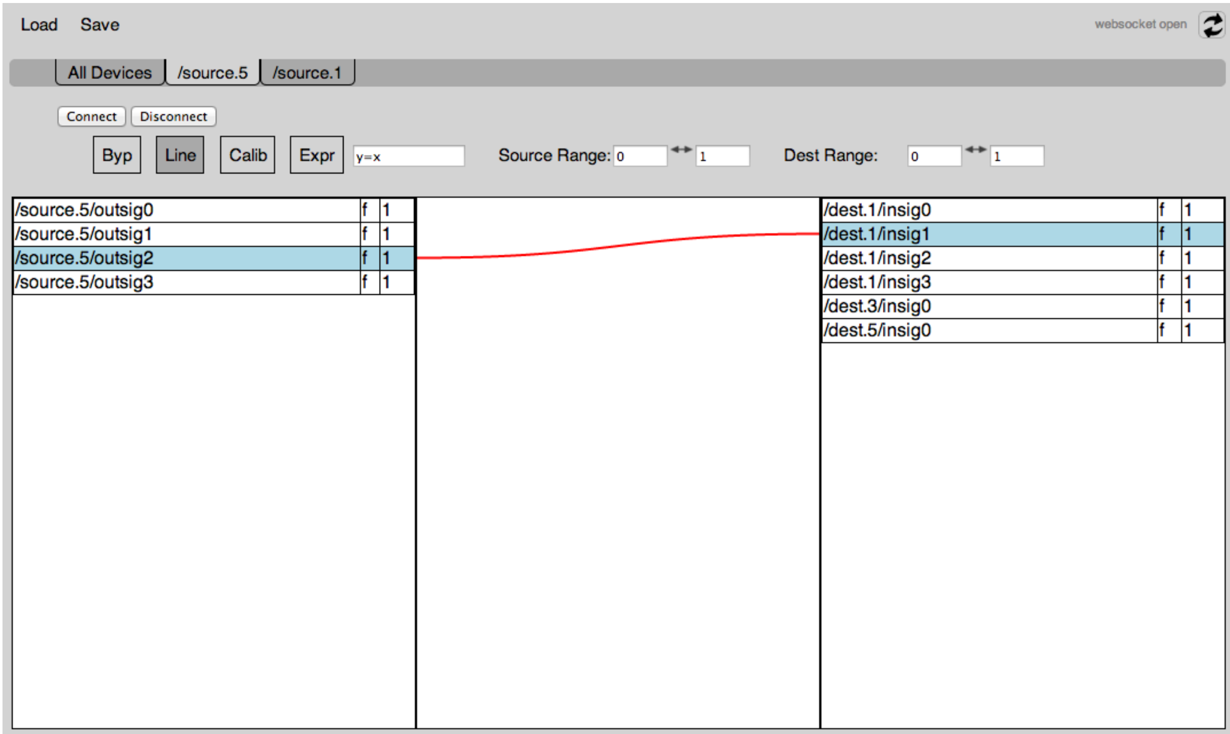


Fig. 3.1 The webmapper interface

connections or links, as well as query the network as to what devices, signals, links and connections are present. The webmapper code creates a simple HTTP server and attempts to open Google Chrome[1] on the host computer. If Google Chrome is not present, the user must navigate directly to the server using the web address `localhost:50000`. The monitor communicates with the libmapper network and the local server, the browser is able to see messages the monitor "posts" to the server (such as 'new device') and respond to them appropriately. The browser in turn can send messages to the server (like 'connect') that will propagate up to libmapper itself, eventually resulting in a message cascading back down to the browser reflecting the change to the network (such as 'new connection').

The interface itself is written for a web browser using the scripting language JavaScript[2] to control web-standard HyperText Markup Language (HTML) elements and Cascading Style Sheets (CSS). The JavaScript code stores four main data structures: devices, links,

[1] Chrome Browser. [Online]. Available: `https://www.google.com/intl/en/chrome/browser/`. Accessed July 17, 2013

[2] JavaScript — MDN. [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`. Accessed July 17, 2013

connections and signals. The code never directly modifies any of this data, and instead waits for messages relayed from libmapper. For example: if a user creates a new link, webmapper does not add the link directly to the links array but simply sends a message to the network. If it receives back a 'new link' message, only then does it add the new link to the array. This is done to ensure that the data structures within webmapper always reflect what is actually present.

Figure 3.1 displays the look of the interface before this project began. Users are able to perform all libmapper functions: connecting, linking and modifying connections, but only the simplest of feature sets is included. In order to form a connection the user must click on a source signal, click on a destination signal and then click on a button labeled "connect." Many useful features of the Max/MSP interface, such as column headers, table sorting, drawing connections and search filtering are not present.

3.1.1 Evaluation of graphical variables

The list in table 3.1 is by no means a complete set, as libmapper may yet expand to include data like device position and users are able to tag devices/signals/links/connections with any extra data they may want.

A fourth data category *boolean* has been added to specify data that only has two values (*true or false*), as it is a common metadata feature. Boolean information is not covered in the Mackinlay paper. Going forward it will be treated more or less like ordinal data, as *true* obviously has a relationship to *false*, even though there is no quantitative value associated with them.

Table 3.1: libmapper metadata types

Devices		
quantitative	*ordinal*	*nominal*
number of inputs	device ordinal	device name
number of outputs		
ip address		
port		

Signals

quantitative	*ordinal*	*nominal*
length	direction	parent device name
minimum value		signal name
maximum value		data type (float, integer, etc.)
sampling rate		units

Links

quantitative	*ordinal*	*nominal*
		link name
		source device name
		destination device name
		scope

Connections

quantitative	*ordinal*	*nominal*	*boolean*
source minimum	instance number	boundary modes	mute
source maximum		connection mode	send as instance
destination minimum		destination data type	
destination maximum		mute	
		expression	

3.2 Development of a flexible interface

Prior GUIs for libmapper have been successfully used for some time, but all have failed to become a standard for the same reason: they cannot accommodate all possible use-cases of libmapper. List based views like the Max/MSP GUI and webmapper cannot show hierarchies while the cluster view implemented in vizmapper can be overly cumbersome for interaction with simple networks. Especially with so much work already completed on prior GUIs, it is more suitable to integrate different approaches into a single GUI than to begin work on some new, hopefully superior approach that would likely prove to be flawed like all that came before it.

Interface integration is accomplished through an extremely simple approach: a drop-down menu is added to the upper left corner of the interface. Options on this menu represent different visualization modes available to the user. By selecting a new visualization mode the GUI drastically changes it appearance, replacing every visual element excepting the upper toolbar.

3.2.1 Top toolbar

Certain tasks and information providing structures are sensible to include across visualization modes. In light of this, a static toolbar is presented at the top of the GUI. This toolbar contains all administrative controls and connection modification fields.

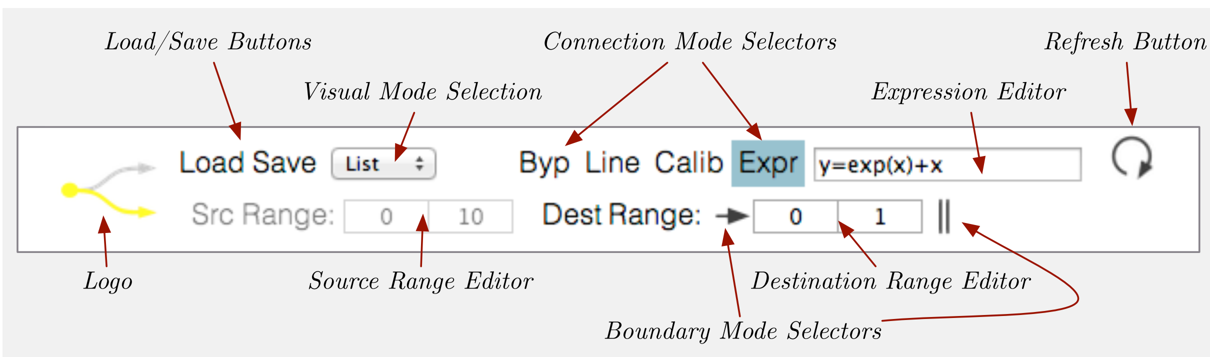


Fig. 3.2 The upper toolbar

- **Administrative controls**
  - *Load/Save buttons*: These elements respond to clicks and save and load mappings, as discussed in section 3.7.
  - *Visual mode selection*: A drop-down menu containing all possible view modes (at the writing of this thesis: List, Grid and Hive).
  - *Refresh Button*: When clicked, all data residing on the GUI is erased and re-gathered. This is useful if the monitor somehow desynchronizes with the network.
- **Connection modification**

– *Connection mode selectors*: If a single connection is selected within the GUI, this array of buttons allows the user to choose between the available connection modes.

– *Expression editor*: Here the user inputs a custom expression, if in "Expr" mode, in other modes this field displays the connection's expression but is not editable.

– *Source range editor*: These two numbers reflect the maximum and minimum values of the input signal, is only editable in the "Line" connection mode.

– *Destination range editor*: Same as above but for the destination signal. Due to boundary conditions these fields are useful in all modes.

– *Boundary mode selectors*: Two buttons that cycle through five boundary modes for both the maximum and minimum destination value. A graphic exists to represent each mode.

All interface features not present in the top toolbar are part of the current visualization mode and are placed into a "container" element below, occupying the remainder of the window.

3.2.2 List view

The first implemented, and thus far most functional view mode for the GUI is the "list" view. Based heavily on the Max/MSP GUI described in section 2.4, the list view provides the most straightforward way to visualize and interact with libmapper. Two tables dominate the visible area, listing source elements on the left and destination elements on the right. Bézier curves form lines between associated list elements on each each list. Because these curves do not always represent the same data structures, the lines themselves are referred to as *arrows* by the GUI code, and by this document.

The view itself is divided into two major modes: "All Devices" and individual linked source devices. Switching between these modes is accomplished through tabs that appear at the top of the container, much like the tabs that appear in modern web browsers. In the All Devices tab, every device displayed on the network is listed in one of the two columns, as in 3.3. Source devices are listed in the left table, while the right table lists destination devices. Intermediate devices, such as implicit mappers described in (Goudeseune 2002), will be listed in both tables. Here arrows represent links between devices. Currently the

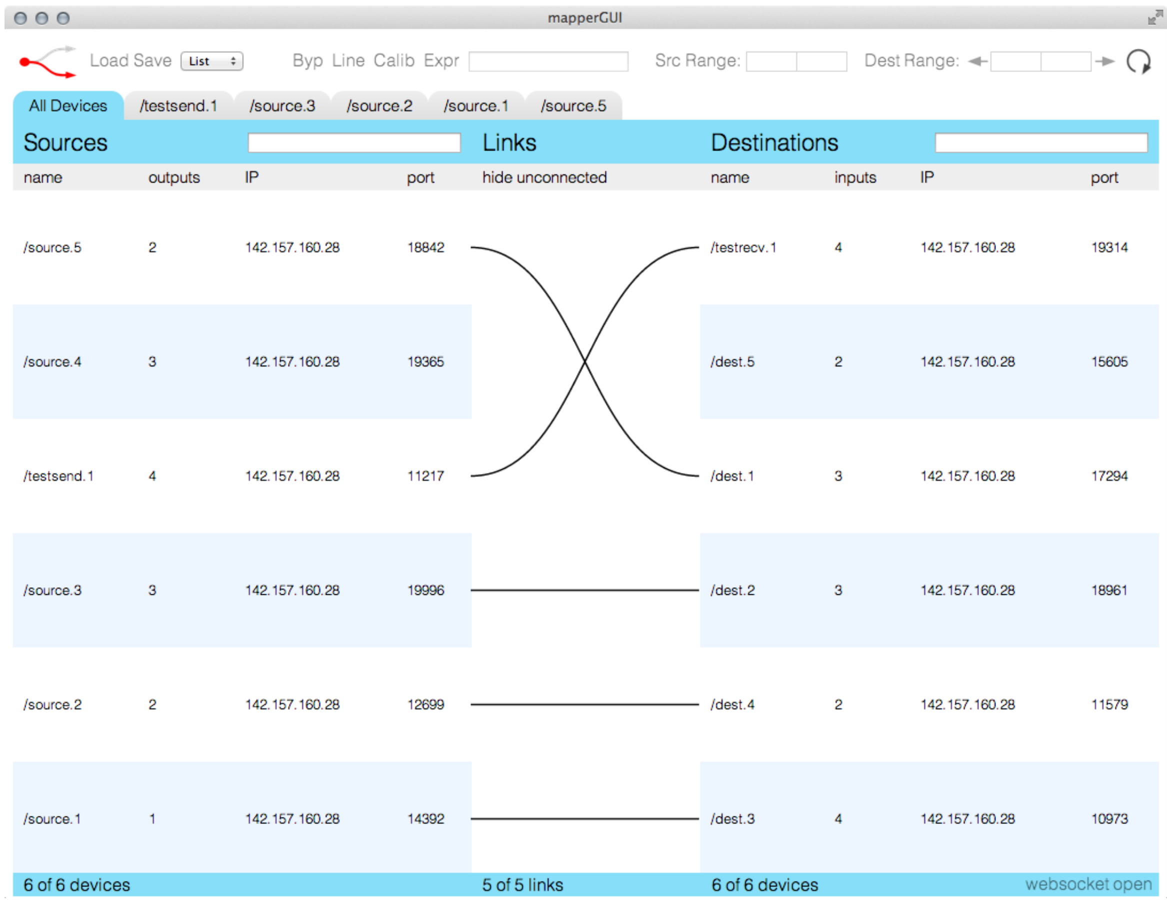


Fig. 3.3 The list view with all devices selected

GUI provides users with names, the number of child signals, IP addresses and a port for every device. Since no connections or signals are displayed, most of the top bar is disabled in the All Devices tab. Saving and loading is also disabled.

The GUI draws a tab for every source device with at least one link to a destination device. Clicking on any of these devices will redraw both tables. The left table now shows all child signals for the selected source device while the right table displays child signals for every destination device linked to the selected device.

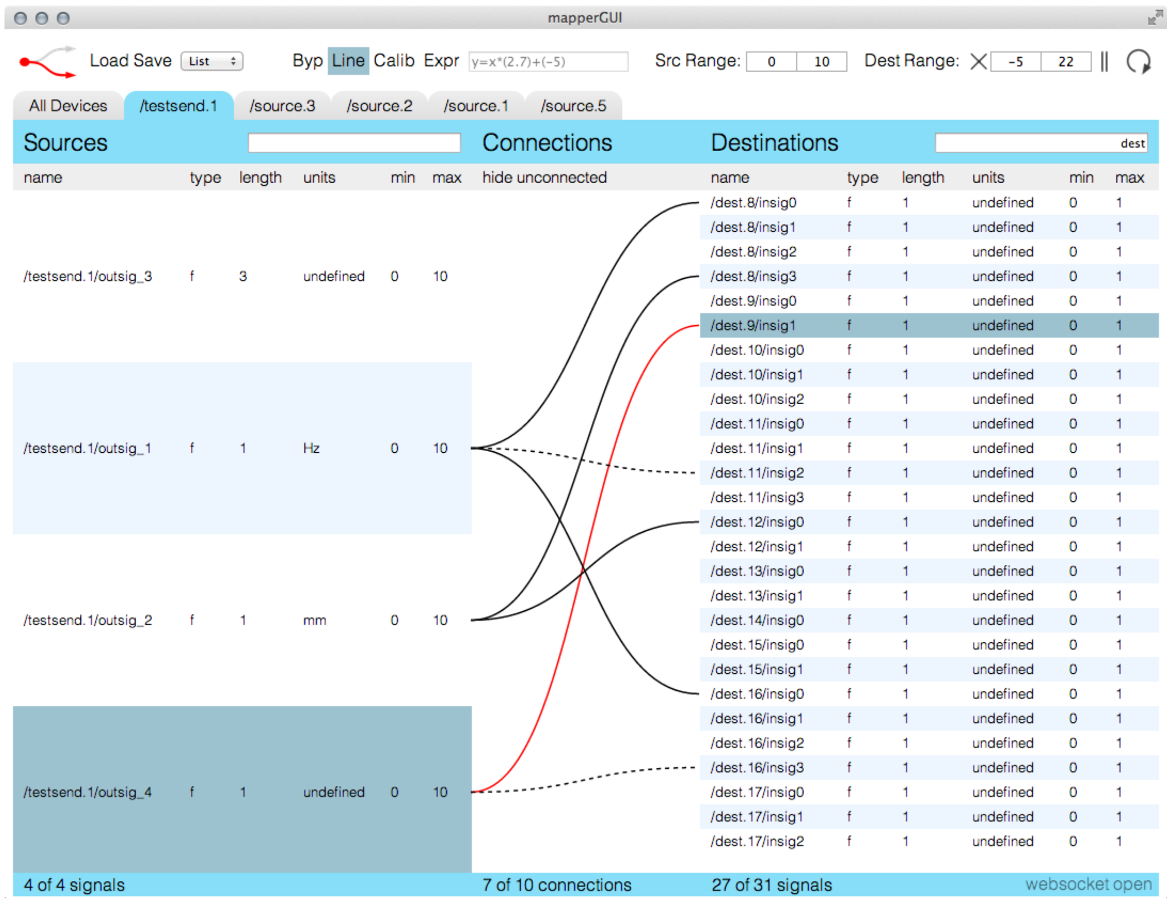


Fig. 3.4 The list view with device **testsend.1** selected

3.2.3 Grid view

3.2.4 Hive plot

3.2.5 Cluster view (vizmapper)

3.3 Control Features

3.3.1 Creating connections/links

3.3.2 Modifying signals

3.4 The Model-View-Controller

Because a modular design is desired, the Model-View-Controller (MVC) metaphor for structuring software applications as described in [KrasnerPope88] was used as a general framework for structuring the application. In fact, the whole scale swapping in and out of independent visual modes can be thought of as a quintessential implementation of MVC.

3.4.1 The model

The model consists of an abstract copy of the network, residing on the local machine. Independent views can consult this data, but cannot directly modify it.

3.4.2 Controller-view pairs

3.5 Graphical Design

wiggly arrows

3.5.1 Typography

3.6 User Centric Design

use cases

3.7 Saving and Loading

3.8 Robustness and Responsiveness

speed tests

3.9 Creation of a Standalone

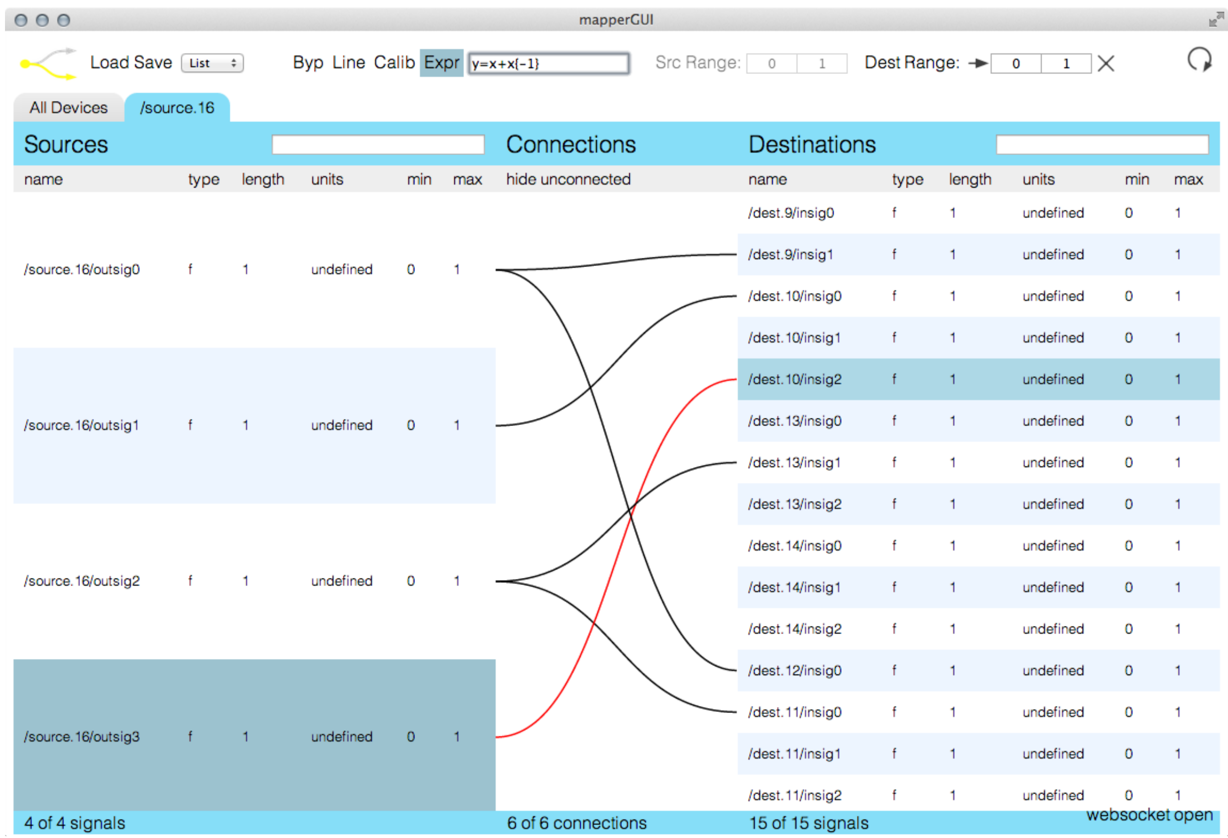


Fig. 3.5 The list view after redesign

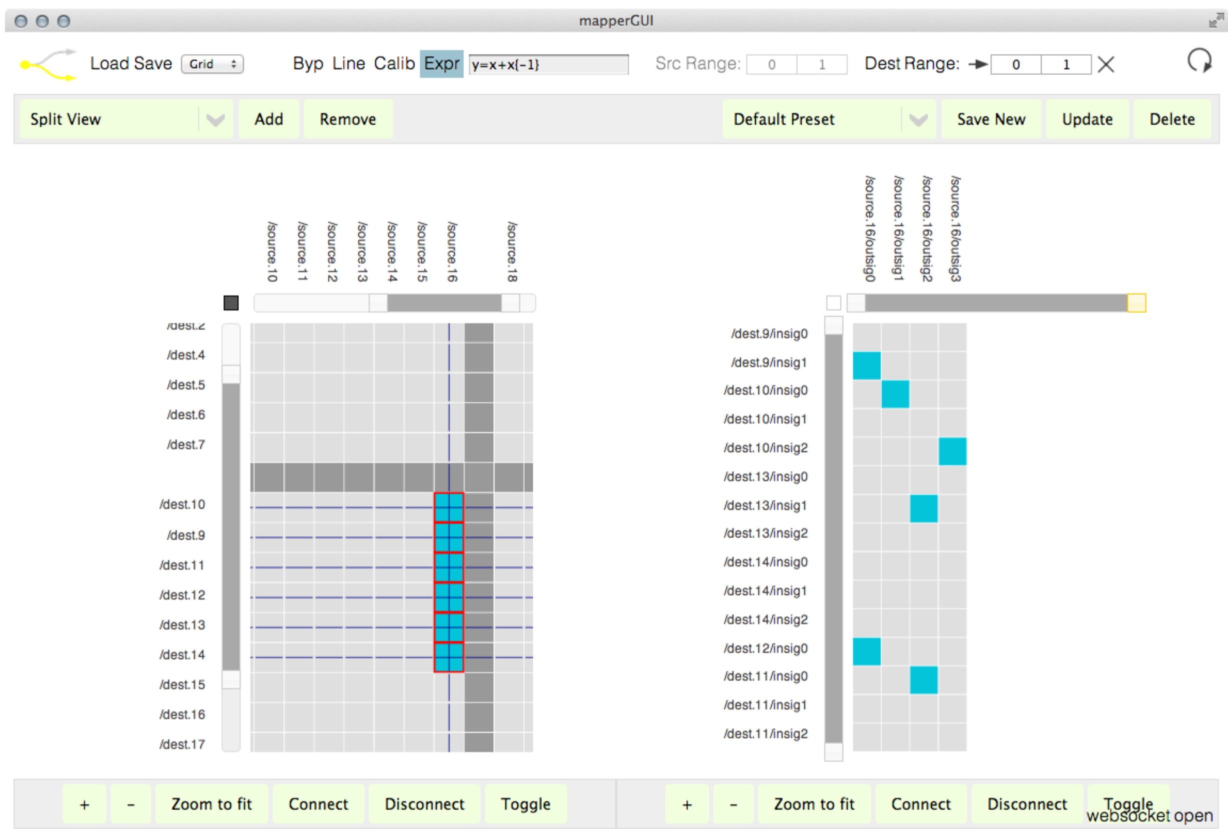


Fig. 3.6 The grid view

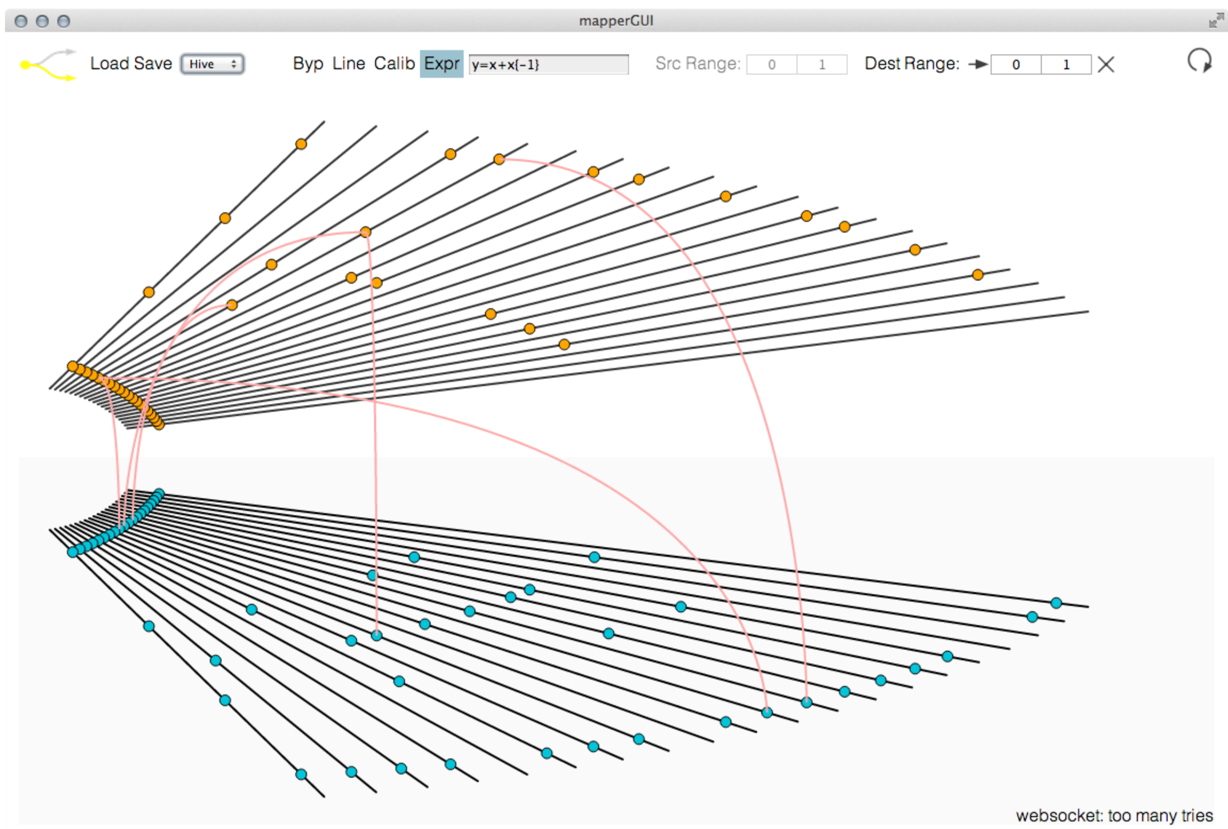


Fig. 3.7 The hive view

Chapter 4

Results & Discussion

4.1 Undoing and Redoing in a Collaborative Distributed Environment

4.2 Edge Use Cases

4.3 User Feedback

4.4 Responsiveness Testing

4.5 Modular vs Hard-Coded

4.5.1 Was the approach successful?

Are sections graphically unified? (Is this even necessary?)

4.6 Visualization vs Interaction

4.7 Unimplemented Features

1. Prefix filtering

2. Network selection

4.8 Different namespaces

Chapter 5

Conclusions & Future Work

5.1 Summary and Conclusions

5.2 Future Work

References

Audio, H. 2103. Eagen matrix. `http://www.hakenaudio.com/Continuum/eaganmatrixoverv.html`.

Baalman, M., V. de Belleval, C. L. Salter, J. Malloch, J. Thibodeau, and M. M. Wanderley. 2010. Sense/stage - low cost, open source wireless sensor infrastructure for live performance and interactive, real-time environments. In *Proc. of Linux Audio Conference*, 242–249.

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

Booth, G. 2010. Inclusive interconnections: Towards open-ended parameter-sharing for laptop ensemble. Master's thesis, University of Huddersfield, Huddersfield, England.

Bullock, J. 2008, March. Braun. Last accessed June, 19 2013, `http://sourceforge.net/projects/braun/`.

Bullock, J., D. Beattie, and J. Turner. 2011. Integra live: a new graphical user interface for live electronic music. In *Proc. of International Conference on New Interfaces for Musical Expression*, 387 – 392.

Chadabe, J. 2000, February. The electronic century part i: Beginnings, electronic musician. *Electronic Musician*: 74–90.

Cleveland, W. S., and R. McGill. 1984, September. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association,* 79 (387): 531–554.

Cook, P. R. 2009. Re-designing principles for computer music controllers: a case study of squeezevox maggie. In *Proc. of the International Conference on New Interfaces for Musical Expression*, 262–263.

Corry, M. D., T. W. Frick, and L. Hansen. 1997. User-centered design and usability testing of a web site: An illustrative case study. *Educational Technology Research and Development* 45 (4): 65–76.

Goudeseune, C. 2002. Interpolated mappings for musical instruments. *Organised Sound* 7 (2): 85–96.

Halmos, P. R. 1970. *Native Set Theory*. Springer-Verlag.

Holten, D. 2006, September/October. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphiscs* 12 (5): 741–748.

Hunt, A., and R. Kirk. 2000. Mapping strategies for musical performance. *Trends in Gestural Control of Music*.

Hunt, A., and M. M. Wanderley. 2002. Mapping parameters to synthesis engines. *Organised Sound* 7 (2): 97–108.

Hunt, A., M. M. Wanderley, and R. Kirk. 2000. Towards a model for instrumental mapping in expert musical interaction. In *Proc. of International Computer Music Conference*, 2–5.

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

Imhof, E. 1982. *Cartographic Relief Presentation*. ESRI Press.

Jensenius, A. R., T. Kvifte, and R. I. Godøy. 2006. Towards a gesture description interchange format. In *Proc. of the International Conference on New Interfaces for Musical Expression*, 176–179.

Kling, R. 1977, December. The organizational context of user-centered software designs. *MIS Quarterly* 1 (4): 41–52.

Krasner, G., and S. Pope. 1988. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming* 1 (3): 26–49.

Kvifte, T. 2008. On the description of mapping structure. *Journal of New Music Research* 37 (4): 353–362.

Mackinlay, J. 1986, April. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics* 5: 110–141.

Malloch, J., S. Sinclair, and M. M. Wanderley. 2008. A network-based framework for collaborative development and performance of digital musical instruments. *R. Kronland-Martinet, S. Ystad, and K Jensen. (Eds.): CMMR 2007, - Proc. of Computer Music Modeling and Retrieval 2007, Conference, LNCS 4969. Berlin Heidelberg: Springer-Verlag*: 401–425.

Malloch, J., S. Sinclair, and M. M. Wanderley. 2013. Distributed tools for interactive design of heterogenous signal networks. In *Multimedia Tools and Applications*.

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

Momeni, A., and C. Henry. 2006. Dynamic independent mapping layers for concurrent control of audio and video synthesis. *Computer Music Journal* 30 (1): 49–66.

Nort, D. V. 2010, January. *Modular and Adaptive Control of Sound Processing*. Ph. D. thesis, McGill University, Montreal, Canada.

Place, T., and T. Lossius. 2006. Jamoma: A modular standard for structuring patches in max. In *Proc. of International Computer Music Conference (ICMC 2006)*.

Robillard, D. 2011, January. Patchage. `http://drobilla.net/software/patchage/`.

Rudraraju, V. 2011, December. A tool for configuring mappings for musical systems using wireless sensor networks. Master's thesis, McGill University, Montreal, Canada.

Schulze, A. N. 2001. User-centered design for information professionals. *Journal of Education for Library and Information Science,* 42 (2): 116–122.

STEIM. 2004, Summer. Junxion - products of interest. *Computer Music Journal* 28 (2): 105–107.

Tuckey, J. W. 1965, April. The technical tools of statistics. *The American Statistician* 19 (2): 23–28.

Tufte, E. R. 1990. *Envisioning Information*. Graphics Press.

Tufte, E. R. 2006. *Beautiful Evidence*. Graphics Press.

Wanderley, M. M., and P. Depalle. 2004. Gestural control of sound synthesis. In *Proc. of the Institute of Electrical and Electronics Engineers*, Volume 92, 632 – 644.

Wildora. 2012, May. Osculator. `http://www.osculator.net/`.

Wolek, N. 2010. The mpg carepackage: coordinating collective improvisation in max/msp. In *Proc. of the Society for Electro-Acoustic Music in the United States*.

Wright, M., and A. Freed. 1997. Open soundcontrol: A new protocol for communicating with sound synthesizers. In *Proc. of International Computer Music Conference*, 101–104.