A Software Tool for Creating and Visualizing Mappings in Digital Musical Instruments

Jonathan Wilansky

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

August 2013

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

2013/08/13

Abstract

In acoustic instruments, physical properties of the instrument determine both the gestures that can be performed on it and the sound produced by it. This is not the case with digital musical instruments (DMIs) that consist of three distinct parts: the playable interface, the sound generating component, and the mapping between them. The implication is that mapping from sensors to sound is an integral part in the design of a DMI, and a process highly influential on how the instrument is played and sounds. Creating mappings in a DMI is a non-trivial task, and typically a lot of different mappings are explored throughout the stages of prototyping, composition, and production. Furthermore, production environments involving DMIs will typically bring together engineers, composers, and performers in valuable and often short meeting times, requiring configuration and visualization of the mapping layer to be quick and simple, catering to both technical and nontechnical participants.

This thesis presents two software components developed at the Input Devices and Music Interaction Laboratory to aid in the mapping process: *libmapper*, a software library enabling connections to be made between data signals declared on a shared network, and *Webmapper*, a list-based graphical user interface to libmapper. An extension to Webmapper was created to address the following two concerns: to provide alternate interfaces for configuring mappings, and to provide an alternate visualizations of the mapping layer. Two new interfaces were created to investigate the tasks: a grid inspired view and a *hive plot* inspired view. The system was developed using HTML5 compliant technologies, and a framework architecture inspired by the *model-view-controller* paradigm was added to Webmapper for code modularity and maintainability.

Advantages and disadvantages of the three different interfaces from Webmapper and its new extension are discussed in regards to their ability to act both as a software user interface and a data visualization tool. The contribution of the work is the demonstration of benefits to alternate methods for configuring and visualizing the mapping layer in DMIs, and the laying of a foundation for future investigations using the created HTML5 compliant software.

Résumé

Dans le monde des instruments acoustiques, les propriétés physiques des instruments déterminent les gestes effectués ainsi que les sons produits; ce n'est pas le cas avec les Instruments de Musique Numériques (IMNs) constitués trois parties distinctes: une interface de contrôle, un système de production sonore et le système de correspondance entre les deux ("mapping"). Ceci implique que l'interaction entre les capteurs et le son fait partie intégrante de l'IMN et influence fortement comment l'instrument sonne et joue. La création de correspondance dans un IMN n'est pas une tâche triviale, et généralement l'exploration de différentes correspondances est nécessaire durant les phases de prototype, composition et production. De plus, les environnement de productions incluant l'utilisation d'IMN peuvent créer des opportunités de rencontre entre les ingénieurs, les compositeurs et les interprètes car la configuration et visualisation des correspondances, qui se doit d'être simple et rapide, requiert un mélange de savoir-faire technique et non-technique.

Cette thèse est basée sur deux logiciels développés au sein du laboratoire IDMIL (Input Devices and Music Interaction Laboratory) aidant le processus de correspondance: Libmapper, une logithèque permettant de connecter des signaux numériques et Webmapper, une interface graphique pour Libmapper. Une extension de Webmapper a également été crée afin d'adresser les deux problèmes suivants: pour offrir une interface alternative de configuration de correspondance et pour offrir une visualisation alternative des couches de correspondance. Deux nouvelles interfaces ont également été crées afin d'ajouter deux nouvelles vues: une en forme de ruche et une en quadrillée. Le système a été développé en HTML5, utilisant une architecture inspiré par le patron *modèle-vue-contrôleur*.

Les avantages et les désavantages des trois différentes interfaces de Webmapper ainsi que de sa nouvelle extension seront traités vis à vis de leur capacité d'agir en tant qu'interface et en tant qu'outil de visualisation. Les contributions de ce travail résident dans la démonstration des avantages des méthodes alternatives de configuration et de visualisation des couches de correspondances dans les IMN, ainsi que dans les fondations pour de futures recherches utilisant le logiciel HTML5.

Acknowledgments

First and foremost, I am deeply grateful to Marcelo Wanderley for welcoming me to his lab, directing me to this thesis topic, and for his constant guidance amidst hectic schedules.

I was greatly inspired by my talented colleagues in the McGill Music Technology area. Thanks to Stephen Sinclair and Joseph Malloch for the development of the libmapper and Webmapper mapping tools, with a special thanks to Joseph for his generous help that began since the very beginning of my thesis.

Thanks to Aaron Krajesky for his efforts into refining Webmapper and being the number one supporter of the new interfaces. Thanks to those at the IDMIL who tested and provided feedback: Mailis Rodrigues, Håkon Knutzen, and Clayton Rosa Mamedes. Thanks to D. Andrew Stewart for feedback and your donation of mapping data, and Carolina Brum Medeiros for the inspiring discussions.

Finally, thanks to my family; Fran and Melvin–my mother and late father– for pushing me to be the best I can be; Mark, Barbara, Mitchell, and Warren for your support over the years; and Vanessa, Ariella, Cole, and now Misha, for understanding why 'Uncle Jonny' has been so busy!

Contents

1 Introduction & Motivation **1**
1.1 Project Overview 2
1.2 Thesis Overview 3
1.3 Contributions 3

2 Background **4**
2.1 Mapping 4
2.1.1 Sound Production in Acoustic Instruments 4
2.1.2 Mapping in Traditional Musical Instruments 5
2.1.3 Components of a Digital Musical Instrument 6
2.1.4 An Example Digital Musical Instrument 7
2.1.5 The Importance of Mapping in Digital Musical Instruments 9
2.2 Software Tools for Mapping in DMIs 10
2.2.1 Background on Libmapper 10
2.2.2 The Libmapper Model 11
2.2.3 The Libmapper GUI 12
2.2.4 The Vizmapper GUI 14
2.3 Summary 16

3 Design **17**
3.1 Goals of the Webmapper Extension 17
3.2 An Approach to the Representation of Data 18
3.2.1 Analysis of the Data 18
3.2.2 Analysis of the graphic system 19
3.2.3 Guidelines for the Visual Representation 20

3.2.4 Representation Using The Display . . . 22
3.2.5 Types of Constructions . . . 22
3.2.6 Analysis of the Libmapper Data . . . 23
3.3 An Approach to User Interface Design . . . 24
3.3.1 The Object-Action Interface Model . . . 24
3.3.2 The Eight Golden Rules of User Interface Design . . . 25
3.4 Summary . . . 26

4 Implementation 27
4.1 Webmapper Data Components . . . 27
4.2 Webmapper Task and User Interface Task Hierarchies . . . 28
4.3 The Grid View . . . 32
4.4 The Hive View . . . 37
4.5 Technical Implementation Details . . . 41
4.5.1 MVC and PAC Architectural Patterns . . . 41
4.5.2 Final Architecture . . . 43
4.6 Summary . . . 46

5 Discussion 48
5.1 Properties of the Data Set . . . 48
5.1.1 Size . . . 48
5.1.2 Complexity . . . 49
5.2 Evaluation Criteria . . . 50
5.2.1 Human Factors for Evaluation . . . 50
5.2.2 Evaluation of the Secondary Goal . . . 51
5.3 Evaluation . . . 51
5.3.1 Time to Learn . . . 51
5.3.2 Speed of Performance . . . 52
5.3.3 Rate of errors by users . . . 55
5.3.4 Retention over time . . . 57
5.3.5 Subjective satisfaction . . . 57
5.3.6 Ability to Visualize . . . 59
5.4 Summary . . . 62

6 Conclusions and Future Work **64**
6.1 Conclusions 64
6.2 Future Work 66
6.2.1 Enhancements to the New Interfaces 66
6.2.2 Hive Plot Node Assignment 66
6.2.3 Introducing Orderable Components 66
6.2.4 Introducing a difference utility 67

References **69**

List of Figures

2.1 A possible approach to the representation of a digital musical instrument, adapted from [1]. 7
2.2 An image of the first T-Stick prototype before being covered by the shrink wrap, adapted with permission from [2]. 8
2.3 A sample of the Webmapper GUI. 13
2.4 A depiction of the navigation in Vizmapper: (A) Shows the top-most level, with details of nested levels excluded from view; clicking the grey circle show details of the second level (B); clicking again on the grey circle will advance to the deepest level (C). 15
2.5 A screenshot of the Vizmapper graphical user interface. 15

3.1 Example use of the first six visual variables from table 3.1. 20
3.2 Example usage of the property of *connectedness* while comparing its perceptual ability to some of the visual variables from table 3.1: (a) size, (b) shape, (c) color, and (d) position. 21

4.1 A sample of the grid view interface with arbitrary data. The devices grid is on the left and it has one device pair added to the signals grid on the right. Cells of included source and destination devices have a vertical and horizontal line respectively. Blue cells correspond to links or connections, and selected cells have a red border. The image shows the visual feedback created by placing the mouse cursor over a cell in the grid. 33

4.2 A sample of the signals view mode with arbitrary data at a custom zoom level. The zoom-scroll bars provide feedback on the zoom level and position inside the grid. The image shows the user clicking and dragging a handle (highlighted orange) to adjust the zoom level. The right hand side of the upper menu bar can be used to switch between signal view presets. 35
4.3 A sample of the devices view mode with arbitrary data. The left hand side of the upper menu bar can be used to switch between view modes or to add devices into the signals grid. 36
4.4 An example of the hive view's hive plot interface with arbitrary data. The mouse cursor can be seen in the plot hovering over the source device "/Source2", causing all its connections to be highlighted. 38
4.5 An example of the Hive view's adapted hive plot interface with arbitrary data. The connection between "/Source1/Signal3" and "/Destination5/Signal1" has been selected by the user and the interface provides visual feedback: signal names are highlighted in the table, the nodes and connection line in the plot are highlighted and bolded, and the signal names are shown in the blue bar at the bottom of the interface. Additionally, the mouse cursor can be seen in the plot hovering over the source device "/Source1" causing it's connections to be highlighted. 39
4.6 Webmapper's final architecture and an example flow of interaction. 47

5.1 Example of a large and complex data set in the list view. The image shows many-to-many mappings in an arbitrary and generic set of devices. 59
5.2 Example of a large and complex data set in the grid view. The image shows many-to-many mappings in an arbitrary and generic set of devices. 60
5.3 Example of a large and complex data set in the hive view. The image shows many-to-many mappings in an arbitrary and generic set of signals. The interface highlights connections interactively based on the position of the mouse. 61
5.4 Ranking of the alternate interfaces according to each criterion. Note that ranking for subjective satisfaction was for small to medium sized data sets only. 63

6.1 Patterns of null correspondence (A), perfect correspondence (B) and non-variation of one of the dimensions (C) in a reorderable grid. 67
6.2 Diagonalization of a reorderable grid with three orderable components. The hypothetical components are represented and ordered by position in each axis and by color value. (A) shows perfect correspondence and (B) shows null correspondence. 67

List of Tables

2.1 Example signal namespaces 11

3.1 Eight visual variables of the graphic system. 19
3.2 Libmapper components for a network representation 23
3.3 Libmapper components for a digram representation 24

4.1 The Webmapper components with their levels of organization and lengths . 28
4.2 Interface Task Hierarchy A (Hive Plot) 29
4.3 Interface Task Hierarchy B (List View) 30
4.4 Interface Task Hierarchy C (Adapted Hive Plot) 31
4.5 Interface Task Hierarchy D (Grid View) 31

List of Acronyms

DMI	Digital Musical Instrument
GUI	Graphical User Interface
TUI	Text-based User Interface
UI	User Interface
IDMIL	Input Devices and Music Interaction Laboratory
MIDI	Musical Instrument Digital Interface
OAI	Object-Action Interface
OSC	Open Sound Control
MVC	Model-View-Controller
HTML	HyperText Markup Language
CSS	Cascading Style Sheet
JS	JavaScript
SVG	Scalable Vector Graphics

Chapter 1

Introduction & Motivation

Sound production in early musical instruments were dependent on the physical and mechanical properties of the instrument, and typically the instrument's playable parts were directly connected to the sound generating parts. In digital musical instruments (DMIs), there exists a separation between the instrument's playable parts (i.e., its "control surface") and the sound synthesis model it uses to generate sound [1]. This gives rise to the intermediate layer known as "mapping", an integral part in the design of a DMI requiring a strategy to map gestures on the control surface to parameters of sound. The strategy is a determinant factor in the "expressivity" of control possibilities in the instrument [3], and creating mappings is a non-trivial task [3]; altering the mapping between a control surface and sound synthesis model can change the entire character of the instrument, and can elicit different psychological and emotional responses from the user [4].

Software applications exist that facilitate the technical process of creating mappings; examples include *OSCulator*[1] and *junxion*[2]. Although these applications facilitate mappings between various multimedia devices, they are commercial closed-source products and are therefore inflexible and non-expandable. Furthermore, they were designed to handle the fundamental task of connecting a small number of multimedia devices, and lack functionality designed for inquiries into the mapping layer and for representing large data sets; in a research environment where DMIs are constantly being explored, there is a greater need to experiment with mappings of various sizes and complexities, and in investigative ways.

[1] http://www.osculator.net/
[2] http://steim.org/product/junxion/

At the Input Devices for Music Interaction Laboratory (IDMIL), and part of the ongoing mapper tools project [5], the software solutions *libmapper* and *Webmaper* were conceived as open-source prototyping tools for mapping DMI control surfaces to sound generating devices. These tools have been used in various contexts ranging from the prototyping of smaller-scale individual projects to larger-scale collaborations amongst researchers from various fields. An example larger-scale project was the production of "Les Gestes" [6], where engineers, composers, and performers were participating in production workshops and experimenting with the mappings for a network of DMIs. Typically during a workshop, a large number of mapping configurations were created and explored. Due to the complexity of the mappings and the real-time nature of the experimentation, participants from various technical backgrounds were confronted with the need to interact with the mapping layer and modify large numbers of connections. All the aforementioned software solutions facilitate the creation of mappings, however, their methods of interaction become challenged when dealing with large sets of data, and they lack functionality designed specifically for visualization of the mapping layer.

This master's thesis addresses the shortcomings of the previous tools in attempts to provide an open-source software solution that can be used in small to large scale projects. Principles of software user interface design and data visualization provide methodological strategies to overcome issues encountered when interacting with software and representing data. This thesis project intends to exploit these principles and expand on the tools previously developed at the IDMIL, aiming to provide alternative and effective means to interact with mappings in DMIs.

1.1 Project Overview

The focus of this project is the enhancement of an existing software system for configuring mappings in a DMI environment. The project builds on previously created tools to extend them with two new user interfaces that were designed using a combination of principles taken from the domains of user interface design and data visualization. The project aims to provide alternative means configure and visualize the mapping layer in DMIs, and to overcome shortcomings of the existing tools when dealing with large data sets. The project investigates the potential for alternate representations of mapping data to interact with and gain insight into the mapping layers of DMIs. Finally, the project aims to organize

and enhance the existing software's architecture in order to create a codebase that is simple to maintain and easy to expand.

1.2 Thesis Overview

Chapter 2 traces the origins of mappings from traditional to digital musical instruments, and emphasizes the important influence that mapping has on a DMI; it also introduces the existing software tools for creating mappings that were extended as part of the thesis project. Chapter 3 details the principles of data visualization and user interface design applied to the new interfaces. Chapter 4 provides the implementation details of the two new interfaces that were created as part of software's extension; it also describes the technical details of the enhanced software architecture. Chapter 5 explains relevant criteria for evaluation of user interfaces, and presents a thorough evaluation of all three interfaces included in the software. An in depth discussion points to strengths and weaknesses of each interface. Chapter 6 presents conclusions, points to future work that will continue after this thesis, and suggests various directions that future research can take.

1.3 Contributions

The contributions of this thesis work are the exploration of issues relevant to creating mappings between networks of DMIs; the creation of two alternate visual representations of the mapping layer in DMIs; the design and implementation of an expandable software system for alternate representations of mappings; and this thesis document that describes and explores the research in detail.

Chapter 2

Background

This chapter is divided into two main sections. Section 2.1 traces the origins of mappings from traditional to digital musical instruments, and emphasizes the important influence that mapping strategies have in DMIs. Section 2.2 introduces the existing software tools for mapping.

2.1 Mapping

2.1.1 Sound Production in Acoustic Instruments

In acoustic instruments, mechanical and physical properties of the instrument determine the sound produced by the instrument. For example, a drummer's hand strikes the drum's membrane causing it to vibrate, and a sound is created determined by the physical properties of the head and shell; a flutist blows a stream of air directed across a hole in the instrument, exciting the air in its resonant cavity, and a sound is created determined by the physical properties of the flute body. There are many sources available describing the physics of the sound produced by musical instruments, along with detailed mathematics for quantifying the behavior. The topic is beyond the scope of this thesis, but the following is a non-detailed introduction adapted from [7]. It is presented here to note the direct relationship between the instrument's playable parts and the sound produced.

In the case of an acoustic guitar, the mechanical vibrations of the strings are the sources of sound in the instrument. The guitarist will pluck a string, causing it to be displaced; the string then creates a restoring force causing it to return to and overshoot its equilibrium

position, sending it into an oscillatory motion. The oscillatory motion resembles a mass and spring following simple harmonic motion, and it is the source of sound generated by the guitar [7].

In terms of energy, as the string oscillates it alternates between potential and kinetic energy, losing energy as it gradually returns to its equilibrium position. Like in most acoustic instruments, sound production actually depends upon the collective behaviour of several vibrators (e.g., the body of the guitar), which may be weakly or strongly coupled together. This causes the instrument as a whole to behave as a complex vibrating system. It is this mechanical nature of the instrument's sound and the musician's ability to excite these mechanisms that result in music; energy is transferred from the guitarist to the string, and the string is the source of sound [7].

2.1.2 Mapping in Traditional Musical Instruments

As the design of certain musical instruments evolved to become more complex, a new degree of separation developed between the instrument's playable parts and its sound producing mechanisms. Components were added to enhance playing techniques, thereby introducing an intermediate layer between the physical actions that excite the instrument and the resulting sound. The following two examples are presented in [4] and demonstrate this separation.

The first example is that of the piano, where the pianist excites the embedded strings only indirectly through the piano's keyboard mechanism; a finger pushes a piano key, and the piano key causes a hammer to strike the string. As the pianist's energy is transferred from his body and through the key to the string, he is indirectly contributing to the sound. The keys act as an interface to the sound generating mechanism, affecting it through other embedded mechanical procedures. This defines a mapping strategy from the interface to sound; a simple one-to-one mapping of a key to a string. The musician's actions, or musical "gestures", are therefore only indirectly connected to the production of sound through the keyboard interface and the mapping strategy. In this way, the musician has less control over the timbre of the resulting sound, but other mechanisms can be introduced to modify the parameters of sound, like the piano pedals controlled by the feet.

The second example describes elements of a pipe organ that add an even further degree of separation between the interface and sound. The pipe organ consists of a set of pipes that

produce sound when air travels through them. However, unlike some other instruments that use a column of air to generate sound (e.g., flute, clarinet), the organ player does not supply the air that generates the sound; the energy and mechanism supplying the air comes historically from at least one other person operating the bellows, or nowadays with electricity. The player controls the flow air by using one or many keyboards and a pedalboard; the keys and pedals operate valves that control the flow of air through the pipes, affecting musical parameters like pitch and timbre. The use of an external energy supply in combination with the often complex routing of keys and pedals to pipes separates the musician's gestures from the resulting sound through an interface and a mapping strategy.

2.1.3 Components of a Digital Musical Instrument

A digital musical instrument (DMI) contains three parts[1]:

1. *the sound generating component*: a DMI generates sound by means of digital sound synthesis. The signal generated by the sound synthesis model is made audible by sending it to an electromechanical device (e.g., loudspeakers) that converts the digital signal into mechanical vibrations in the air.

2. *the playable interface*: the DMI provides a means to control the parameters of the synthesis engine in real-time; this is referred to as the "control surface", "controller", or "(hardware) interface" and it defines the set of gestures that can be performed on the instrument. It is typically composed of sensors capable of detecting various kinds of user input.

3. *the mapping*: a mapping strategy is then needed to map gestures from the control surface to the parameters of the sound synthesis model.

A diagram of the three components is shown in figure 2.1. When creating a DMI, the designer is faced with a vast array of possibilities for designing each component of the instrument. The designer can be inspired to replicate traditional acoustic instruments, enhance traditional instruments, or create entirely new designs altogether. The DMI can be an object to hold, an attachment to the body, or part of the room itself (e.g., interactive floor, motion capture). In each case, details for each part of the instrument are carefully selected.

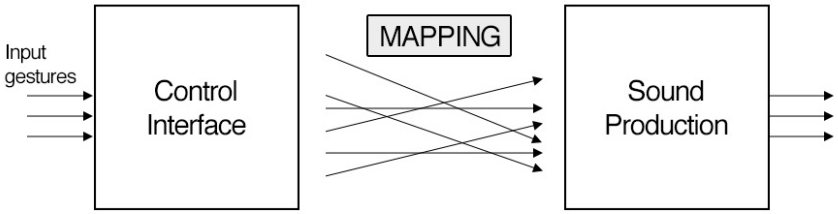


Fig. 2.1 A possible approach to the representation of a digital musical instrument, adapted from [1].

Sound synthesis models are able to simulate acoustic sounds or generate entirely new sounds. They can trigger pre-recorded samples or MIDI[1] notes, use digital signal processing to alter pre-recorded sounds or sounds captured through a microphone in real-time, or generate new sounds entirely through sound synthesis models.

For gesture acquisition from the control surface, numerous types of sensors are readily available and affordable. A review of sensors that are commonly used in musical applications can be found in [1], and they include force-sensitive resistors, accelerometers, bend sensors, piezoelectric sensors, infrared sensors, ultrasonic sensors, and more. Sensors facilitate the acquisition of many kinds of gestures or measurements from the user, such as: the pressing of a button; the swiping of a surface; distances to a part of the body or another object; the motion or acceleration of the instrument; the bending of a limb; tracking of the entire body; and more.

Finally, a mapping strategy is required between the control surface and sound generating model. This determines how the playable interface and the corresponding gestures will correlate to parameters of the sound generating component. The mapping defines what sensors and gestures control what parts of the sound and how.

2.1.4 An Example Digital Musical Instrument

As a practical example, I refer to the T-Stick, a family of DMIs developed at the IDMIL by Joseph Malloch [8]. The T-Stick is among the few new DMIs that have surpassed the prototyping stage to emerge with a standardized definition for the family of instrument. It has been used by a reasonable number of artists and researchers in the nearby music

[1] a technical standard for communication between electronic musical instruments, computers and other related devices

technology community. It is therefore a good example with sufficient documentation, with access to substantial mapping data that has been explored during the development of the thesis project.

A T-Stick is constructed with a cylindrical tube of ABS plumbing pipe whose length determines its classification in the T-Stick family. Sensors and electronics are placed inside and on the surface of the pipe, which is then sealed with shrink wrap plastic. This method provides a structure and reinforcement that makes the instrument durable and water resistant. A photo of a T-Stick is shown in figure 2.2.

Fig. 2.2 An image of the first T-Stick prototype before being covered by the shrink wrap, adapted with permission from [2].

Its specifications of sensors enables the T-Stick to respond to a variety of different gestures. One side of the pipe is covered with numerous capacitive sensors (48 in the first prototype). The other side is equipped with one or two pressure sensors that span most of its length. There is an accelerometer at each end of the pipe and a piezoelectric sensor in the middle. The combination of these sensors means that the T-Stick has a low-resolution multi-touch position sensor covering the length of the pipe, capable of providing the center, width, and velocity of areas touched along its length, coupled with a pressure sensitive surface on the opposing side. It can sense movement, acceleration, or tilting of the pipe, along with deformations such a bending or twisting. Higher-level gestures, such as jabbing, smacking or swiping, can also be inferred from analysis of the raw sensor data,

These specifications enable the T-Stick to be an interesting interface for musical performance. At the time the article was written in 2006, the T-Stick appeared in four concerts and numerous demonstrations, including hundreds of collective practice hours by users. Pieces have been written for it by student composers, including D. Andrew Stewart[9] whose mappings were made available for the thesis project. Example performances using the T-Stick can be seen on the IDMIL website[2].

[2] `http://www.idmil.org/projects/the_t-stick`

2.1.5 The Importance of Mapping in Digital Musical Instruments

While the mapping layer is inherently defined in most acoustic instruments by the physical and mechanical characteristics of the instrument, it is not inherently defined in the case of DMIs with control interfaces that are distinctly detached from the sound generating mechanisms. The implication is that mapping from sensors to sound is an integral part in the design of a DMI, and a process highly influential on how the instrument is played and sounds. The mapping layer can therefore be seen as a link between technical hardware and expressive sound, and the common ground shared between the designers of the instrument and the designers of the music.

Experiments were described by Hunt, Wanderley and Paradis [10] that explore effects of the mapping layer; the setups described consisted of fixed synthesis models and control surfaces that were investigated through alternate mapping strategies. Results showed that a simple "one-to-one" mappings to parameters of sound were not very stimulating for the test users, whereas more complex mappings enabled more expressive results. Additionally, more complex mappings entailed a learning curve that was observed to be more appealing to the test users. The experiments demonstrate that by changing only the mapping layer in a DMI, the quality of the instrument is changed in regards to its control and expressive capabilities, affecting the behaviour of the instrument and its feel to the performer.

The non-triviality of mapping in DMIs was already presented in 1997 in [3] and the work focused on the influence of the mapping strategy in the context of musical expression. It notes the example of the additive synthesis model that "has the power to virtually synthesize any sound, but is limited by the difficulty encountered in simultaneously controlling hundreds of time varying control parameters". Mapping strategies are commonly categorized into a small number of groups that are determined by the number of correspondences between the gestures and parameters of sound ([3], [4]). Ng points out how different mapping strategies have different usages in different contexts, and the complex relationships that can arise with multilayered mapping strategies [11].

2.2 Software Tools for Mapping in DMIs

Although commercial software applications that facilitate mappings between multimedia devices exist (e.g., *OSCulator*[3] and *junxion*[4]), they are closed-source, platform-dependent products, lacking functionality designed for inquiries into the mapping layer and for representing large data sets. In a research environment where DMIs are constantly being explored, there is a greater need to experiment with mappings of varying sizes and complexities, and in investigative ways; this section presents mapping tools relevant to the thesis project that were developed at the IDMIL as collaborative and flexible open-source alternatives to the pre-existing tools.

2.2.1 Background on Libmapper

Since defining the mapping layer is a non-trivial task (see section 2.1.5), and because it is unlikely that there can ever be a single 'best' mapping strategy, the process of defining the mappings in DMIs becomes one of exploration, experimentation, and trial and error. At the IDMIL, and part of the ongoing mapper tools project[5], a software solution called Libmapper[6] was conceived as a prototyping tool for mapping DMI interfaces to sound generating devices.

We can trace the origins of Libmapper to the McGill Digital Orchestra project[7], a project that brought together researchers from different fields, working collaboratively in developing tools for live performance with digital technologies. As described in [5], the work involved collaborations between engineers, composers and performers in all stages of production from building new DMIs, to composing the musical pieces that would be performed. Because these tasks were occurring simultaneously, researchers found themselves in need of tools to optimize the workflow during valuable and often short meeting time. Furthermore, composers and performers who were not experienced in the technical aspect of the DMIs were expected to be involved in the experimentation and decision-making process with regards to how the instruments would be played and sound. Researchers therefore needed a new, easy, and quick way to modify the mappings of their instruments in a way

[3] http://www.osculator.net/
[4] http://steim.org/product/junxion/
[5] `http://www.idmil.org/projects/mappingtools`
[6] `http://libmapper.github.io/`
[7] `http://www.music.mcgill.ca/musictech/digitalorchestra/`

that catered to both technical and non-technical participants.

The result of this situation was the creation of a software plug-and-play "orchestral neighborhood" where instruments and synthesizers could announce their presence on the network and be made available for connections. In addition, a graphical user interface (GUI) was created as the visual means to display the devices available on the network, for making the connections between them, and for modifying connection properties.

2.2.2 The Libmapper Model

This section describes the structure of the libmapper's data, and terminology will be addressed to avoid confusion with the various terms that have been applied to the same concepts in different contexts.

In mathematics, mapping refers to a rule that assigns elements from one set, called its *domain*, to another set, called its *range* [12]. The number of elements in a set is referred to as its *cardinality*.

The Libmapper network consists of a set of devices, each with its own set of signals. Libmapper formats messages similarly to the Open Sound Control (OSC) protocol[13], and Devices and signals are identified by their announced OSC namespace. Namespaces are inherently hierarchical with forward slashes separating levels in the hierarchy. Libmapper places no restrictions on the length or depth of a device or signal namespace. In this paper, the terms "name" and "namespace" will be used interchangeably when applied to the identification of a device or signal. Signals have a direction and are either an input or an output signal.

Namespace	**Description**
tstick.1/raw/accelerometer/1/x	the data stream of T-Stick number one's accelerometer values in the x dimension
tstick.2/cooked/accelerometer/2/amplitude	the data stream of T-Stick number two's combined accelerometer data

Table 2.1 Example signal namespaces

Output signals contain a stream of data pertaining to a sensor on the DMI's control surface, and will be referred to as *source* signals. A device containing at least one source signal can be referred to as a *source device*. Table 2.1 lists two example signals, demonstrating that signals can represent raw data sensor values or higher-level "cooked" values

with semantic meaning.

Parameters of the sound synthesis engine that are controlled by another stream of data (i.e., a source signal) will be referred to as *destination* signals, and a device containing at least one destination signal will be referred to as a *destination device*.

The terminology of sources and destinations was chosen for the remainder of this thesis as it is appropriate in context of mapping "something to something". Note that a device can contain both source and destination signals, and can therefore be considered both a source and destination device.

Libmapper is organized into a two-tiered structure. Before signals of any device can be mapped to each other, their corresponding devices must first be mapped. A potential mapping between devices is called a *link*, and a mapping between signals is called a *connection*. A connection implies a routing from the source signal's data stream to the destination signal's input parameter. Libmapper currently supports *one-to-one* or *one-to-many* mappings.

A basic connection has no signal processing applied to the stream of data, but functions can be used to modify the signal's data before it is routed to the destination. We will refer to these signal processing functions as *transformations* or *transfer functions*.

In computer science, a *map* is a data structure that stores elements based on key-value pairs[14], and we will borrow this term to describe the entire set of connections, links, and transfer functions on a given libmapper network.

2.2.3 The Libmapper GUI

The Libmapper GUI provides the interface through which the user is able configure the mappings for devices that have been announced on the network. Originally developed for Max/MSP[8], Stephen Sinclair, a developer of Libmapper and Ph. D. student of the IDMIL, developed the interface known as "Webmapper" that enables a web browser to communicate with the Libmapper network through a Python[9] based server. Aaron Krajeski, a Master's student of the IDMIL, has continued the development and refinement of the Webmapper GUI as part of his thesis. A sample of the current interface can be seen in figure 2.3. This section describes the interface and typical workflow when using Webmapper.

[8] `http://cycling74.com/products/max/`
[9] `http://www.python.org/`

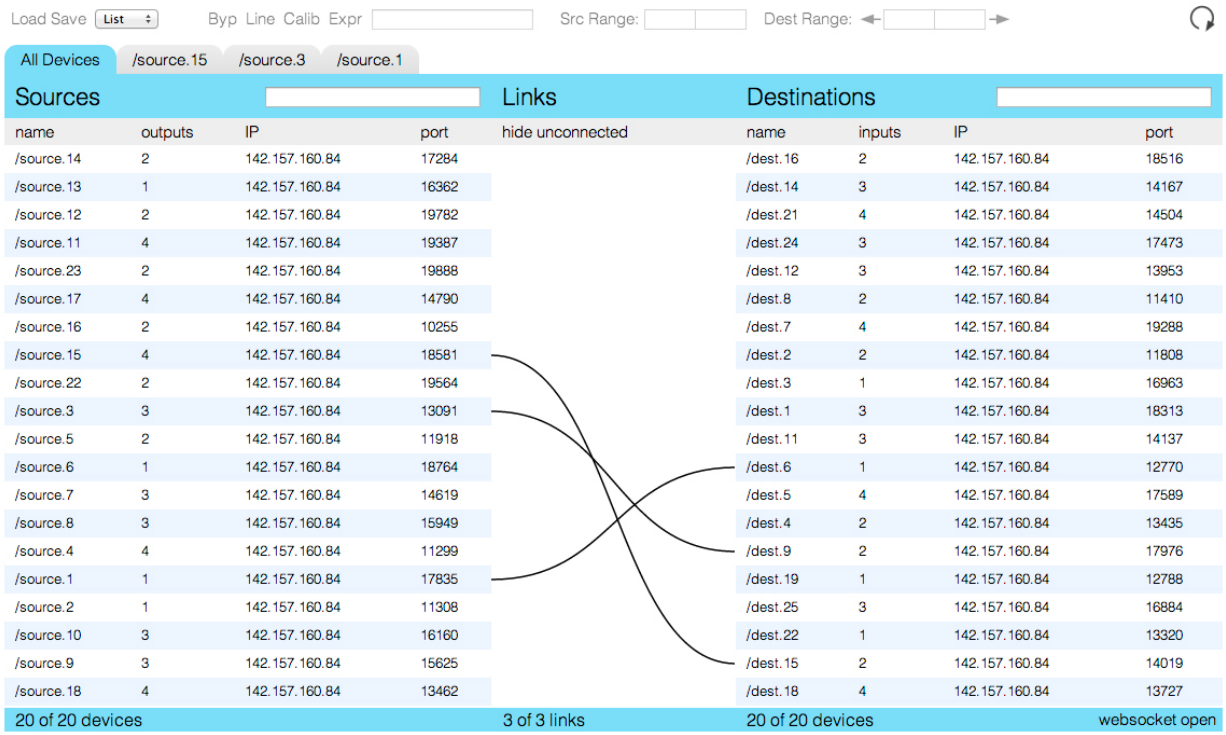


Fig. 2.3 A sample of the Webmapper GUI.

Device namespaces are arranged into two tables on screen, with source devices on the left hand side and destination devices on the right. A device that is both a source and destination will be listed in both sides. Each row corresponds to a single device, displaying its name, number of input or output signals, IP address, and port number. The design resembles a bipartite graph with each table representing a set whose elements can be mapped to the other's.

In accordance with libmapper's two-tiered structure described in section 2.2.2, a link is created by clicking and dragging a row from one of either tables to the other. The link is represented on screen with a curved line connecting the two rows, and a new tab is created containing the signals of each device. The interface becomes organized by tabs, with the leftmost tab always displaying the devices on the network, and subsequent tabs displaying signals of each linked device. Note that a tab is created for every one-to-many device mappings, meaning there will be one tab for each linked source device and it contains signals of the source device and signals of all linked destination devices.

A tab containing signals behaves in the same way as the devices tab, except that rows

correspond to signals. Dragging a signal from one table's row to the other's creates a connection between them, telling libmapper to route the source signal's data stream to the destination. Each row displays the signal-specific properties:

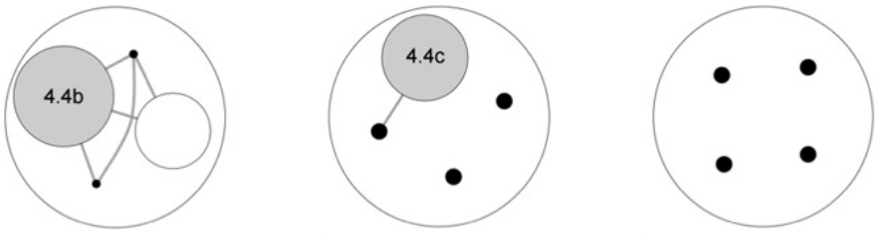
- the signal's name, data type (e.g., integer, floating point number, etc.)
- unit type (e.g., Hertz, centimeter, etc.)
- its minimum and maximum values

When a connection is made, the default transfer function is set to "bypass", where no mathematical operation is performed on the the data (i.e., the function f(x)=x). To modify connection parameters (e.g., transfer functions) the user clicks on the line representing the connection; the GUI then highlights the connection in red and its parameters can be set using the 'edit bar' at the top of the interface. Parameters include scaling, clipping, or setting an expression that will be applied to the signal's data. A connection is deleted using the "delete" keyboard shortcut.

2.2.4 The Vizmapper GUI

The GUI to Libmapper was intended as a tool to configure mappings in a simple and intuitive way. Vijay Rudraraju, for his Master's thesis [15], worked on an extension to the GUI called "Vizmapper" that uses principles of data visualization to represent the network in a different way. Although not currently part of the Webmapper software, Vizmapper demonstrated that alternate interfaces can provide new and interesting ways to interact with the mapping layer.

Vizmapper was inspired by a method to represent hierarchical data called the balloon tree visualization [16]. In a balloon tree, nodes are positioned and grouped using enclosing circles that differentiate between different levels in the hierarchy. Curved lines are used to represent adjacency relations between terminal elements (black dots), and straight lines represent inclusion relations between intermediate elements (grey dots) and different levels of the hierarchy. Since both the straight lines and enclosing circles represent inclusion relations, Vizmapper removes the straight lines in attempts to reduce clutter, thereby creating a modified balloon tree visualization. Furthermore, Vizmapper provides an interactive hierarchical interface through which the user can navigate between nested branches in the hierarchy. Figure 2.4 describes this navigation.



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

Fig. 2.4 A depiction of the navigation in Vizmapper: (A) Shows the topmost level, with details of nested levels excluded from view; clicking the grey circle show details of the second level (B); clicking again on the grey circle will advance to the deepest level (C).

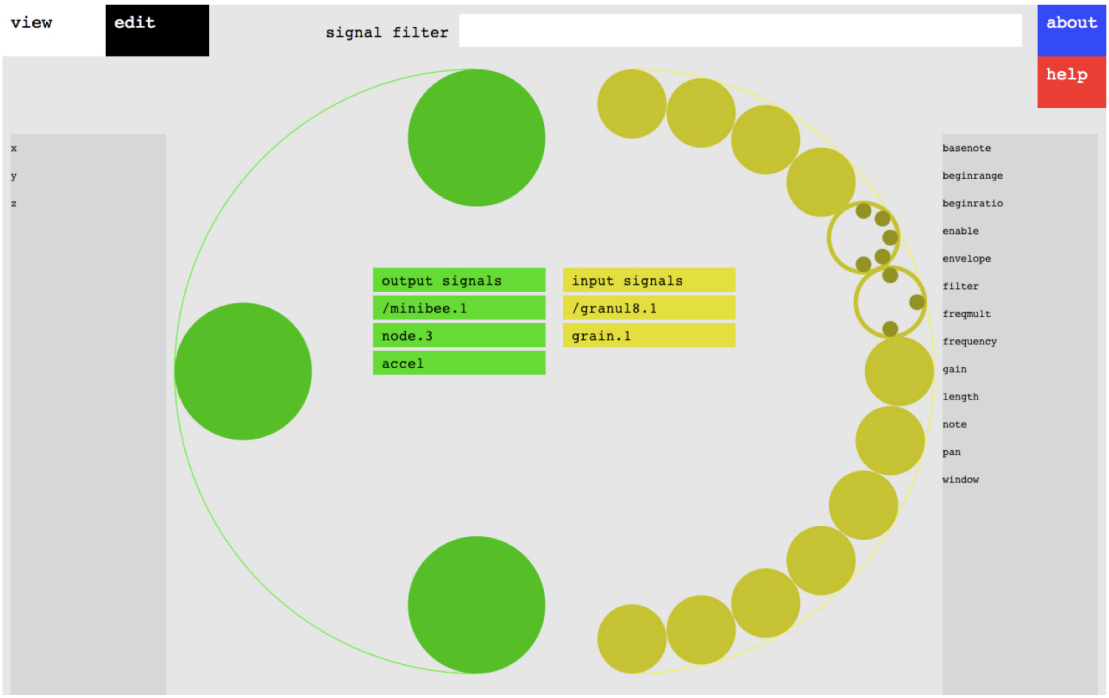


Fig. 2.5 A screenshot of the Vizmapper graphical user interface.

In the actual Vizmapper interface (figure 2.5), the dots represent signals, with source and destination signals constrained to the left and right portion of the screen respectively. Signals are grouped by enclosing circles according to the hierarchy defined by the signal's namespace (ex: "body/arm/hand/finger/index"). A filled circle represents a terminal signal (one that can be connected), and a circle with no fill represents a signal at an in-

termediate level in the hierarchy. Clicking on an unfilled circle will traverse deeper to the next level in the corresponding signal's namespace. For reverse traversal, two columns are provided in the center of the screen that display a trace of the path taken in namespace hierarchy, and clicking on a button in a column will return the visualization to a previous level in the hierarchy.

The interface is divided into two modes, called *view* and *edit*. The former is used to navigate the hierarchy and place the desired source and destination signals into view. The latter is used to select the signals and create the connection between them, using the transfer function if one was supplied.

2.3 Summary

In this chapter, the background context for the thesis project was provided and terminology was addressed. Mapping is presented as an important part in the design digital musical instruments, and software tools were presented that facilitate the configuration of mappings between digital instruments and sound generating devices. Two graphical user interfaces to the software mapping tools developed at IDMIL were presented as different methods to configure and visualize the mappings in a digital musical instrument network.

Chapter 3

Design

The previous chapter introduced the software tools for mapping that would be extended as part of the thesis project. This chapter introduces the goals of the extension and the design principles guiding its development. Section 3.1 lists the motivations for the project. The relevant data visualization and user interface design principles are introduced in Section 3.2 and Section 3.3 respectively.

3.1 Goals of the Webmapper Extension

The Libmapper and Webmapper software solutions address the need to create and modify mappings in a simple and intuitive way. Additionally, Vizmapper has demonstrated that principles of data visualization can be used to offer alternate means to interact with and represent the mapping layer. After significant experience in using both the Webmapper and the Vizmapper GUI, the developers of Libmapper have come to believe that there is no single 'best' user interface, and that different interfaces and network representations could be useful to different users in different situations.

The tool I developed is an enhancement to the Webmapper software, and it will be referred to in this chapter as the "Webmapper extension". Its concern is threefold:

1. to provide an alternate interface for configuring mappings
2. to provide an alternate visualization of the mappings
3. to provide a means for comparing different mappings to each other

The focus of this thesis is on the first two concerns. Development of the extension was guided by principles of user interface (UI) design and data visualization that are presented in the following sections, and two alternate interfaces were created as part of the thesis project.

3.2 An Approach to the Representation of Data

Bertin's "Semiology of Graphics" [17] is often cited as the first to provide a methodology to the visual representation of data. This seminal work was acclaimed to be ahead of its time, and it became a source of inspiration to other dominant contributors in the field. Bertin has become a recognized authority in the study of graphics, and his work is still relevant today. Some of his principles are presented here as a basis for an informed methodology when representing the mappings in the Webmapper extension. They are presented in two parts: analysis of the data (Section 3.2.1) and analysis of the graphic system (Section 3.2.2).

3.2.1 Analysis of the Data

Analysis of the data involves breaking down its information into what Bertin calls the *invariant* and one or more *components*. The invariant is the complete and invariable notion common to all aspects of the data, and the components are the variational concepts. For example, in a hypothetical graph depicting the daily average temperatures for July 2013, the invariant is "the average temperature on a given day", and the components are "average temperature" and "a given day".

The length of a component represents its number of possible values; a *short* component has length of less than five, and a *long* component has greater than fifteen. The notion of length does not apply to continuous variables with a series of infinitely divisible numbers.

Bertin describes three levels of organization or levels to categorize the components, and they are ordered by metaphor of height. They are presented in the following list from low to high:

1. *qualitative* (or *nominal*)
2. *ordered*
3. *quantitative*

Qualitative components are at the lowest level and are characterized by elements that cannot be ordered in a universal manner. For example, "types of trees" are qualitative because there is no universally accepted way to order different species of trees.

Ordered components are at the middle level and are characterized by having a universally accepted ordering with equidistant elements. An example is a classification of size into three groups: "small", "medium", and "large". There is a universally accepted ordering of size from small to large and reordering the middle category would impart a source of confusion in the representation of the range. Additionally, classifying elements into qualitative groups of size defines equidistant elements because there are no measurable units for the comparison.

Quantitative components are at the highest level and are characterized by components with countable units and measurable ratios. "Distance in meters" is an example of a quantitative component because the distance is measured in meters and ratios can be determined from it.

3.2.2 Analysis of the graphic system

Analysis of the graphic system provides a methodical means to overcome practical problems encountered when representing data; it is through the appropriate association of the properties of the data with properties of the graphic system that make for a systematic representation. Bertin describes the eight *visual variables* listed in table 3.1 that can be utilized in the representation of data. The first six are exemplified in figure 3.1

1. size
2. value
3. texture
4. color
5. orientation
6. shape
7. horizontal position in the display
8. vertical position in the display

Table 3.1 Eight visual variables of the graphic system.

Similarly to properties of the data, each visual variable has a *length* determined by the

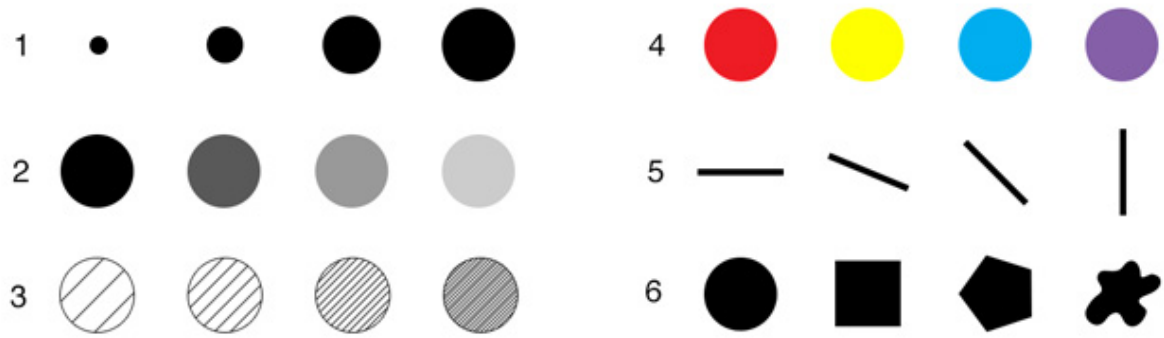


Fig. 3.1 Example use of the first six visual variables from table 3.1.

number of different steps that can be perceived as dissimilar. Also like properties of the data, each visual variable can be characterized by a *level of organization*:

- A variable is *selective* when it allows for the immediate isolation of any graphic elements sharing the same quality of the variable.
- A variable is *associative* when it allows for the immediate grouping of all graphics elements differentiated by the variable.
- A variable is *ordered* when the visual ordering of its variations is immediate and universal.
- A variable is *quantitative* when its variations can be immediately identified and expressed by a numerical ratio.

Bertin's list of visual variables does not include the property of *connectedness*: lines connecting different graphical elements. Ware reminds us that this property is perceptually more dominant than size, shape, color, and proximity [18] as shown in figure 3.2. When using lines to connect elements, smooth continuous lines are preferable to lines with abrupt changes in direction because they are easier for the eye to follow.

3.2.3 Guidelines for the Visual Representation

Bertin prescribes the following principles as a set of guidelines to follow when choosing a visual variable to represent a given component:

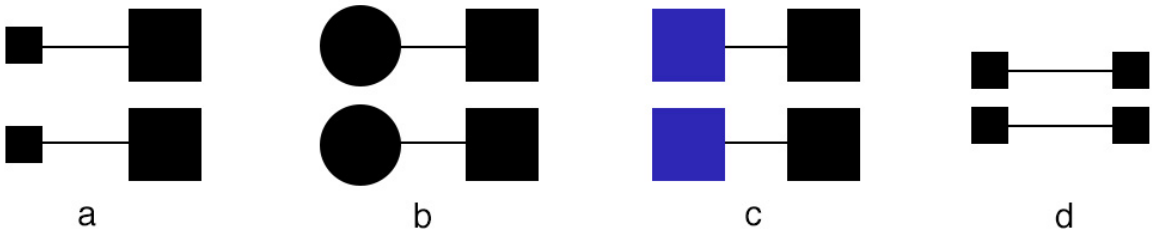


Fig. 3.2 Example usage of the property of *connectedness* while comparing its perceptual ability to some of the visual variables from table 3.1: (a) size, (b) shape, (c) color, and (d) position.

1. The number of visual variables used to represent a set of data must be equal to or greater than the number of components of the data.
2. The variable chosen to represent a component must have a length equal to or greater than the length of the component
3. The variable must have a level of organization equal to or higher than the level of the component.

All of the components in Webmapper's data are qualitative (see Section 4.1). Since qualitative components are equidistant, each element has equal importance and the graphic representation should not disturb this notion by highlighting any particular element. Since qualitative components have no universally acknowledge order, they can be reordered arbitrarily for the purposes of information processing. Two perceptual approaches can be used when representing qualitative components:

1. A *selective* approach is engendered by questions of an elementary or intermediate reading level, and it enables us to find a specific element (e.g., a specific device). When questions relating this level of reading are pertinent, it is important that the component be represented by a selective variable.
2. An *associative* approach is engendered by questions of intermediate or overall reading level, and it enables us to group a set of elements (e.g., all source devices). When questions relating this level of reading are pertinent, it is important that the component be represented by an associative variable.

3.2.4 Representation Using The Display

Position in both dimensions of the display (i.e., visual variables numbers 7 and 8 from table 3.1) are the most robust graphical elements and can represent any component of the data. Position has the longest length[1], with the number of divisions limited by the capabilities of the graphic rendering system and the viewer's perceptual ability. Tufte [19] points out that 625 different points can easily be perceived in one square inch. Furthermore, all four levels of organization can be perceived in the display:

1. Selective: graphic elements differing in position can be isolated and seen as different
2. Associative: graphic elements sharing an area in the display can be grouped together
3. Ordered: The position of graphic elements along either axis will be perceived by an immediate order; left to right, top to bottom, or degree of orientation around a fixed point.
4. Quantitative: the display can be divided into units of measurable size that allow the perception of ratios of lengths, angles, or areas between different graphic elements.

Ware [18] reminds us that a powerful way to emphasize relationships between different data entities is to use the gestalt principle of *proximity*. The principle explains that graphical elements in close proximity are perceptually grouped together, and elements not in close proximity are seen as distinct, confirming the notions of selective and associative in the display.

3.2.5 Types of Constructions

Bertin divides graphical representations, or *constructions* into four groups. Two of these groups, *diagrams* and *networks* can be applied to Webmapper's data (Section 3.2.6) and are introduced here.

A **diagram** can be constructed when the correspondences in the data are established between elements of one component and elements of another component. To construct a diagram, first the decision is made on how each component will be represented, and then the correspondences are recorded.

[1] shape actually has the longest length but not mentioned here because the number shapes that can be perceived as different is significantly reduced at smaller sizes

A **network** can be constructed when the correspondences in the data are established between elements of a single component. The order of steps to construct a network is opposite to the diagram; first the correspondences are recorded and then a representation is deduced that will produce the simplest structure.

Note that a network can also be represented as a diagram by simply doubling the component through which the correspondences are made; correspondences are then made between the original component and its copy.

3.2.6 Analysis of the Libmapper Data

This section categorizes libmapper's underlying data structure based on Bertin's data visualization principles presented in section 3.2.1. This categorization is the basis for the metaphors that were used to create the interface objects and actions described in Section 4.2. Several categorizations are possible and presented here.

The most fundamental task facilitated by libmapper is the creation of connections between signals. This translates simply into the notion of:

- **the invariant**: "a mapping between a source signal and a destination signal".

Libmapper does not differentiate between the different concepts of a source and destination device or signal in the data structures containing the device and signal objects. As explained in Section 2.2.2, a device can contain both source and destination signals, making it both a source and destination device. Therefore, a direct translation of libmapper's data structure into components for representation is shown in table 3.2. Using this classification of components, correspondences of the invariant are made between all elements of a single component (i.e., signals) and a network representation can be constructed.

1. devices
2. signals
3. links between devices
4. connections between signals
5. transformations applied to connections

Table 3.2 Libmapper components for a network representation

As explained in Section 3.2.5, a diagram can be constructed from any network by simply doubling the component through which correspondents of the invariant are made. In this case, a better approach to creating a diagram would be to differentiate between the notion of sources and destinations; by doing so, irrelevant elements (i.e., those that cannot be mapped) are removed from the sets. Table 3.3 lists the components for representation according to this differentiation. Using this categorization, correspondences are made between two different components and a diagram can be constructed. This is the approach currently taken by all the constructions in Webmapper and its extension because when faced with the task of mapping a source signal to a destination parameter, it becomes more intuitive and efficient to separate the notion of sources and destinations in the representation. This is not a requirement, and additional constructions can use a network representation if desired.

1. source devices
2. destination devices
3. source signals
4. destination signals
5. links between devices
6. connections between signals
7. transformations applied to connections

Table 3.3 Libmapper components for a digram representation

It should be noted that because the presence of a link can be inferred through the presence of a connection, the construction does not necessarily need an explicit representation for links between devices. We will see in Section 4.4 that the hive plot representation does not represent links explicitly.

3.3 An Approach to User Interface Design

3.3.1 The Object-Action Interface Model

Graphical user interfaces provide an alternative to text-based user interfaces (TUIs) through visual representations of the commands that can be performed by the application. Actions that were performed in TUIs through commands typed into the command line are replaced

by the manipulation of GUI objects and their corresponding actions. The theory of design called the *object-action interface* (OAI) model [20] provides insight into the processes that can occur for users to be successful in using a GUI to accomplish a task. It describes how users accomplish tasks facilitated by the software through the linkage of the *task domain* with the *interface domain*.

The task domain object and actions correspond to the "real-world" objects or concepts that the user works with to accomplish their intentions, and the actions that can be applied to them. The interface domain object and actions are the visual representation of the task objects inside the GUI, and the actions that can be applied to them by the software.

Performance of the tasks becomes more intuitive if a clever visual representation of the real world objects and actions are provided because it creates metaphors for the manipulation of familiar objects. An example is the "desktop" metaphor of modern operating systems like Microsoft Windows, or file structuring metaphors that organize digital "files" into "folders". Interface objects and actions can be more easily understood and remembered when they have a logical structure that is anchored to familiar task objects and actions. Because the syntactic details of textual commands in TUIs are de-emphasized or non-existent in GUIs, users familiar with the task domain can learn the interface relatively easily through demonstrations, explanations of features, or simply by trial-and-error.

The OAI model emphasizes the need for the assessment of user profiles and recognizes the potential for diversity among users. Users can vary in their social, cultural or technical background; their physical, cognitive or perceptual abilities; their level of expertise in the task domain; their familiarity with the interface domain (i.e., experience and familiarity with the use of a computer); their frequency of use of the software; and more. Assessment of the user profiles will help the designer to create the interface domain with informed ideas of the user's needs.

3.3.2 The Eight Golden Rules of User Interface Design

Along with the theoretical model for the design of the user interface, Shneiderman [20] has created what he calls "the eight golden rules of interface design". These rules are presented here and were applied as much as possible in the Webmapper extension.

1. *Strive for consistency.* Consistent sequence of actions should be required in similar situations, and style and terminology should be consistent in the interface elements

like buttons and menus.

2. *Enable frequent users to use shortcuts.* Frequent users familiar with the task and interface domain will likely want to speed up the time required to carry out tasks.
3. *Offer informative feedback.* Tasks should be coupled with appropriate user interface feedback. Attention drawn by the feedback should be proportional to the importance of the task.
4. *Design dialogs to yield closure.* Sequences of actions should have a clear ordering from beginning to end.
5. *Offer error prevention and simple error handling.* The system should be designed to reduce the chance of errors caused by user actions and provide the ability to correct them.
6. *Permit easy reversal of actions.* As much as possible, actions should be reversible, to help relieve user anxiety and encourage exploration of the interface.
7. *Support an internal locus of control.* The interface should provide the sense that the user is in control and that it is responding the user's input. Surprising events or tedious interactions should be avoided.
8. *Reduce short-term memory load.* The interface should be made as simple as possible to reduce the amount of information needed to memorize in order to perform the tasks.

3.4 Summary

This chapter described the goals of the Webmapper extension and explained relevant guiding principles from the fields of data visualization and user interface design. A methodological approach was used for the representation of the mapping data, and four different methods of interactions were presented to describe the implementation of the original Webmapper interface and to guide the development of the new interfaces. The different methods of interaction for the new interfaces were conceived to provide diversity in their appeal to different types of users and in their ability to represent different sized data sets.

Chapter 4

Implementation

For the scope of this thesis, two different representations were constructed and implemented by the author as alternate user interfaces to configure and visualize mappings in the Webmapper GUI. These alternate interfaces are part of what has been referred to as the Webmapper extension, and they have been incorporated into Webmapper's latest release. For the remainder of this thesis, Webmapper and its extension will simply be referred to as Webmapper, and each alternate interface (also referred to as a view or subview) has been given its own name. The new interfaces developed by the author are the ***grid view*** and the ***hive view***; Webmapper's original list-based view will be called the ***list view***.

This chapter describes the implementation details of the alternate interfaces and the technical implementation of the extension. Section 4.1 describes the components of the data. Section 4.2 describes the structuring of the user interface tasks. Sections 4.3 and 4.4 describe the grid view and the hive view respectively. Section 4.5 describes the technical implementation details and the architectural structure of the final software application.

4.1 Webmapper Data Components

The components of the Webmapper from table 3.3 are re-listed in table 4.1 for convenience, and their corresponding levels of organization and lengths[1] were added:

Note that transfer functions have been omitted because they are not represented as

[1]Bertin's method pertains to a fixed data set, but the context of Webmapper as a user interface implies possible variations in length. The notion of a variable length has been added to include a range of possible lengths.

Component	Level	Length
source devices	qualitative	variable: short to long
destination devices	qualitative	variable: short to long
source signals	qualitative	variable: short to long
destination signals	qualitative	variable: short to long
links between devices	qualitative	short (binary: linked or unlinked)
connections between signals	qualitative	short (binary: connected, unconnected)

Table 4.1 The Webmapper components with their levels of organization and lengths

part of the constructions; given the endless possibilities for transfer functions, it would be impossible to conveniently represent them all without introducing unnecessary complexity. A future version can, however, categorize transfer functions by type (ex: linear, differential, etc.), to create a more representable component. Transfer functions are therefore left as a property of a connection to be edited through a separate global "edit" bar at the top of the interface of each different view. The edit bar can be seen in figure 2.3.

4.2 Webmapper Task and User Interface Task Hierarchies

The process of designing an interface using the OAI model begins with a description of the tasks. Once the tasks are defined they can then be translated to interface domain objects and actions through metaphoric representations. Fundamental tasks are usually high-level and complex, and therefore require a decomposition into several smaller and lower-level subtasks. This means that both the task and the interface tasks will be broken into hierarchies of actions and objects. The goals of the software listed in Section 3.1 translate to two high-level tasks:

1. configure mappings between signals of devices on the libmapper network
2. visualize the devices, signals, links, and connections

As a demonstration of how the OAI model was used to design the new interfaces, the subtask that is the most fundamental and repeated action performed by the software is presented. It is then be broken into a hierarchy of even lower-level subtasks before it is translated into the interface domain. Four different hierarchies of interface subtasks are actually presented, and each one corresponds to a different view inside Webmapper.

The Task Domain

The subtask presented is: "**create a connection between two signals**". The following list decomposes the subtask into four smaller, lower-level subtasks:

1. select a source signal pertaining to a particular device
2. select a destination signal pertaining to a particular device
3. define the transfer function for the connection
4. create the connection

The User Interface domain

The first interface task hierarchy was created by following a very direct translation into the user interface domain. It is shown in table 4.2 and it describes the hierarchy associated with the hive plot that's implemented in the hive view.

1. view the complete set of signals
2. select a source signal pertaining to a particular source device
3. select a destination signal pertaining to a particular destination device
4. define the transfer function for the connection
5. create the connection

Table 4.2 Interface Task Hierarchy A (Hive Plot)

This approach is acceptable for smaller-scale libmapper networks and will allow mappings to be created easily between two (or a small number of) devices. However, as the number of devices increase it makes sense to organize signals by device. This is the approach libmapper takes by organizing its data into a two-tiered structure: signals pertain to device, and a link between devices is required before their signals can be connected. This organizational approach is better suited to larger network environments with multiple DMIs or software synthesizers, and it creates the task hierarchy that is associated with the list view shown in table 4.3.

Although libmapper's structure is organized into two steps for creating mappings between signals (i.e., by first creating a link and then creating a connection), there is no requirement that all user interfaces follow the same procedure. This is because links can be made automatically without requiring the user to create them explicitly; when the

1. view the entire set of devices
2. select a source device
   (a) view the set of source devices
   (b) select a source device
3. select a destination device
   (a) view the set of destination devices
   (b) select a destination device
4. create a link between the selected devices
5. select a source signal
   (a) view the set of source signals
   (b) select a source signals
6. select a destination signal
   (a) view the set of destination signals
   (b) select a destination signals
7. define the transfer function for the connection
8. create the connection

Table 4.3 Interface Task Hierarchy B (List View)

user makes a connection between two signals, the software can automatically check if the corresponding devices have a link, and if not, it can create the link before creating the connection; when the user requests to remove a connection, the system can check if there are any other connections between the corresponding devices, and if not, it can remove the link. For this reason, we can define a third task hierarchy that exclude the steps for linking, shown in table 4.4. This hierarchy is associated with the adapted hive plot that's implemented in the hive view.

The fourth and final interface task hierarchy used in Webmapper was designed to modify the notion of linking by allowing the user to view a set of signals pertaining to unlinked devices. This is the approach taken by the grid view and the task hierarchy is shown in table 4.5.

1. view the entire set of devices
2. select a source signal
   (a) view the set of source devices
   (b) view the set of signals of the desired device
   (c) select a source signal
3. select a destination signal
   (a) view the set of destination devices
   (b) view the set of signals of the desired device
   (c) select a destination signal
4. define the transfer function for the connection
5. create the connection

Table 4.4 Interface Task Hierarchy C (Adapted Hive Plot)

1. view the entire set of devices
2. select a source device
   (a) view the set of source devices
   (b) select a source device
3. select a destination device
   (a) view the set of destination devices
   (b) select a destination device
4. create a new view with the selected devices
5. select a pair of signals
   (a) select the view with the desired device's signals
   (b) view the set of source signals
   (c) view the set of destination signals
   (d) select the cell corresponding to a pair of signals
6. define the transfer function for the connection
7. create the connection

Table 4.5 Interface Task Hierarchy D (Grid View)

4.3 The Grid View

Description

The grid view was created by representing the components of links and connections as cells of a grid. Sources and destinations are grouped associatively and distributed along a respective dimension of the grid; sources are distributed along the horizontal axis to create columns and destinations along the vertical axis to create rows. Element namespaces are used as the label markers along each axis, and elements are ordered lexicographically for a consistent layout. Cells of the grid represent a possible link/connection between the pair of devices/signals at the corresponding row and column. An unlinked/unconnected cell is grey and a linked/connected cell is blue. The assignment of dimension is subjective and was chosen arbitrarily.

The grid view implements the interface task hierarchy D from table 4.5. The grid view follows the same pattern as the list view by dividing the mapping description into two constructions: one for devices (called the "devices grid") and one for signals (called the "signals grid"). When the grid view is initialized, the user is presented both grids; the devices grid is on the left and populated by all devices declared on the network; the signals grid is on the right and initially empty. Unlike the list view, the ability to see device signals is not an automated process through device linking. Devices must be added to the signals grid explicitly by selecting a device label (or a cell corresponding to a pair of devices) and then pressing the "Add" button; signals of the added devices will then be displayed in the signals grid. Devices are removed from the signals grid in the same manner using the "remove" button. Cells of devices that are included in the signals grid are indicated using texture and orientation; a vertical or horizontal line is added to cells of added source or destination devices respectively, creating an easily identifiable "+" symbol for cells corresponding to an included pair. A screenshot of the interface can be seen in figure 4.1.

Preset Views

The grid view is equipped with functionality for managing multiple configurations of the signals grid using what was named "presets". Once the desired devices have been added to the signals grid its configuration can be saved as a preset, causing its list of included

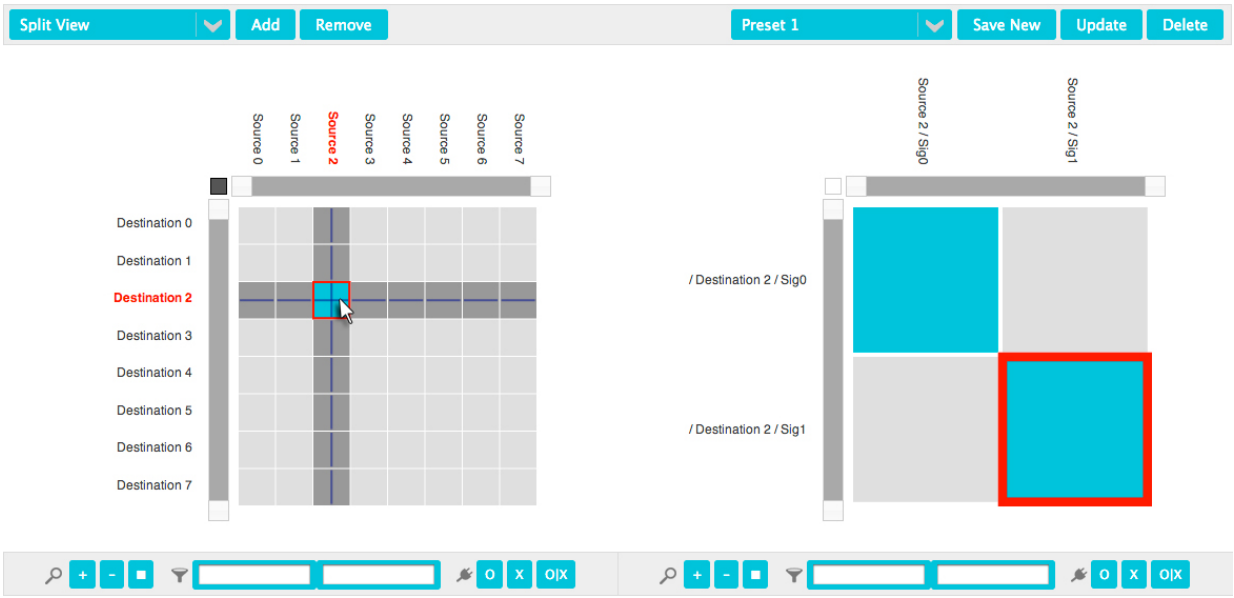


Fig. 4.1 A sample of the grid view interface with arbitrary data. The devices grid is on the left and it has one device pair added to the signals grid on the right. Cells of included source and destination devices have a vertical and horizontal line respectively. Blue cells correspond to links or connections, and selected cells have a red border. The image shows the visual feedback created by placing the mouse cursor over a cell in the grid.

devices to be stored in memory. Presets are managed using a drop down list in the top right portion of the interface. By creating multiple presets, the user can quickly switch back and forth between different self-defined sub-views of their system.

Linking and Connecting

To create links or connections, users can select and navigate among cells of the grid using the computer keyboard or mouse. The selected cell is highlighted in red, and cells of the corresponding row and column are highlighted to help guide the eye as it searches for the corresponding labels of the row-column pair. Once a cell is selected, a link/connection can be made or removed by clicking on buttons at the bottom of the interface or by using keyboard shortcuts. If a connection is made between signals of unlinked devices, a link is created automatically and displayed in the devices grid. If the last connected signals between a pair of devices is removed, the link is also removed. The grids support the selection of multiple cells at the same time.

As in the list view, the ability to change properties of a connection is done through the global edit bar at the top of the screen (not shown in figures).

Namespace Filtering

When the data set grows large, it becomes more difficult to find specific labels. Inspired by the list view, the grids also support namespace filtering. The filters will display only those devices/signals matching the supplied regular expression criteria. Unmatched elements are removed temporarily from the display, and can be made visible again by clearing the filter.

Zooming

The grids are equipped with two methods of zooming. By default the grids are set to 'zoom to fit', meaning that the zoom level will constantly resize itself to keep all devices/signals in view. Users can also set a custom zoom level by using the provided GUI buttons to zoom in or out, or by using the zoom-scroll sliders for either dimension. These custom sliders were extended from the JQueryUI[2] slider widget. The sliders offer feedback pertaining to the current zoom level and position; the handle sizes of the sliders are proportional to the percentage of the grid that is currently in view, and their position reflects the position of the view with respect to the entire grid. Zooming is achieved by clicking on the button at either end of the handle and dragging it to the desired proportion. Scrolling is achieved by clicking and dragging the handle itself. Zooming out in both zoom modes is not restricted to a certain maximum size, however at a certain point labels begin to overlap; this decision to allow overlapping labels was made as a trade off for a visualization that can represent the maximum number of signals. Figure 4.2 shows the signals grid at a custom zoom level.

Maintaining Aspect Ratios

Both methods of zooming maintain aspect proportions, keeping cells of the grid as squares. This decision was made for aesthetical reasons, but also to avoid distorting the shapes of the GUI elements that are representing qualitative components (i.e., cells varying in width and height could impart a misleading representation to elements of equal importance). When in zoom-to-fit mode, the grid keeps all cells in view by zooming to accommodate

[2] `http://jqueryui.com/`

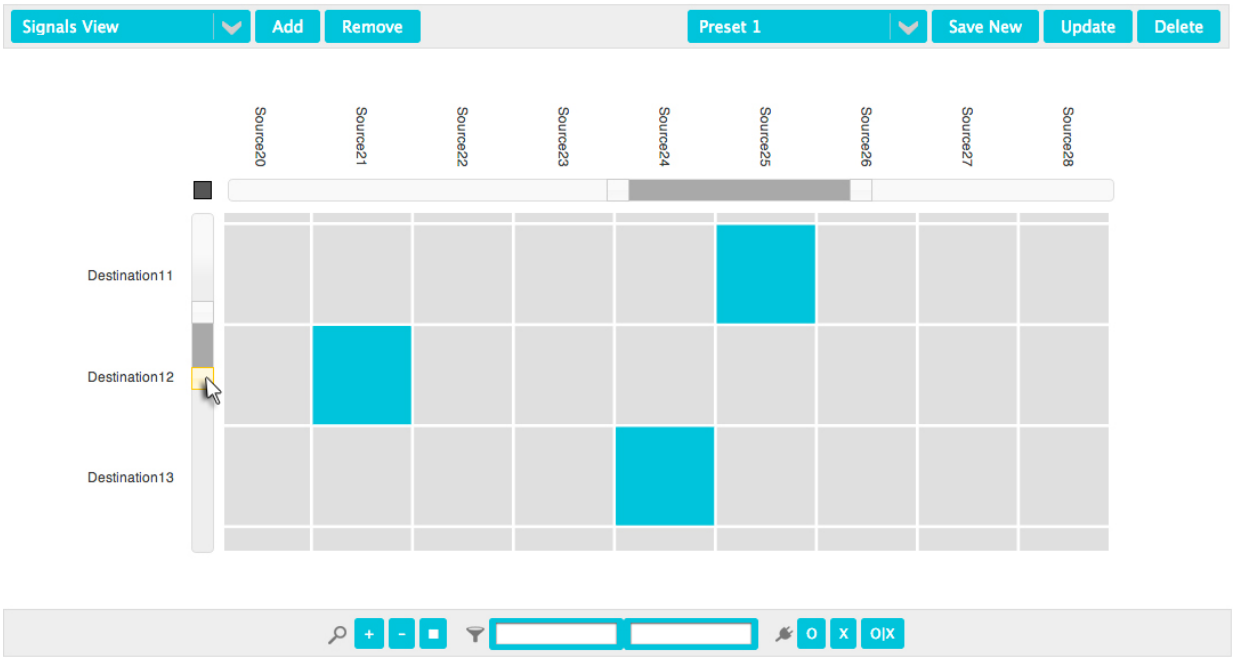


Fig. 4.2 A sample of the signals view mode with arbitrary data at a custom zoom level. The zoom-scroll bars provide feedback on the zoom level and position inside the grid. The image shows the user clicking and dragging a handle (highlighted orange) to adjust the zoom level. The right hand side of the upper menu bar can be used to switch between signal view presets.

the largest dimension with respect to the aspect ratio of the grid. With the custom zoom mode, scrollbars are provided for separate dimensions but they are linked when zooming to maintain aspect ratios.

View Modes

The grid view initially displays both grids, but two additional view modes are available to display a single grid at a time and maximize the grid area. Grids in all view modes are dynamic and will resize to fill the maximum space defined by the browser window. An example of the devices view mode containing 39 source devices and 19 destination devices is showin in figure 4.3.

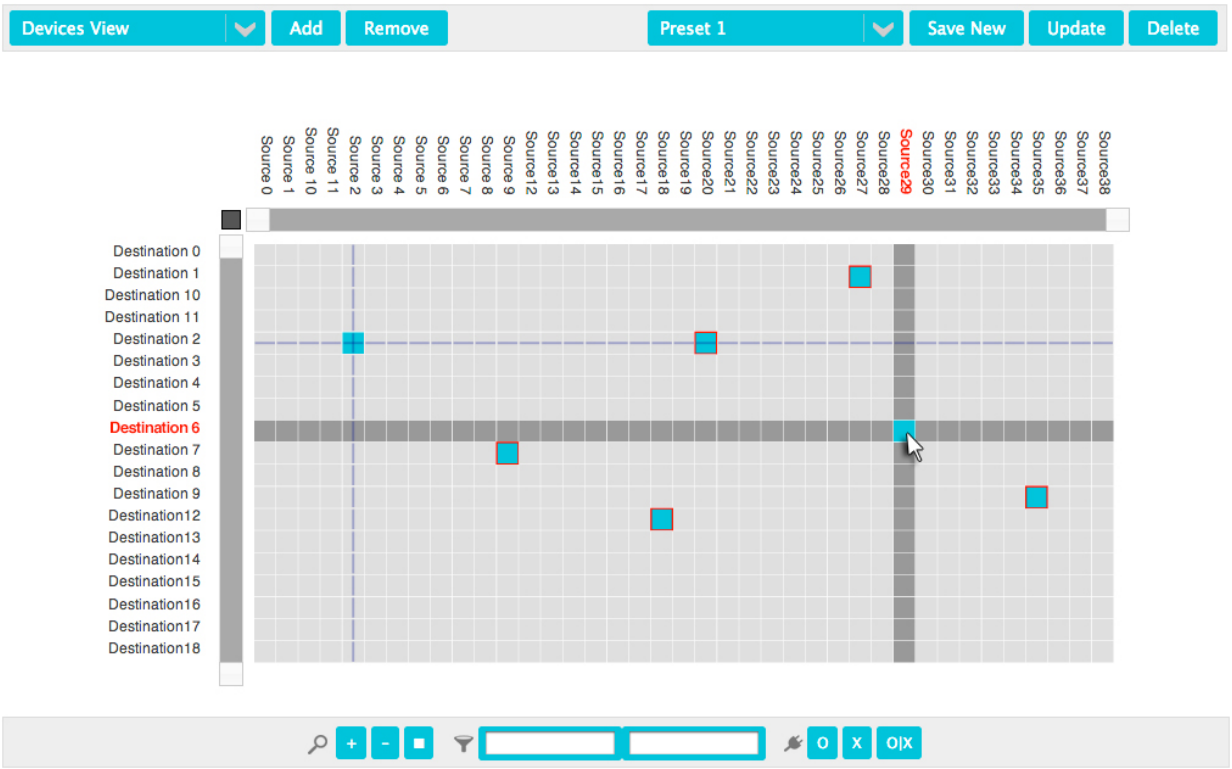


Fig. 4.3 A sample of the devices view mode with arbitrary data. The left hand side of the upper menu bar can be used to switch between view modes or to add devices into the signals grid.

Keyboard Shortcuts

Keyboard shortcuts are provided for all of the user interface tasks, including adding devices to the signals grid, switching the view mode to display both or a single grid, saving presets, cycling between saved presets, moving the selected cell by 1 or 3 indices, toggling a connection, and zooming in or out. When both grids are displayed, a small black square in the upper left corner of the grid area indicates which grid is active and will receive triggered keyboard shortcuts. Once familiar with these shortcuts, the interface becomes completely accessible through the keyboard, eliminating the need to use the mouse, and effectively increasing the speed at which more experienced users can configure mappings.

4.4 The Hive View

The Hive Plot

Krzywinski's Hive plot [21] places nodes on radially-oriented linear axes. The plot gets its name from the orientation of its axes and curved lines connecting nodes resembling the top of a beehive. The non-overlapping axes provide an uncluttered layout, and unlike network representations that define their structure based on aesthetics, the hive plot structure is defined by two attributes derived by the structure of the underlying network. This creates a consistent layout representation when the network has a consistent structural profile, and aims to provide a "visual signature" of the underlying data. The hive plot considers three guidelines:

1. rules that govern assignment of nodes to axes,
2. rules that govern layout profile of axes (i.e., position, scale and orientation), and
3. rules that control the format of edges drawn as curves between nodes.

Description of the Hive View

The hive view was created using the interface task hierarchy A (table 4.2). The categorization of components from table 4.1 translate easily into two axes: one for source signals and the other for destination signals. Signals are represented by nodes and are assigned correspondingly to either the source or destination axis. Bezier curves represent connections between signals. The axes are perpendicular to maximize space, but viewed from a simulated perspective to mimic a three-dimensional space. Unlike the list view, the hive plot is not organized into a two-tiered structure and links do not need to be made explicitly. Furthermore, links and devices are not represented explicitly because they can be inferred through the representation of connections and signals respectively. Devices (represented through their nested signals) are grouped by position along their axis using the principle of proximity, and are color-coded by the assignation of a random color.

Connections are represented by smooth bezier curves joining the connected nodes. Transfer functions are not represented, but a future version can incorporates categories of transfer functions through the use of another visual variable, such as color or texture (e.g., dashed lines) on the connecting lines. A screenshot of the interface can be seen in figure 4.4.

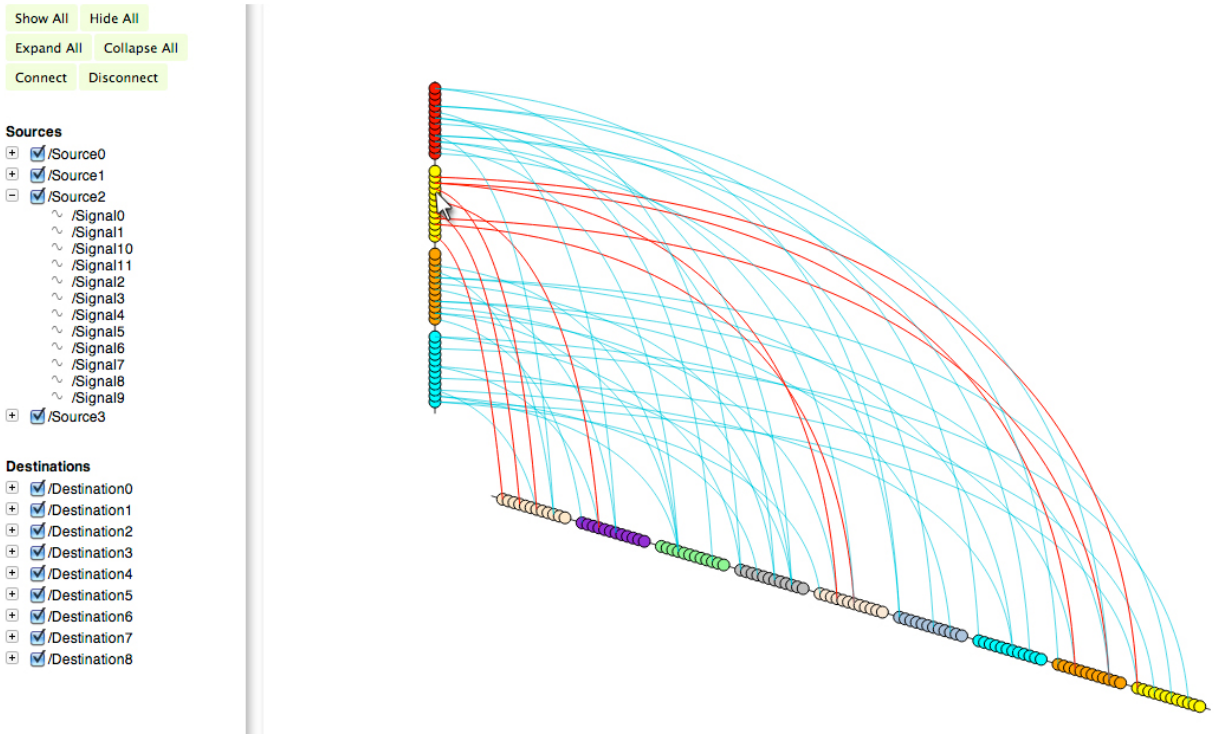


Fig. 4.4 An example of the hive view's hive plot interface with arbitrary data. The mouse cursor can be seen in the plot hovering over the source device "/Source2", causing all its connections to be highlighted.

Node coordinates in a hive plot are typically derived from the absolute or rank-ordered value of a node parameter relating to the network [21]. However, since devices and signals are qualitative components with no universal ordering, and since the meaning of signals are established through their names, this was not the route taken in the hive view. Instead, signals are ranked lexicographically according to their namespace, a property detached from the structural properties of the network. This decision was made to provide the most consistent layout when devices, signals, and connections are added or removed, in hopes of providing the a memorable visual signature as the mapping evolves.

Description of the Adapted Hive View

A second hive view called the *adapted hive* plot was created by modifying the rule that governs the layout profile of axes. Motivation for this second view mode was to provide more variety in the imagery created by the visual signatures of the plot. The containment

of signals to two axes in the hive plot substantially limits the range of positions in both dimensions at which nodes could be placed, thereby reducing possible variations in the imagery of the plot. Furthermore, when the mapping description is large, the available space in which to place nodes becomes problematic when there are only two axes.

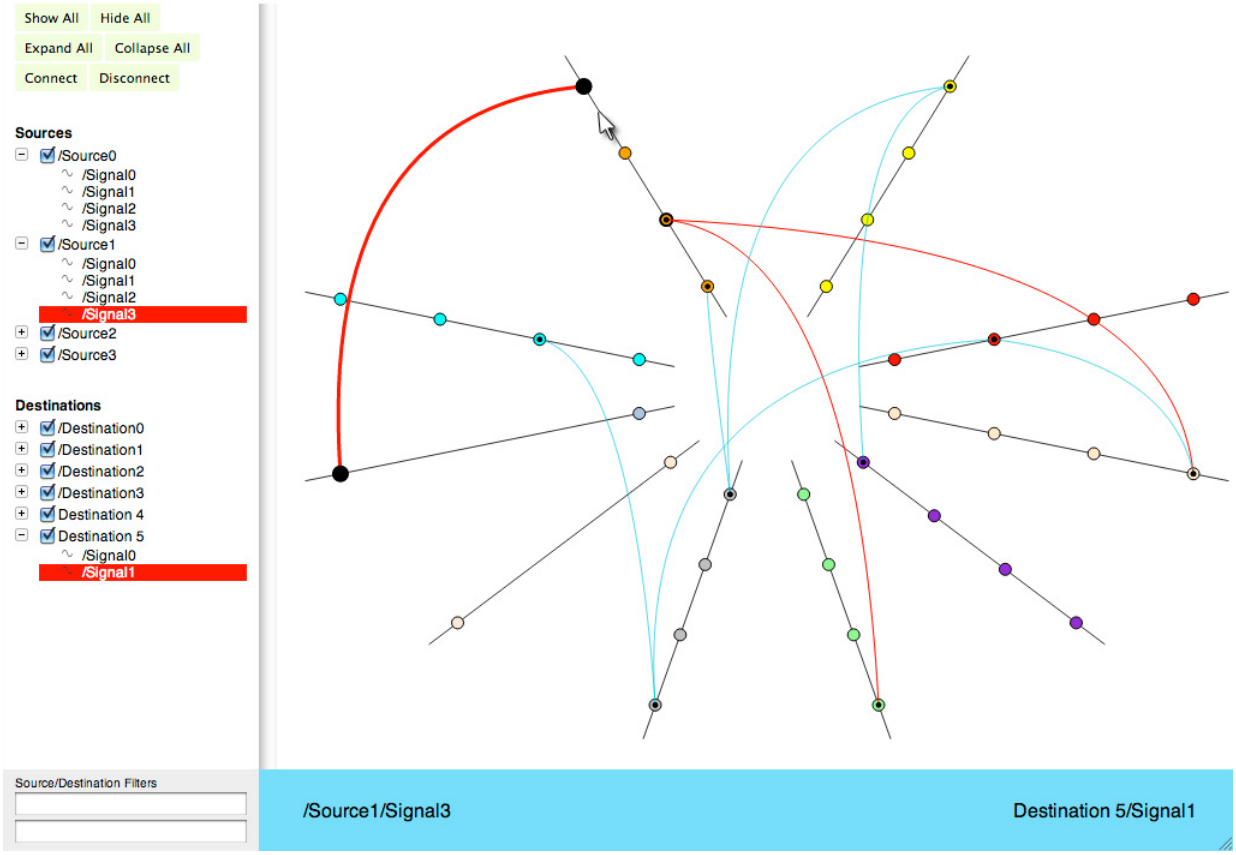


Fig. 4.5 An example of the Hive view's adapted hive plot interface with arbitrary data. The connection between "/Source1/Signal3" and "/Destination5/Signal1" has been selected by the user and the interface provides visual feedback: signal names are highlighted in the table, the nodes and connection line in the plot are highlighted and bolded, and the signal names are shown in the blue bar at the bottom of the interface. Additionally, the mouse cursor can be seen in the plot hovering over the source device "/Source1" causing it's connections to be highlighted.

In the adapted hive plot an axis is created for every device. Signals are then represented by nodes placed on the axis corresponding to its parent device. Grouping of nodes by proximity is replaced by axis assignment, but the assignment of color remains as an aid in

identification. Axes are oriented and spaced out equally around the center of the display, with the upper half of the display for sources and the lower for destinations. When the mapping description contains only one source and one destination device, the adapted hive plot resembles the original hive plot, but as devices are added the visualization becomes more varied in its display. An example with four source and six destination devices can be seen in figure 4.5.

Identification of Signals and Visual Signatures

Labels are omitted on the plot itself with the original intention of placing more focus on the visual signature. However, since elements in the plot can ultimately only be identified through their namespace, this is achieved using an interactive table found on the left hand side of the display. The table consists of two hierarchical lists, one for sources and the other for destinations. The hierarchy is defined by two levels: devices and their nested signals.

The plot and table are equipped with interactive capabilities to aid in identification of signals and in recognition of subsets of the visual signature; by placing the mouse cursor over a device (i.e., a device label in the table, or an axis in the adapted hive) all the nodes and connections associated with the device are highlighted; by placing it over a signal (i.e., a signal label in the table, or a node in either hive view), all the connections associated with the signal are highlighted. Mousing over a device is shown in both figures 4.4 and 4.5.

Similarly to the grid view, the table is equipped with text filters allowing the user to enter a regular expression to restrict the displayed signals to matching namespaces.

Connecting and Linking

The adapted hive view uses the interface task hierarchy D from table 4.5. Mappings can created or destroyed by selecting a source and destination signal either in the table or the plot, and using the provided “connect” and “disconnect” buttons. As usual, connection parameters are edited through the global edit bar at the top of display (not shown in figures).

4.5 Technical Implementation Details

This section describes the technical implementation, introducing the relevant architectural patterns and describing the final structure.

HTML5 Compatibility

Webmapper was originally developed to run in web browser with intentions of being cross-platform compatible. Webmapper uses the Python programming language[3] to register a "monitor" that communicate with the libmapper network. The monitor acts as the communication bridge between Webmapper's HTML components and the libmapper network; it is thus capable of triggering all libmapper functions such as creating or modifying links and connections, or querying the libmapper network to discover any aspect of its data.

By running in a web browser, development of the Webmapper extension made use of HTML5[4] compliant strategies including:

- Hypertext Markup Language (HTML): the core language of the World Wide Web [5]
- Javascript (JS): a scripting language used to script web content [6]
- Cascading Style Sheets (CSS): a style sheet language used for describing presentation semantics of HTML documents. [7]
- Scalable Vector Graphics (SVG): an XML-based format for two-dimensional vector based graphics that supports interactivity and animation [8]

In addition, JQuery[9], a library built atop JS, was sparingly used to benefit from its higher-level procedures that require less code than typical JS.

4.5.1 MVC and PAC Architectural Patterns

An overall structure was created and added to Webmapper borrowing strategies from both the model-view-controller (MVC) and the presentation-abstraction-control (PAC) architec-

[3] `http://www.python.org`
[4] http://www.w3.org/html/wg/drafts/html/master/
[5] `http://www.w3.org/TR/html51/`
[6] `http://www.ecma-international.org/publications/standards/Ecma-262.htm`
[7] `http://www.w3.org/TR/css-2010/`
[8] `http://www.w3.org/TR/SVG11/`
[9] `http://jquery.com/`

tural patterns. Design patterns aim to make code more organized, and this seemed as a wise route to take considering the specification for multiple interfaces, and to provide a software tool to the community that is easily maintainable.

The MVC architecture was originally described from its use in the Smalltalk-80 system [22] and it structures the application into three types of modular components. The *model* of an application is the domain-specific software simulation or implementation of the application's central structure. It represents the application's data, and contains functions for manipulating the data. *Views* display aspects of the model and deal with everything graphical. They can request data from the model and are responsible for its display. Views can be nested and contain other subviews or be contained within superviews. *Controllers* are used to send messages and requests to the model, and provide the interface between the model with its associated views and the user input devices (e.g., keyboard, mouse). Note that the MVC specifications include "pluggable views" capable of handling user input like a controller.

MVC has the notion of *dependents*, where views and controllers of a model are registered as dependents of the model to be informed whenever something in the model is changed. A typical interaction cycle is as follows: the user takes some input action and the active controller notifies the model to change itself accordingly; the model carries out the prescribed operations, possibly changing its state, and broadcasts to its dependents (views and controllers) that it has changed; upon notification of change views can then inquire to the model about its new state, and update their display if necessary. Controllers may change their method of interaction depending on new states of the model.

Given that Webmapper will be used to display only a single view at a time, the MVC architecture was decided as somewhat excessive; instead of defining a controller for each view, and instead of having both the controllers and views handling user input, the created architecture intends to provide a more centralized interaction cycle that's simpler to implement and understand.

Subsequent research and browsing on community-driven forums for practical examples revealed that there are several differing interpretations and implementations of MVC. For example, Burbeck describes how *active* and *passive* MVC differ in how dependents are notified of changes in the model [23]. Leff and Rayfield even refers to PAC as the same thing in spite of notable differences [24]. All MVC implementations, however, share the key concept of separating the user interface from the underlying data.

The PAC model [25] is similar to MVC, and it structures the application into three parts: the *presentation* defines the input and output behavior of the application as perceived by the user; the *abstraction* corresponds to the semantics of the application along with the functions that it is able to perform; the *control* part contains the mapping between the abstraction and presentations, and is intended to manage the overall interaction between the user and the application. PAC differs from MVC in two main points. First, message passing in MVC must maintain consistency through all the three components, so the notion of control is spread across all structured components (i.e., the model, view and controller) whereas it is explicitly centralized in PAC (i.e., in the control). Second, PAC combines input and output behaviour into one component (i.e., the presentation) whereas MVC distributes them across two objects (i.e., the view and controller).

4.5.2 Final Architecture

This section describes the final architecture inspired by the MVC and PAC patterns. Terminology from the MVC components has been maintained although their implementations have been modified.

The View

The view is responsible for displaying desired elements of the model. It is organized by a main "superview" that holds a container for individual views (e.g., the list, grid, and hive views) to be loaded in and out. Views can be selected and loaded using the drop down menu at the top of the display.

When the application is initialized, the main view loads the first subview in the list (i.e., the list view) and sends a request to initialize the model's data. When a new subview is selected, the main view saves any view specific settings, unloads the current sub-view, loads the new subview, and if necessary, restores the previously saved sub-view settings.

The view contains a reference to the model and the controller notifies the view when there is a change in the model. The view handles the notification by querying the model and updating its display. Views query the model, but they do not modify it.

The edit bar is located at the top of the display, and is structured as an MVC style view-controller pair. It remains static throughout all the subviews and has its own controller that is responsible for notifying it of any selected connections and for communication with

the python monitor for modifying connection parameters.

The Controller

The controller is responsible for all communication between the views and the monitor; it takes any updates from the python monitor and passes them to the model; it watches for any commands triggered by the loaded view and passes it to the monitor; and it is responsible for notifying the view to update its display when there is any change to the model.

The Model

The model is constantly maintained (through the controller) to mirror all devices, signals, links, and connection in the libmapper network. The model contains functions for higher-level querying from the view. Though views can access the model, only the controller modifies it.

Flow of Interaction

The final structure of the Webmapper architecture can be seen in figure 4.6. The figure shows the sequencing of a command issued by a user in the view. Arrows in blue follow the flow of commands to libmapper; arrows in red follow the flow of data coming from libmapper. Yellow circles represent elements inspired by the MVC and PAC architectural patterns. The grey background encompasses all Webmapper components. The flow of interaction from the user to libmapper is as follows:

1. The user interacts with the view, triggering a command destined for the libmapper network.
2. The command is sent to the controller.
3. The controller receives the command, formats it, and makes a request to the monitor.
4. The monitor translates and passes the request to libmapper.
5. The result of the processed request is returned to the monitor.
6. The monitor translates the result and passes it to the controller.
7. The controller updates the model with the supplied data from the monitor.
8. The controller notifies the view that the model has been changed.
9. The view queries the model and updates its display.

Note that Webmapper being a collaborative environment, changes can result from other users modifying the network. In that case, the flow of interaction will begin at number five (the blue arrows).

Benefits

Although the initial implementation demanded more code, the final modular design creates a robust system that is easily understandable and maintainable. The set of possible commands from the task domain is made obvious in a single location (i.e., the controller) by explicitly listing all events (and their parameters) being watched for from both the subview's placeholder container and the python monitor. This means that when creating a new view, the developer does not need to be concerned with lower-level implementation details of communication with the python monitor or libmapper; he simply needs to be aware of the events that can be triggered by the view and implement them according to the view's needs. The view has only a single function that needs to be implemented for updating the display upon notification of a change. Furthermore, the views are isolated from the monitor and the controller is responsible for mediating messages between them. This means if implementation details of the monitor or libmapper change, only the controller needs to accommodate the changes; whereas if the views were communicating directly with the monitor then all the views would need to be updated as well.

Inheritance

The grid view was faced with needing multiple grids that were mostly identical in function and behavior. There were, however, notable idiosyncratic differences between functions in the devices and the signals grid. For example, the device grid has functionality for adding and removing devices into the signals grid, and displays textures on the cells of included devices. Another example is that devices and signals have different properties associated with them. Classical object-oriented inheritance was used to resolve situations that required additional or modified functionality; a grid "superclass" was created containing all the shared functionality, and then a class was created for each the devices grid and the signals grid, inheriting shared functionality from the superclass but adding or overriding any functions that needed special behavior. A similar situation was faced and resolved in the hive plot for handling both view modes. The object-oriented principles of inheritance

was invaluable in making the code robust and reusable.

4.6 Summary

This chapter thoroughly described the two new interfaces created in Webmapper and their different methods of interaction. Their conception was shown to follow a methodological procedure using principles from data visualization and user interface design. It has been shown that different styles of visualization, combined with different user interface task hierarchies, enables different methods of interaction with the mapping layer. The HTML-based technical implementation details of the extension were described, and the final structure of the software that was created using principles from the MVC and PAC architectural design patterns were described to make the codebase easier to maintain and expand.

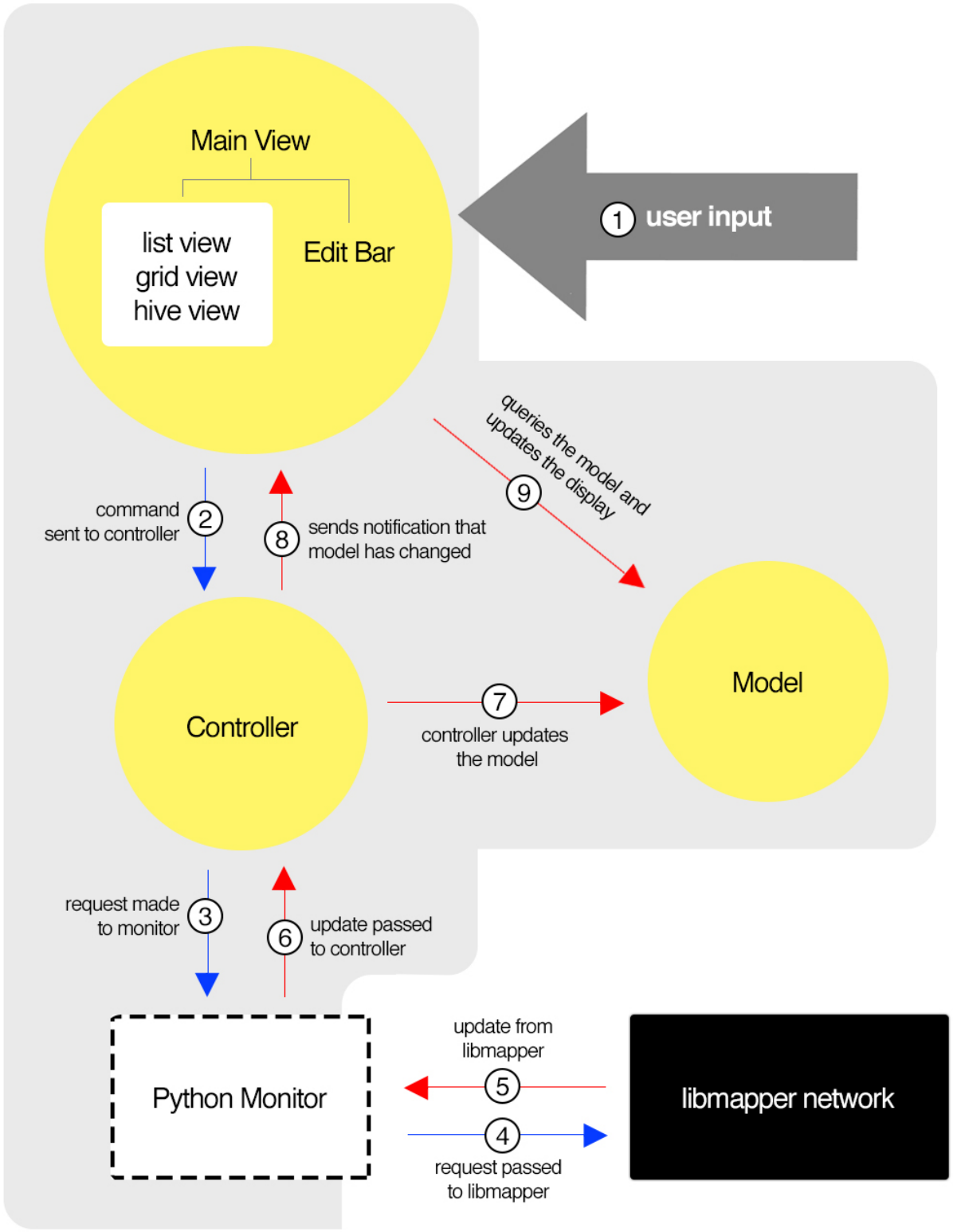


Fig. 4.6 Webmapper's final architecture and an example flow of interaction.

Chapter 5

Discussion

This chapter introduces the criteria used for the evaluation of the alternate interfaces in Webmapper. Informal feedback was collected through a small number of test users, and discussions pertaining to each criterion is presented for each interface. The three interfaces are then ranked from one to three (from better to worse) for each criterion.

5.1 Properties of the Data Set

Webmapper was designed to be a generic mapping tool to be used in situations requiring mappings between DMIs and sound generating devices. For this reason, the data set is not fixed and will vary among different users and contexts. For example, Webmapper can be used by a single person creating mappings in a single DMI; at the other extreme, many users can work in collaboration on the mappings between a network of DMIs. A mapping description in Webmapper can therefore consist of any number of devices, signals, or connections, and the software interfaces must take this into consideration. For issues relating to the number of devices and signals we refer to the *size* of the mapping; for issues relating to the number of connections we refer to the *complexity* of the mapping.

5.1.1 Size

For practicality we define minimum and maximum values for the number of devices and signals that typically will not be surpassed. The values were chosen through discussions with the developers of Libmapper to establish a set of possible scenarios in which the

software would be used based on past and expected applications. The maximum values defined here are therefore not explicit and the user interfaces should also be able to handle data sets that surpass these values. Values beyond the minimum are out of scope and would not require the use of software.

- **Minimum**: one source device and one destination device, with two signals per device
- **Maximum**: 80 source devices, 80 destination devices, and 100 signals in a single device

We define *small* as less than five, *medium* as between 5 and 25, and *large* as greater than 25. Within the range defined by the minimum and maximum, we can assume any possible combination of small, medium, or large sizes of source devices, destination devices, source signals, and destination signals.

5.1.2 Complexity

The libmapper software currently supports one-to-one and one-to-many mappings between signals[1]. One-to-one mappings imply that the complexity of any link will not grow beyond the cardinality of its smallest set; one-to-many mappings imply that the complexity of a given link is limited by the cardinality of its range (i.e., the set of destination signals). Libmapper may in the future support many-to-many mappings, implying that theoretically n *by* m connections could become possible, where n and m are the cardinalities of the domain and range[2].

When mappings become more complex their representation becomes more challenging, and each different view can be evaluated on how different levels of complexities can be perceived. A data set with few connections will be referred to as *simple* whereas one with many connections will be called *complex*.

Note that this notion of complexity can also be applied to links, however complexity in links do not necessarily correspond to complexity in connections; this is because a link is defined by libmapper as two devices with a *potential* connection between their signals.

[1] Although not officially supported, many-to-many mappings can be created, causing a destination signal to alternate randomly between data from its connected sources. Ideally many-to-one mappings would have some algorithm to combine the incoming data, but since that algorithm is not yet implemented many-to-one mappings are not officially supported.

[2] though it is unlikely that every source signal will be connected to every destination

Links are used differently in all three views. Inside the list view they are an organizational feature used to place signals of two devices into view inside a newly created tab; the grid view automates the linking process ensuring that only devices with connected signals are linked; and the hive view does not represent links at all. Therefore, the concept of complexity can be applied to links but it should be discussed independently of complexity in signals. Links can be many-to-many mappings because the connections to signals of a destination device can come from signals of different source devices.

5.2 Evaluation Criteria

5.2.1 Human Factors for Evaluation

The various views were evaluated based on the five measurable human factions taken from Shneiderman [20]:

1. ***Time to learn***. How long does it take for typical members of the user community to learn how to use the commands relevant to a set of tasks?
2. ***Speed of performance***. How long does it take to carry out the fundamental tasks?
3. ***Rate of errors by users***. How many and what kinds of errors do people make in carrying out the fundamental tasks?
4. ***Retention over time***. How well do users maintain their knowledge over time?
5. ***Subjective satisfaction***. How much did users like using various aspects of the system?

Feedback from informal discussions with users was collected, along with observations made from monitoring these users as they used the software. These users were undergraduate and graduate students at the IDMIL that were exposed very brief demonstrations of the interfaces. The users were those who expressed interest in the thesis project and volunteered to test the interfaces during the end phases of their development. They were monitored both as they began using the interfaces and after becoming familiar with them. They all used the software to create mappings for the particular DMI that they were building. Additionally, feedback was also collected from a couple of graduate alumni.

5.2.2 Evaluation of the Secondary Goal

The adaption from text-based interfaces to graphical interfaces requires the visual representation of abstract concepts when designing UI objects and actions [20]. Furthermore, principles of data visualization studied by Bertin and others ([18] [17] [19]) provide methods for improving the way data is displayed to achieve informative representations.

Each view inside Webmapper provides an alternate representation of the underlying network and follows a different set of user interface objects and actions (Section 4.2). Therefore, each view utilizes a different method of visualization; the list resembles a bipartite graph, the grid displays a matrix of signal pairs with binary values (i.e., mapped or unmapped); and the hive view provides graphical plots of all the signals in the network. Therefore, each view has a varying level of abstraction in the visualization that can be evaluated on its ability to cope with different data sets. A greater ability to visualize was attributed to representations that could represent larger and more complex data sets.

5.3 Evaluation

5.3.1 Time to Learn

For each view, all users managed to accomplish the fundamental task of creating a mapping between two signals relatively quickly. There were, however, some slight differences in the time and difficulty that was required when learning how to use each interface.

In the **list view**, users were comfortable with exploring the interface through clicking and dragging. The list view also provided an obvious sequence of steps to follow; after having managed to create a link, a new tab pops up immediately, begging to be clicked on. Users intuitively clicked on the tab, saw a list of signals, and made a connection between two signals. After the first successful mapping was created they were able to create others with ease. This could be because the users are already familiar with computers so the knowledge from tables, tabs, and clicking and dragging is transferred. Furthermore, the technique of connecting by dragging to create lines has a familiar metaphor; it resembles patch cables in real-life audio equipment, and the method of connecting objects in the popular visual programming languages Max/MSP[3] and PD[4].

[3] `http://cycling74.com/products/max/`
[4] `http://puredata.info/`

With the **grid view** the response was somewhat different. It was less intuitive for some users to work with lists spanning two dimensions and cells as intermediate UI objects. To other users, this seemed as equally intuitive as the list view, perhaps because of their familiarity with other audio hardware and software. For example, digital drum machines arrange tracks as rows and subdivisions of the beat as columns, with cells corresponding to note on or offs; MIDI editors behave similarly; and the monome[5] arranges its buttons in a grid. Furthermore, zoom sliders are common in audio editing programs (e.g., Apple's Garageband[6]). So there is potential for the transfer of knowledge from other applications in music, though perhaps a smaller percentage of users will be familiar with these applications then in the list view.

Additionally, the grid view was designed with features that were less automated than the list view. By allowing the creation of multiple user-defined views through adding devices and saving presets, more complexity is added in the user interface tasks that requires more time to learn.

The **hive view** was the quickest to learn. Both plots display all the signals on screen in a single image, and provides the simple ability to click two nodes and make a connection. The table provided in the hive view offers the same functionality by displaying a complete list of the signals with clickable elements. Although less intuitive, the process was simple to understand and learned quickly. The interface tasks lacked any additional steps necessary in the grid and hive view to create links and views, and therefore was the simplest to learn.

Conclusion

The time to learn was proportional to the complexity of the user interface task hierarchy, affected by the user's familiarity with other software applications with similar metaphors or UI objects and actions.

Ranking: 1. Hive 2. List 3. Grid

5.3.2 Speed of Performance

The speed at which users could perform the mapping tasks varied in each view. Performance was monitored after the users successfully grasped all functionality in the complete set of

[5] `http://monome.org/`

[6] `http://www.apple.com/ca/apps/garageband/`

UI actions. The dominant factors affecting speed of performance was the ability to locate signal names and the organizational structure of the UI actions, and the evaluation was therefore broken into two parts.

A. Ability to Identify Signals

The **list view** enabled the best performance time for locating and identifying signals. It devotes most of its display for this task, and the UI objects are the namespaces themselves. Placing device or signal names in a vertical alphabetized list enabled quick retrieval of a particular device or signal with small to medium sized data sets, whereas the time required to locate a signal increased significantly with longer data sets. By placing sources and destinations associatively in two separate lists, they could be dealt with independently.

The **grid view** also distributes sources and destinations associatively into two separate lists, but user performance was impeded because the vertical labels of the columns were difficult to read. Furthermore, by using cells to represent pairs of elements, an additional step is created requiring the user to locate the labels in the corresponding row and column of the cell.

The **hive view** also provided associative lists of signals as a column on the left of the display, but in this case both lists for sources and destinations share the same area, and the designated area was small compared to the list view. These differences impeded the speed at which signals could be located in the lists. Although the lists are hierarchical, locating a particular signal becomes tedious as items are expanded. Users also had the option to locate signals using the plot, but the evaluation of speed in this case is not so simple. At first glance, the plot takes the longest to identify signals because it requires mousing over individual elements in the plot or in the list. But since the adapted hive plot provided a view of signals that was graphical and color-coded, once familiar with the assignment of devices to axes and signal color-codes the process of locating signals is substantially improved; being unfamiliar with the assignment, however, means that locating signals in the plot is impossible. Although the hive view was evaluated as the slowest in user performance, it should be noted that for small data sets, although dependent on the user's ability to memorize color codes and locations of nodes, it can be the quickest way to locate signals.

Conclusion

The speed of performance is tightly associated to the speed at which signals can be identified and selected. Any difficulties or additional steps that arise in the process of identification slows down performance. The time required to locate a signal in a list-based view is proportional to the length of the list, suggesting that intelligent filtering and sorting capabilities are important. Memorization of signal locations in the visual representation can eliminate the reliance on textual labels and speed up performance.

Ranking: 1. List 2. Grid 3. Hive

B. Performance of Tasks Based Organizational Structure

When the data set grows larger, the organizational structure of the UI objects and actions start to have a greater effect on the overall performance to carry out the tasks.

The **hive view** can be seen as a hybrid view because if offers both a list-based view and a graphical representation. The list is especially difficult to navigate when the size of the data set is large; although the list is hierarchical, navigation is challenged when multiple items are expanded. Regarding the plot, the interface actions prescribe a method of interaction that is also quite challenging. They require the user to decipher complex visual representations that can be created by large and complex data sets. They also require skillful control of the mouse; Fitts law [26] predicts that the time required to move the mouse to a given target proportional to the distance to the target and inversely proportional to its size, meaning performance of interaction in the plot is challenged by small nodes spread across the entire display.

When the data set is large, the **list view**'s organizational structure of creating links improves the overall performance of the tasks because the interface becomes more manageable through tabbed subviews. The grid view offers a similar organizational structure, but it benefits from a particular advantage: its ability to experiment with signals spanning any combination of source or destination devices. In the list view, a tab is automatically created for every linked source device and it will contain the signals corresponding to all the source's linked destination devices. Therefore, alternating a connection between two particular source devices requires a constant switching between tabs. The **grid view** can accommodate this interaction with its custom user-defined views that can be populated with signals from any device. Although there is an initial overhead in defining the custom

views, performance is improved as the data set grows larger. Furthermore, when the data set gets larger, it can be faster to create custom views accessible through a drop down list then to search among a multitude of automatically generated tabs.

It was noticed that because scrolling and zooming using the mouse wheel was not implemented in the grids, zooming and scrolling required the manual use of the scroll bars. This impedance can be overcome through keyboard shortcuts for moving the selected cell (i.e., to scroll) or zooming, but the use of these shortcuts varied among the users; some were simply more comfortable with or preferred to use the mouse. This was not considered for the evaluation because mouse-wheel scrolling could easily be added in the future.

Conclusion

The user interface task hierarchies define the flow of interaction when carrying out tasks and strongly influence the speed of performance. The list organizes signals through links, the grid through custom views, and the hive uses a single display for all signals; each view offers its own set of advantages and disadvantages with varying speeds of performance.

Ranking: 1. Grid 2. List 3. Hive

5.3.3 Rate of errors by users

When performing the fundamental task of creating a mapping between two signals, the fundamental type of error is the creation of a connection between the wrong signals (i.e., two signals not intended to be mapped to each other). Although the informal tests that were done with users was not sufficient for any quantifiable comparisons to be made, some assumptions can still be made. The ability to correct the error is also taken into consideration.

The fundamental error can stem from two possibilities attributed to the design of the user interface:

1. **The misidentification of the intended signal**. The user searches for the user interface object associated with the intended signal but identifies it incorrectly and selects the UI object associated with another signal.
2. **The accidental manipulation of an unintended signal**. The user correctly identifies the UI object associated with the intended signal, but accidentally performs an action on another UI object.

In the first case, the error is related to the identification of signals, the first factor described in Section 5.3.2 as being closely related to speed of performance. In the second case, the error is related to the design of the UI objects and actions. In both cases, the design of the UI objects and actions are involved in influencing the rate of errors because it establishes how signals are identified and how UI objects are manipulated.

Because the **hive view** the offers both the table and the plot, it is difficult to predict the rate of errors. The table in the hive view functions similarly to the lists in the view except that its design is restricted to a smaller space. This can lead to more selection errors as labels are smaller and packed closer together. The plot can lead to many selection errors since the labels are not supplied and identifying a signal by a node requires matching a node from the plot to an item in the list. In spite of its difficulty in identifying signals, the hive view provides an area at the bottom of the screen that shows the currently selected signals. This reduces the chance of creating an error in mapping by having an area for identification that can be checked before the connection is made.

The **list view** was found to provide the quickest means to identify signals, and its design caters to an error-free process for selecting signals. When the data set is small, the rows in the each list expand vertically to fill the available space, making for easily selectable UI objects. When the data set is large, rows are forced into their minimum height and scrolling is enabled. List of sources and destinations can be scrolled independently to place the intended signals into view. The automatic highlighting of elements on mouseover' provides clear feedback to the user and reduces the chance of errors. Once the signals have been correctly identified and selected, the keyboard shortcut creates the connection without any chance of error. Dragging and dropping, however, can increase the rate of errors as it requires precise control of the mouse.

The **grid view** offers a clear selection process through the use of the keyboard shortcuts. The source and destination can be selected independently by pressing the horizontal or vertical arrow keys respectively. The mouse can create more difficulty in the selection process because it requires precise control and forces the identification in both dimensions to be managed simultaneously. The difficulty is compounded by the vertical oriented column labels that are difficult to read. The automatic highlighting of the selected cell and corresponding row, column and labels provide visual feedback and allow for the confirmation of the selected signals before making the connection.

Conclusion

The rate of errors is strongly connected with the ability to identify signals and the manner in which the user interface is manipulated with the keyboard or mouse. Visual feedback during interaction aids in the identification process. Interaction requiring difficult mouse movements can increase the rate of errors. Allowing a clear confirmation of the selected signals before making the connection significantly reduces or eliminates the rate of errors attributed to the UI. Since all views allow the selection to be confirmed before creating the connection, and because further testing would be required to evaluate the success rate of interaction with each interface, the rank ordering of this evaluation criteria was equalized.

Ranking: 1. List, grid, and hive

5.3.4 Retention over time

Monitoring of the different users spanned the course of two semesters as some students at the IDMIL expressed interest in the project and volunteered to test the interfaces out during the end phases of their development. The only evaluations that can be made in terms of retention over time pertain to the short term and are by no means quantifiable.

All the users managed to retain how the interfaces worked without issues. This could be because the fundamental task of creating mappings is relatively simple and straightforward so once the knowledge of how to carry out the task in the user interface domain is learned, the process can easily be remembered.

Furthermore, this category of evaluation was deemed less important than the others. Those who were interested in using the tool to create mappings with DMIs were motivated to learn the interface. Not only are the Webmapper interfaces pretty quick to learn, but once learned the tasks become pretty intuitive and hard to forget. Furthermore, if forgotten, the process of remembering is very quick. It was therefore decided not to evaluate each view in this criterion of evaluation.

5.3.5 Subjective satisfaction

Although completely subjective, users seemed to have an overall preference towards a specific view. In some cases, the preference was divided into using a particular view for a particular task; in one case, a user preferred to use the list view to create the mappings but switched to the grid view as a visualization.

Note that each user had their own unique data set pertaining to their own set of devices. In each case, the data did not surpass a network arrangement of more than 6 devices or 20 signals per device. This limited observations and discussion to smaller data sets, whereas user feedback would most likely be affected by larger data sets.

The overall preference was affected by familiarity with other software applications; one user who was familiar with Max/MSP preferred the list view whereas another who was familiar with digital drum machines preferred the grid view. Some commented how the orderly arrangement of cells in a grid made them feel more at ease when compared to a disorderly display of intersecting lines in the list and hive views. Another user was more interested in the visual representation created by the adapted hive plot, and was willing to work with it in spite of the intersecting lines and lack of labels in the plot. On the other hand, difficulties created by the interface caused one user to abandon a particular view completely; this happened in the grid view with a user became frustrated with the vertical labels. Another enjoyed using the grid for its semblance to the game Battleship, a grid-based puzzle guessing game played between two players, perhaps hinting towards a "fun factor" that motivates a user to overcome difficulties created by the interface.

Conclusion

To configure mappings, users preferred to use a view that offered the best sense of an internal locus of control (i.e., golden rule number 7 from Section 3.3.2). Difficulties created by the interface in performing the fundamental task severely affected this preference. Familiarity with other similar interfaces had a strong influence, along with the fun-factor created by the UI object and actions. The ability to visualize the network had some effect.

Because visualizing the network only supports the fundamental task of creating mappings, final evaluation was based on subjective satisfaction towards the fundamental task of actually creating the mappings, and pertained to small and medium sized data sets only. Further testing would be required, but it is believed that the grid view's ability to organize the mappings into multiple customizable subviews would provide for a better locus of control when the data set is large.

Ranking: 1. List 2. Grid 3. Hive

5.3.6 Ability to Visualize

In the **list view**, the crux of visualization takes place in the middle column between the lists of sources and destinations on either side. Connections are represented in the middle column by curved lines joining elements from each side. The space devoted to this column is less that a third of the available space in the display. This facilitates a simple means to make correspondences between sources and destinations for small and non-complex data sets. However, as the data set grows larger and becomes more complex, connection lines begin to overlap and become difficult (if not impossible) to follow. An example of a complex set of devices is shown in figure 5.1, where the complexity creates a blob of overlapping lines and multiple rows of tabs.

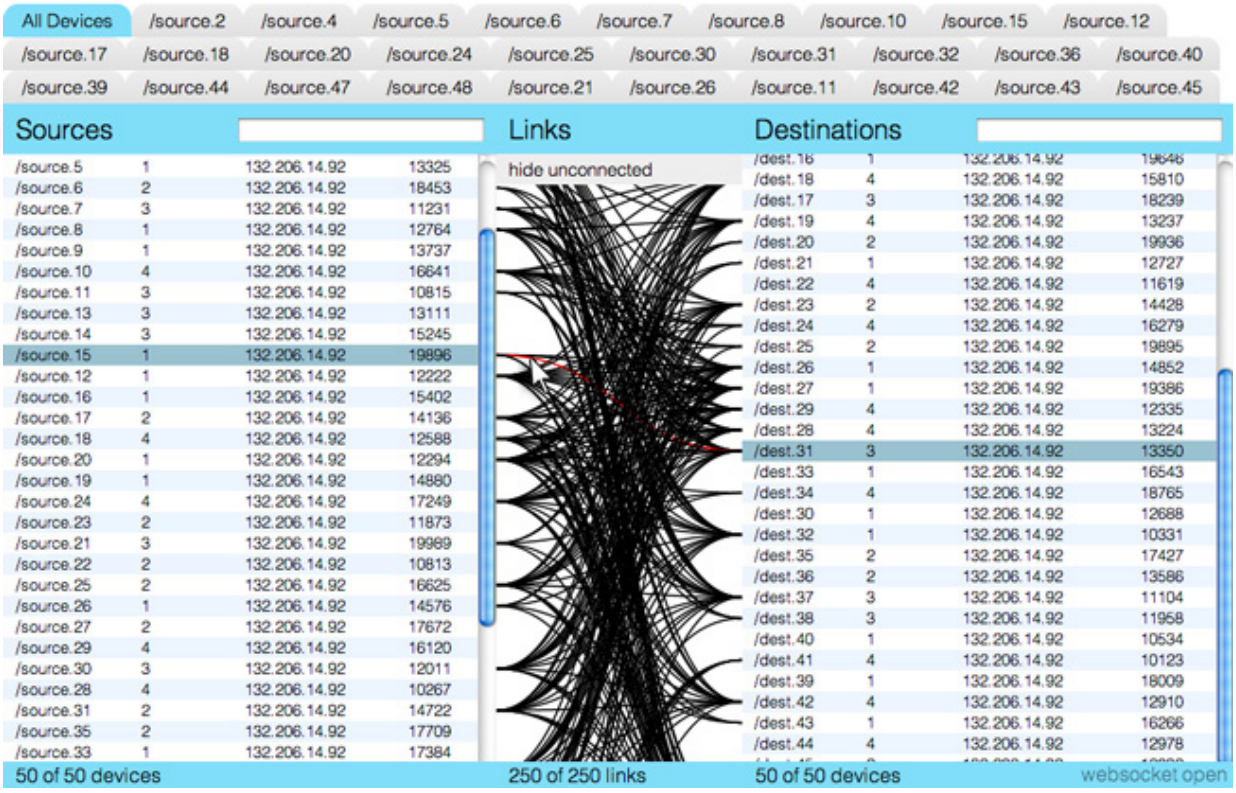


Fig. 5.1 Example of a large and complex data set in the list view. The image shows many-to-many mappings in an arbitrary and generic set of devices.

In the **grid view**, the crux of the visualization takes place in the grid of cells, with approximately half of the available space devoted to it. By prescribing this much space to the visualization, the task of identifying signals becomes more difficult as the labels

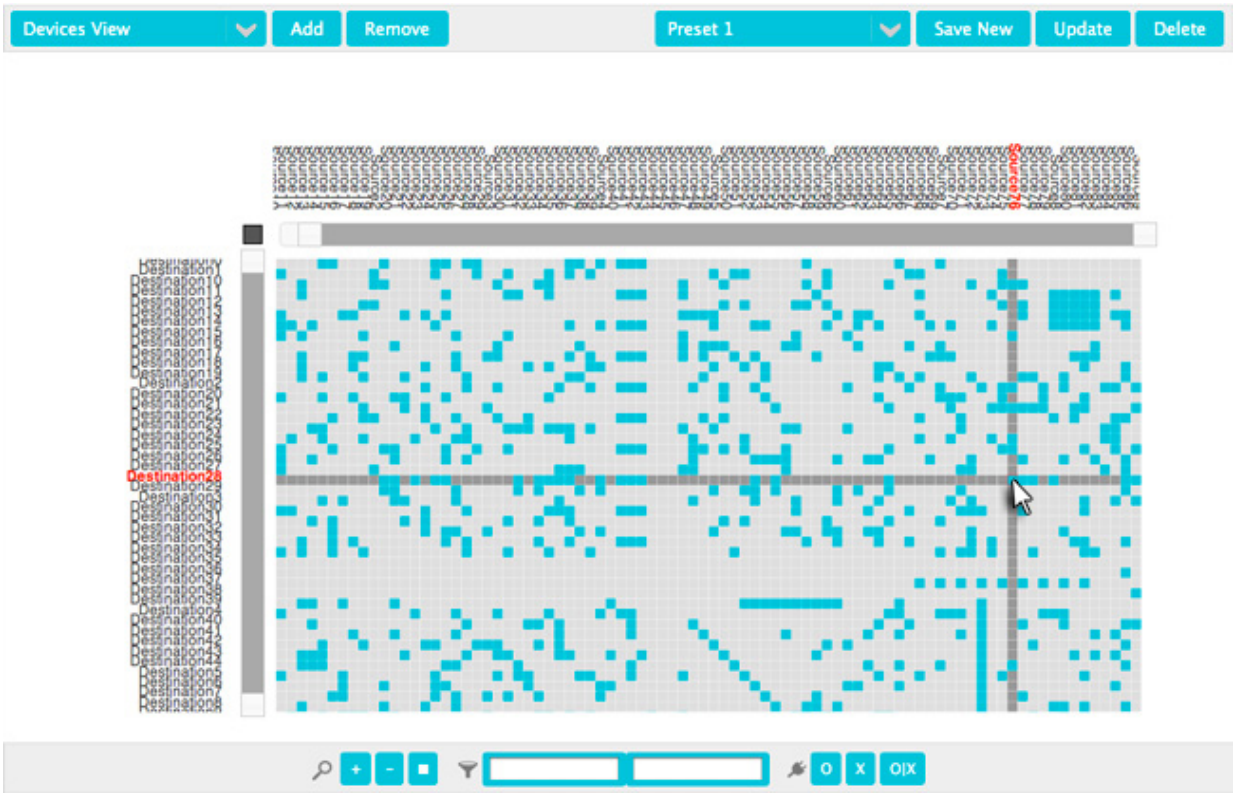


Fig. 5.2 Example of a large and complex data set in the grid view. The image shows many-to-many mappings in an arbitrary and generic set of devices.

compete with the grid for space in the construction. The grid view suffers distinctively from long signal namespaces that will claim more area in the display; it also suffers from requiring identification of labels along two dimensions, with vertical labels that are hard to read. However, this becomes a tradeoff for an enhanced visualization; the grids have an obvious advantage in their ability to represent larger and more complex data sets. The grid eliminates the use of lines that can be difficult to follow and is thus able to represent the maximum size and maximum complexity of data. In certain situations, it can represent the maximum size and complexity in a single image[7]. Although namespace labels become illegible a zoom level that accommodates large data sets[8], this is a tradeoff from the identification task to the visualization task, and interactive features can improve the situation somewhat (e.g., a fisheye zoom on mouseover' of labels, or pulling out the

[7] in the signal grid, it depends on how many devices have been added, and the number of signals pertaining to each added device

[8] approximately 50 elements in either dimension

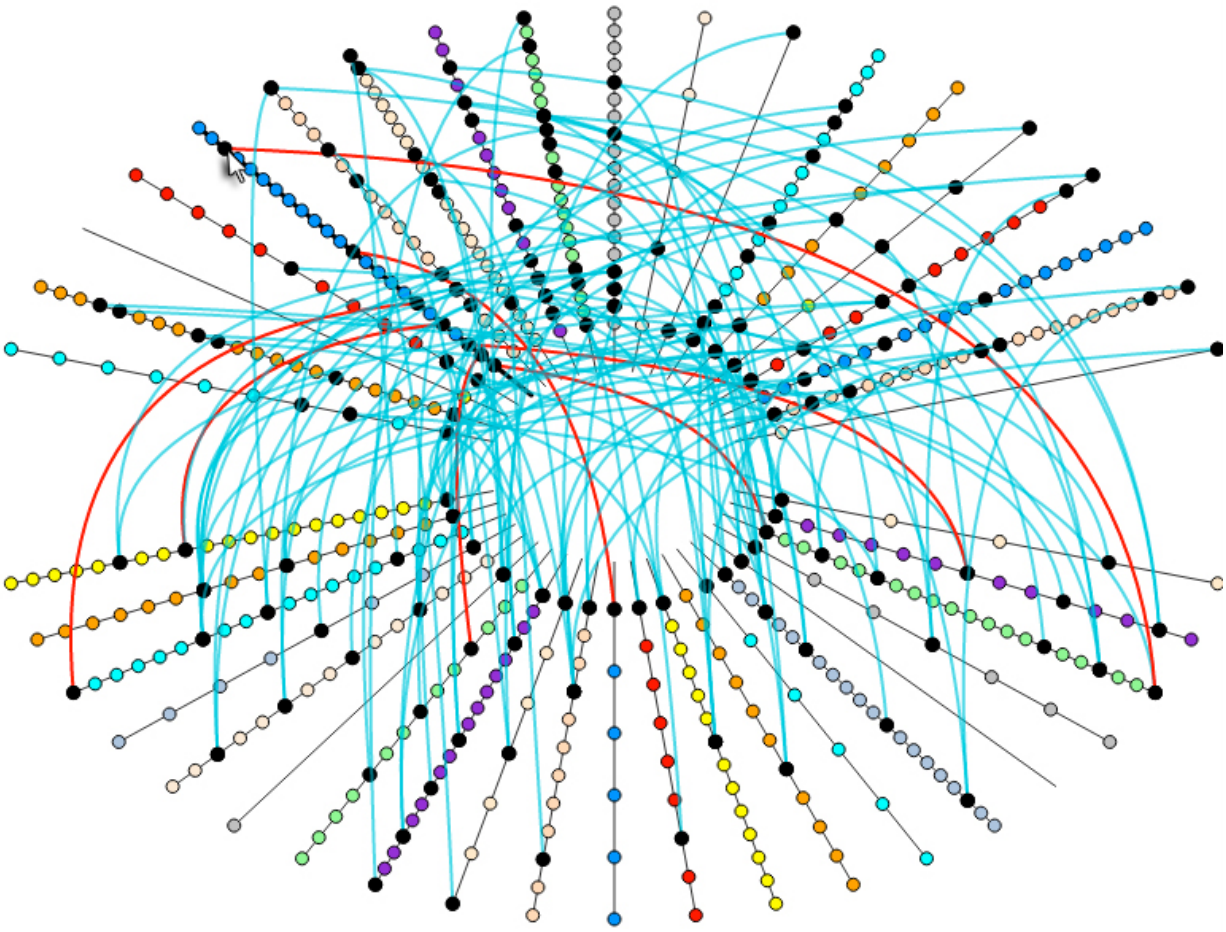


Fig. 5.3 Example of a large and complex data set in the hive view. The image shows many-to-many mappings in an arbitrary and generic set of signals. The interface highlights connections interactively based on the position of the mouse.

corresponding labels on mouse over of a cell). An example large and complex set of devices is shown in figure 5.2. The image demonstrates that the mouse over interactivity aids in identification even when the mappings are complex. Additionally, some patterns of linked cells can be easily identified, such as a sequences of horizontal, vertical, or diagonal lines, a block of cells, repeating patterns, or the absence of links in a set or rows or columns.

The **hive view** dedicates about two-thirds of its available space to the visualization. Both view modes (i.e., the hive plot and the adapted hive plot) serve small to medium data sets but the adapted hive is also suited for large data sets. The adapted hive can

accommodate data sets with maximum size and complexity, and contrary to the grid view, it can support displaying all signals from all devices on the network in a single image. Although the hive represents connections by lines that can be hard to follow, it has other characteristics that make it more geared towards a thriving as visualization: the layout profile of axes and nodes, the use of color codes, and the unique visual signatures created by the plot. Although limited in its speed of performance when carrying out the main task, this becomes a tradeoff for an interesting and powerful visualization. In the final evaluation, the adapted hive plot thrives with small to medium sized and simple data sets, has a larger space devoted to the construction, and has interactive capabilities for highlighting subsets of the visual signature; however, it suffers from the difficulty created by the connecting lines, especially with large and complex data sets. An example large and complex mapping is depicted in figure 5.3.

Conclusion

The ability to visualize was proportional to the increased use of the visual variables from Section 3.1, the size of the space designated to the construction, and its ability to represent larger and more complex mapping descriptions. The ability to visualize competes challenges the ability to identify signals by labels.

Ranking: 1. Grid 2. Hive 3. List

5.4 Summary

This chapter presented the evaluation criteria and then applied the criteria to the evaluation of the three different user interfaces inside Webmapper. Evaluation was based on informal monitoring of and feedback from test users who used the system to configure small to medium sized mappings, along with the author's explorations into using the system with larger sets of generic data. The evaluations were thoroughly described and Figure 5.4 shows a graphic representation of the rankings from each section. Note that the criteria for subjective satisfaction was ranked for small to medium sized data sets only. The results indicate that alternate interfaces provide different methods of interaction and visualization, each with strengths and weaknesses that will cater to different users in different situations.

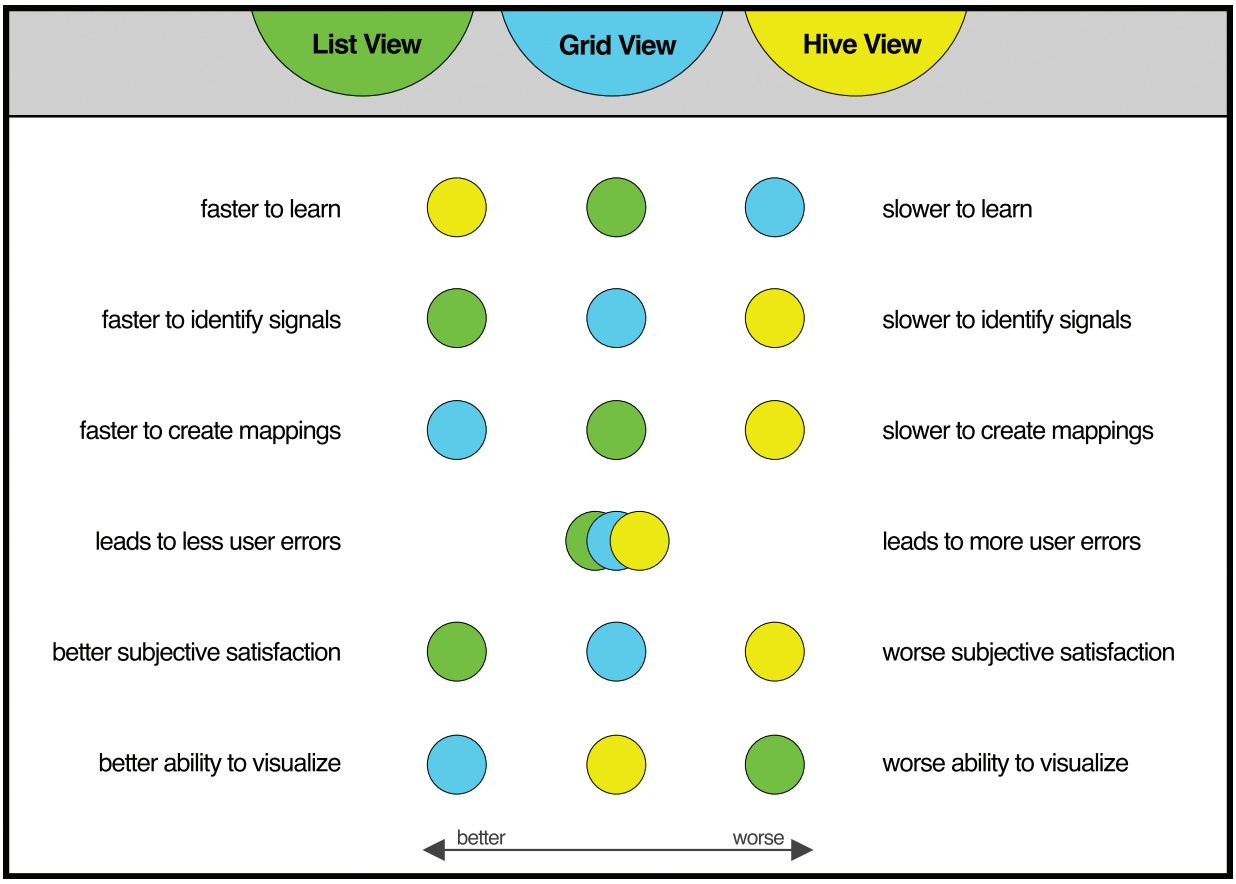


Fig. 5.4 Ranking of the alternate interfaces according to each criterion. Note that ranking for subjective satisfaction was for small to medium sized data sets only.

Chapter 6

Conclusions and Future Work

This chapter summarizes the work presented in this thesis, offers conclusions, and discusses future work and possible research.

6.1 Conclusions

This thesis began by exploring issues relevant to mapping in digital musical instruments. The case for mapping strategies in DMIs began by first looking at the sound production mechanisms in traditional acoustic instruments. Sound generated by acoustic instruments is the result of the physical properties of the instrument itself, and typically the control surface is directly involved. Instruments that were designed with components to enhance playing techniques were then described because they start to create an intermediate layer between the gestures that excite the instrument and the resulting sound. The church organ was presented as an example of an instrument containing an external source of energy and a mapping strategy to connect parts of the instrument's interface with parameters of sound. This example provided a basis towards the description of the three components of a DMI: the sound generating component, the control surface, and the mapping between them. The T-Stick was presented as an example digital musical instrument, with details of its specification of sensors and construction.

Mapping in DMIs was shown to not be a trivial task and it can drastically alter the character of the instrument including how it is played and sounds. The libmapper and Webmapper software mapping tools developed at the IDMIL were presented as the source of motivation for the thesis project, and an extension was created to Webmapper's list-based

GUI by developing two alternate interfaces. The new interfaces were intended to provide an alternate method to configure and to visualize mappings, and their development was methodologically guided by principles of data visualization and the object-action-interface model of user interface design.

The first new interface was based on the construction of a grid to represent connections between pairs of signals. This type of view derived most of its power by being able to represent mappings without the use of connecting lines; lines connecting elements provide a direct and intuitive way to visualize mappings, but as the data set grows larger and more complex they become difficult, if not impossible to follow. The grid is capable of dealing with large data sets of maximum complexity, and although it increased the number of steps required to identify signals in the GUI, it provides a method of interaction similar to other audio applications that some found intuitive and fun to use.

The second interface was created inspired by the Hive Plot, where nodes are placed on radially-oriented linear axes. The construction yielded interesting results by creating a visualization in a simulated three-dimensional space. Although the view was not optimal for identifying signals or configuring connections with larger and more complex data sets, it provided interesting visualization possibilities and opportunities for interaction with the underlying data and its "visual signature".

Evaluation of the interfaces was done through informal monitoring of and feedback from test users that were familiar with the concept of mapping in DMIs. Specific categories of criteria was used for the evaluation, and each of the three interfaces demonstrated different strengths and weaknesses in each category. The results conclude that there is no single "best" interface and that alternate interfaces provide different methods of interaction and visualization that will cater to different users in different situations.

The technical implementation of the Webmapper extension was structured by concepts borrowed from two architectural patterns, and designed to be easily understandable, maintainable, and expandable. In conjunction with previous efforts to develop libmapper and Webmapper, the final architecture lays the foundation for additional interfaces to be added relatively quickly because programming efforts on any new interfaces can focus on higher-level, view-centered concepts and need not be concerned with lower-level libmapper implementation details.

6.2 Future Work

6.2.1 Enhancements to the New Interfaces

Based on the feedback received from users, it is believed that improvements can be made to both the grid and the hive views to circumvent some of their shortcomings that were observed during user testing.

In the grid view, an improvement would be to increase the legibility of the vertical labels by rotating them towards a horizontal orientation. To enhance the grid view's ability to provide custom views on subsets of the system, it can be modified to support showing, hiding, and reordering of individual rows and columns in the signals grid. Other enhancements can include implementing mouse wheel scrolling, dragging and dropping, a full-screen mode for expert users, and improved keyboard control.

The following enhancement is an alternate direction that the adapted hive view can take. By allowing the user to change the length and position of device axes or to define their own color-coding scheme, the plot can be customizable, encouraging a more memorizable layout and "visual signature". This, along with introducing interactive tooltips with signal names, can effectively reduce (or eliminate) the need for the table of signal namespaces.

6.2.2 Hive Plot Node Assignment

As explained in Section 4.4, node coordinates in a hive plot are derived from the absolute or rank-ordered value of a node parameter relating to the network [21]. Although not the route taken in the current Webmapper hive plot, future research can alter the rules that govern the placement of nodes to axes by using graph metrics. This would facilitate investigations into mapping strategies by relating them to structural properties of their graph.

6.2.3 Introducing Orderable Components

Bertin describes the process of "diagonalization" [17] that involves organizing the visual representation of a grid through permutations of rows and columns. The process will enable correspondences to be observed from the resulting image that could range from null correspondence to perfect correspondence, or towards non-variation of one of the dimensions (see figure 6.1).

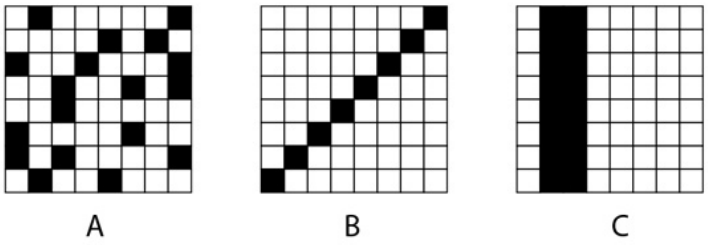


Fig. 6.1 Patterns of null correspondence (A), perfect correspondence (B) and non-variation of one of the dimensions (C) in a reorderable grid.

If orderable or quantitative properties can be assigned to signals (e.g., energy, position, brightness, color, etc.) representations of the mappings can benefit from universal ordering. The grids can be especially interesting because they facilitate making correspondences between two or more components, as shown in figure 6.2. By introducing orderable components into the data set, future research into mapping strategies can utilize data visualization principles made possible by the grids inside Webmapper's grid view.

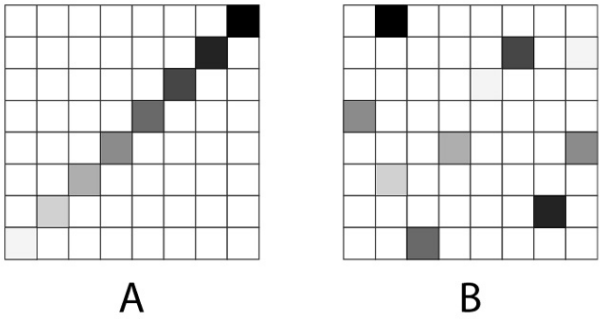


Fig. 6.2 Diagonalization of a reorderable grid with three orderable components. The hypothetical components are represented and ordered by position in each axis and by color value. (A) shows perfect correspondence and (B) shows null correspondence.

6.2.4 Introducing a difference utility

New challenges have emerged as Libmapper has been used in an increasing number of production environments. The main challenge addressed here is the need to compare different mappings to each other. This need became apparent during the production of "Les Gestes" [6], where engineers, performers, and composers were participating in production workshops

and experimenting with a network of DMIs. Due to the complexity of the mappings and the real-time nature of the experimentation, it became hard to pinpoint differences between two different mapping configurations.

Webmapper allows mappings to be visualized, but it can be enhanced with functionality for comparing different maps to each other. An adapted technique of the "diff" file comparison utility (for an example implementation see [27]) is proposed for the comparison of mapping data; the output of the "diff" can then be fed to Webmapper to exploit its potential for visualization.

References

[1] E. R. Miranda and M. M. Wanderley, *New Digital Instruments: Control and Interaction Beyond the Keyboard*. Middleton, Wisconsin: A-R Publications, 2006.

[2] J. Malloch, "A consort of gestural musical controllers: Design, construction, and performance," Master's thesis, McGill University, Montreal, Canada, August 2007.

[3] J. B. Rovan, M. Wanderley, S. Dubnov, and P. Depalle, "Instrumental gestural mapping strategies as expressivity determinants in computer music performance," in *Proceedings of Kansei- The Technology of Emotion Workshop*, (Genova), 1997.

[4] A. Hunt and M. M. Wanderley, "Mapping performance parameters to synthesis engines," *Organised Sound*, vol. 7, no. 2, pp. 97–108, 2002.

[5] 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 *LNCS*, (Berlin / Heidelberg), pp. 401–425, Springer-Verlag, 2008.

[6] "Les gestes / gestures." `http://idmil.org/projects/gestes`. [Aug 2013].

[7] N.-H. Fletcher and T.-D. Rossing, *The Physics of Musical Instruments*. Springer-Verlag, 2nd ed., 1998.

[8] J. Malloch and M. M. Wanderley, "The T-Stick: From musical interface to musical instrument," in *To appear in the Proceedings of the 2007 International Conference on New Interfaces for Musical Expression (NIME07)*, (New York City, USA), 2007.

[9] D. A. Stewart, *Catching Air and the Superman*. PhD thesis, McGill University, Montreal, Canada, February 2010.

[10] A. Hunt, M. Wanderley, and M. Paradis, "The importance of parameter mapping in electronic instrument design," in *Proceedings of the 2002 Conference on New Interfaces for Musical Expression*, pp. 149–154, 2002.

[11] K. C. Ng, "Music via motion: Transdomain mapping of motion and sound for interactive performances," *Proceedings of the IEEE*, vol. 92, pp. 645–655, April 2004.

[12] J. Stewart, *Single Variable Calculus: Early Transcendentals*. Pacific Grove, CA: Brooks/Cole Publishing Company, 2nd ed., 1991.

[13] M. Wright and A. Freed, "OpenSoundControl: A new protocol for communicating with sound synthesizers," in *Proceedings of the International Computer Music Conference*, ICMA, 1997.

[14] M. T. Goodrich and R. Tamassia, *Data Structures and Algorithms in Java*. Chichester: John Wiley & Sons, 4th ed., 2005.

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

[16] 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.

[17] J. Bertin, *Semiology of Graphics: Diagrams, Networks, Maps*. Madison, WI: University of Wisconsin Press, 1983.

[18] C. Ware, *Information Visualization: Perception for Design*. Amsterdam: Morgan Kaufmann, 2nd ed., 2004.

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

[20] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 3rd ed., 1998.

[21] M. Krzywinski, I. Birol, S. J. Jones, and M. A. Marra, "Hive plots—rational approach to visualizing networks," *Briefings in Bioinformatics*, vol. 13, no. 5, pp. 627–644, 2012.

[22] G. E. Krasner and S. T. Pope, "A description of the model-view-controller user interface paradigm in the Smalltalk-80 system," *Journal of Object Oriented Programming*, vol. 1, no. 3, pp. 26–49, 1988.

[23] S. Burbeck, "Applications programming in Smalltalk-80 (tm): How to use model-view-controller (MVC," *Smalltalk-80 v2. 5. ParcPlace*, 1992.

[24] A. Leff and J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in *Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference*, pp. 118–127, 2001.

[25] J. Coutaz, "PAC, an object oriented model for implementing user interfaces," *Proceedings Interact*, vol. 87, no. 2, pp. 431–436, 1987.

[26] P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement," *Journal of Experimental Psychology*, vol. 47, no. 6, p. 381, 1954.

[27] J. W. Hunt and M. D. McIlroy, *An algorithm for differential file comparison.* Bell Laboratories, 1976.