Webmapper: A Tool for Visualizing and Manipulating Mappings in Digital Musical Instruments

Johnty Wang[1], Joseph Malloch[2], Stephen Sinclair[3], Jonathan Wilansky, Aaron Krajeski[1], and Marcelo M. Wanderley[1]

[1] Input Devices and Music Interaction Laboratory, CIRMMT, McGill University
{johnty.wang, aaron.krajeski}@mail.mcgill.ca,
jonathan.wilansky@gmail.com, marcelo.wanderley@mcgill.ca
[2] Graphics and Experiential Media Lab, Dalhousie University
joseph.malloch@dal.ca
[3] Multimodal Simulation Lab, Universidad Rey Juan Carlos
stephen.sinclair@urjc.es

Abstract. This paper describes the motivation, implementation, and usage of the application *Webmapper*, a tool for visualizing and manipulating mappings in the context of digital musical instrument (DMI) design. *Webmapper* is a user interface for interacting with devices on the *libmapper* network, a distributed system for making dynamic connections between signals within discrete devices that constitute a DMI. This decoupling of the mapping as a separate entity allows flexible representation and manipulation by any tool residing on the network—exemplified by *Webmapper*. We demonstrate the capability and potential utility of providing different representations of mappings in the work-flow of DMI design under a variety of collaborative and individual use cases, and present four visualizations applied to mappings used from a previous project as a concrete example.

Keywords: mapping, DMI design, prototyping, collaboration, visualization

1 Introduction

Mapping, in the context of digital musical instruments (DMIs)[1], pertains to the translation of input signals into resultant sound. Since mapping determines the ultimate behaviour of the instrument, it is an important part of the design process and an interesting area of research [2].

This paper describes the motivation, implementation, and usage of the application, *Webmapper*, a tool that supports multiple approaches to visualizing and manipulating mappings in the context of DMI design. First, the contexts which inspired *Webmapper* are introduced including a brief introduction to the *libmapper* framework that provides the underlying connectivity features. Then, the structure and implementation of *Webmapper* is presented using examples of

mappings from projects demonstrating the various visual representations implemented. Finally, an evaluation of the different views are presented, along with a discussion and future work related to the application.

2 Background and Related Work

2.1 Mapping Tools for DMI Design

A number of general-purpose graphical tools, designed for building interactive and multimedia systems, are used by the community to create mappings in the context of DMI design. Some—such as Max[4], Pd[5], and TouchDesigner[6]—provide full programming environments that can be used to define the structure of the entire instrument. These tools not only allow visual representation of the connection and processing of signals that make up part of the mapping process, but also can embed the interfaces to hardware and software components related to the sensor input devices as well as output synthesis systems. In terms of representation, these visual environments provide a signal-flow or "patching" interface that resembles the physical connection of wires in an audio processing chain.

There are also toolboxes and applications dedicated to the process of designing mappings specifically for DMIs. Some examples such as OSCulator[7] and junXion[8], are standalone applications that provide drivers for hardware input devices and allow transmission of signal data to programmed endpoints with data scaling. Other toolboxes provide specific mapping features to an existing environment such as a library of mapping and signal conditioning primitives [3], matrix-based manipulations specifically for mapping [4], mapping between different dimension spaces via geometric representations [5], or creation of mappings via machine learning [6].

One key commonality among all these existing tools is that they provide a single method of representing the mapping. *Webmapper*, on the other hand, allows more than a single way of representing and manipulating the mapping structure. The Jack Audio Connection Kit [9] provides an API that allows applications to access and modify the audio and MIDI connections between virtual endpoints on a local system, which results in the possibility of multiple command-line and GUI tools. However, Jack was designed to work with connections only.

2.2 libmapper

Through working on a number of collaborative projects involving DMIs spanning more than 10 years, a software framework for creating dynamic mappings,

[4] `https://cycling74.com/products/max/`
[5] `https://puredata.info/`
[6] `https://www.derivative.ca/`
[7] `https://osculator.net/`
[8] `http://steim.org/product/junxion/`
[9] `http://jackaudio.org/`

libmapper [7], was developed. Some concepts that prompted the development of *libmapper* include:

- **Experimentation:** The design of DMIs involves many variables such as the selection of sensing components, mapping, and synthesis techniques. These are not standard procedures and the process often involves exploration and experimentation.
- **Diversity:** Since work with DMIs often involves collaborators from different backgrounds, there isn't a single tool or approach that will work for everyone. Fixed representation standards may be limiting.
- **Distributed control:** Under collaborative contexts, it may be useful to allow multiple users to view and modify the mapping configuration at the same time.

To facilitate experimentation, it is necessary to provide the ability for connections between components to be quickly created and modified. The diversity of users suggest it may be useful to provide more than a single view and interaction method on the state of the system. Concurrency implies the need for a network based model where more than a single user can access and manipulate data at the same time. As a result, *libmapper* was developed as a framework upon which more modular and flexible approaches to mapping design can be realized. At its core, *libmapper* as a software library provides the means to expose a device to a network that allows automated discovery and dynamic connection with signal conditioning built into the connection itself.

The fundamental components on the *libmapper* network are *devices* and *signals*. An input device may contain a number of output signals which may for example be values corresponding to sensors intended to measure a set of gestures, and an output device such as a synthesizer will feature input signals that correspond to control parameters that affect the generated audio. *libmapper* allows *links* to be made between devices which construct high level associations between devices, and *maps* which are dataflow connections between parameters of interest. Another key feature of *libmapper* is that some basic signal conditioning can be built into the connection itself so that commonly used methods such as scaling, clamping, and basic filtering can be added.

Bindings for *libmapper* exist for many popular programming languages and include C/C++, Java, Python, and Node.js. External objects for Max and Pure Data are also available.

3 Webmapper

Two other GUI applications, *Maxmapper* and *Vizmapper* existed prior to the development of *Webmapper*. The former was an interface implemented in the Max environment that provides display and manipulation of connections on the network based on a list representation, and served as the seminal example of a usable graphical tool to view and manipulate devices on the network. The latter was an exploration of alternative representations of larger and more complex

networks [8], and its existence also demonstrated that a different visual representation of the mapping can be running concurrently due to the distributed nature of the network. A basic command-line application was also made to manipulate and view mappings on the *libmapper* network[10].

Webmapper, as its name implies, is a browser-based application. Originally, the motivation to build a web-based application was to provide the ability to run the user interface within a browser on a variety of desktop and mobile platforms. Additionally, the frameworks and libraries for modern web development platforms support scalable development and deployment of visual user interfaces.

A highlight of *Webmapper* is that it provides more than a single view on the mapping structure, which allows multiple, as well as concurrent visual representations.

3.1 Architecture and Implementation

Webmapper is implemented using a Python back-end that serves two main functions. First, the server provides interfaces to the *libmapper* network and allows querying and modification of the state of running devices. Second, the server hosts the front-end HTML/JavaScript content and synchronizes the state of the network with the user interface. An overall architecture of *Webmapper* is shown in Figure 1.

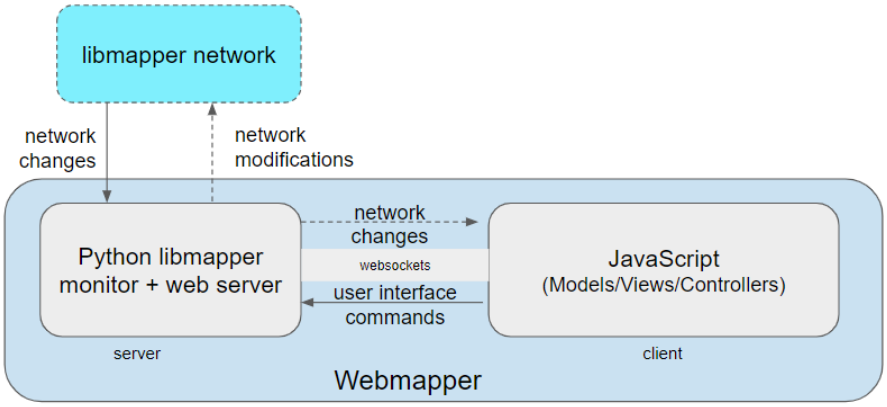

Fig. 1. Webmapper Architecture

[10] `https://sourceforge.net/projects/umapper/`

3.2 Views

The multiple views implemented in *Webmapper* were based on prior tools that had already been developed, as well as graphical design considerations pertaining to the correlation of properties of the network to various visual dimensions [9]. Each view provides a different method of visualizing and modifying the connections between devices and signals. Creation and modification of mappings are implemented via graphical input methods such as drag and drop between the visual elements, click to select, and keyboard shortcuts for removing connections. The full list of possible interactions for each view is described in [[9], chapter 4].

The following is a description of each view, followed by a visual demonstration example. The examples were created using saved mapping configuration files from a previous project, *Les Gestes: une nouvelle génération des instruments de musique numérique pour le contrôle de la synthèse et le traitement de la musique en performance par les musiciens et les danseurs*[11], a collaborative research project directed by Sean Ferguson and Marcelo Wanderley at McGill University and choreographer Isabelle van Grimde from the Montreal-based dance company Van Grimde Corps Secrets[12]. This project involving multiple wearable interfaces and a modular software synthesis system. At the time when these mappings were created, *Webmapper* had not yet been implemented so the only view available was the list based representation provided by *Maxmapper*. In this sample mapping there are two input devices connected to three output modules. The two input devices are identical wearable DMIs worn by dancers, and they control different parameters of three output devices simultaneously. One receiving device, a spatializer controller, is controlled by both input devices while two synthesizers are driven by each input device independently. Each view implemented allows a different way of visualising the connections in the network.

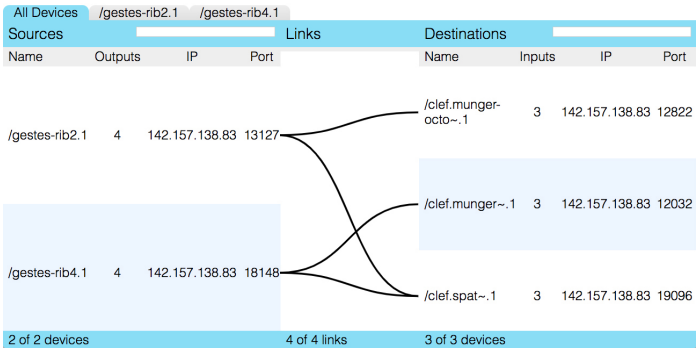

Fig. 2. List View device connections

[11] Gestures: a new generation of digital musical instruments for controlling synthesis and processing of live music by musicians and dancers

[12] `www.vangrimdecorpssecrets.com/`

List View The List View, one of the most direct ways of visually representing connections, simply provides a bipartite graph showing source devices on the left and destination devices on the right. Lines with arrowheads connect between source and destinations. Once a link is made through dragging and dropping between a source and destination device, a new tab window is created for the source device that allows signal to signal mappings to be made. Figure 2 shows the two input devices connected to 3 output devices. Here we can see that input device 1 is connected to output devices 1 and 3, while input device 2 is connected to output devices 2 and 3.

Selecting the tab window of input devices, we see how the individual signals are connected to the synthesizer inputs, as shown in Figure 3 left and right for the two devices.

The main advantage of the list view is that it lists all the connections between devices and signals at once. However, when there are a lot of connections, the visualization can become cluttered very quickly.

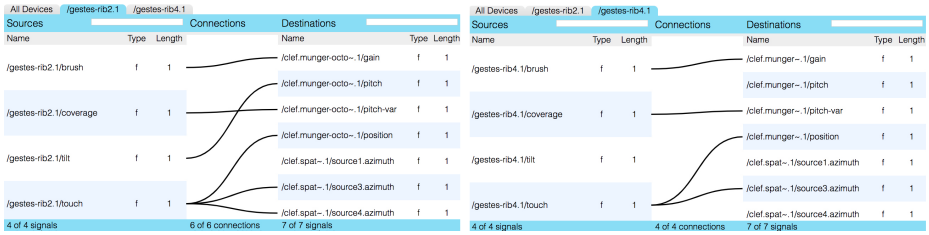

Fig. 3. List View signal connections for input device 1 and 2

Grid View In this view, inspired by the EagenMatrix[13] application, the network is represented by two grids. The left grid lists source devices on the horizontal axis and destination devices are on the vertical axis. Intersection points, if filled in blue, show the existence of links between devices. Devices must be added to the right grid to show their signals and connections; vertical/horizontal lines indicate the device has been added, and the right grid provides a similar representation for signals and connections. In Figure 4, only the first input and first output devices have been added. In figure 5, all devices have been added. The grid view is equipped with the ability to save view configurations into presets, allowing you to switch quickly between them.

One advantage of the Grid view is that, unlike the List View, a large number of connections will still be legible since there are no overlapping lines used to represent each link.

[13] `http://www.hakenaudio.com/Continuum/eaganmatrixoverv.html`

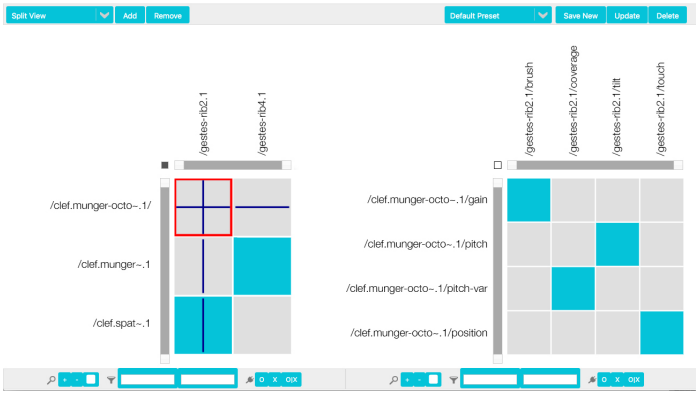

Fig. 4. The Grid View, showing connections between two devices

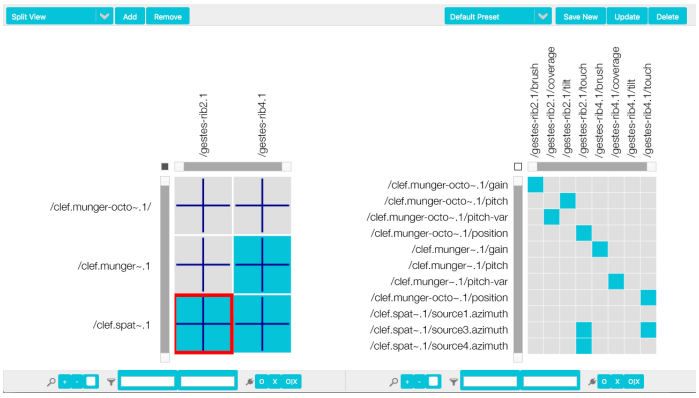

Fig. 5. The Grid View, showing connections between multiple devices

Balloon View Based on the tool implemented in [8], this view displays signals as a nested hierarchy—generated from textual analysis of the signals' OSC address URLs—of smaller circles within a larger one representing the device. Like the Grid View, it allows multiple source devices to be displayed at the same time (Figure 6).

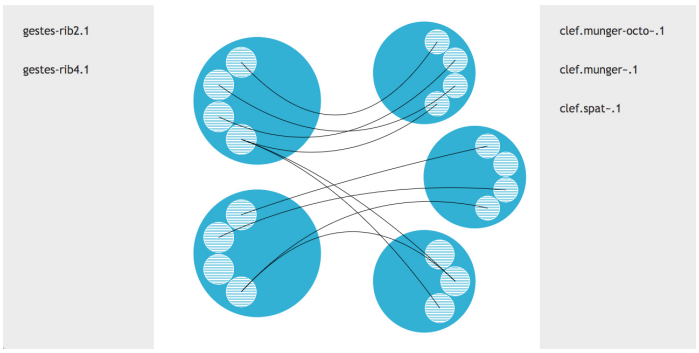

Fig. 6. The Balloon View

Unique to the Balloon View, a device can be selected by clicking inside the circle causing a "zoom" into the device, which then shows the individual signals as larger circles and lists the individual signals of the device on the side legend, providing further levels of detail. Figure 7 shows an input device 1 selected, followed by output device 1 in Figure 8. Clicking on the top of the device labels will "zoom out" of the selected device.

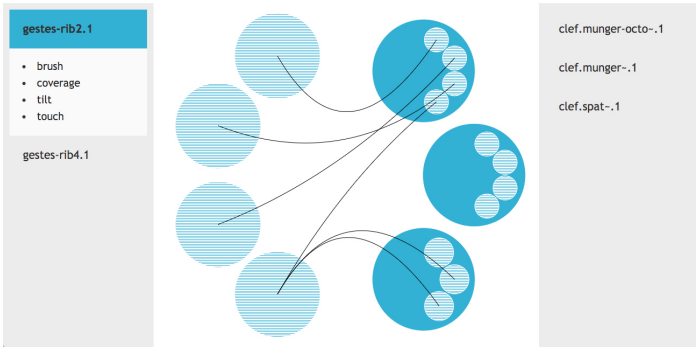

Fig. 7. Zooming in on a source device in the Balloon View

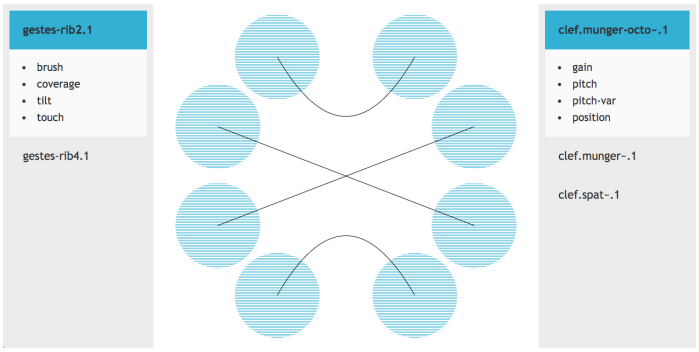

Fig. 8. Zooming in on both the source and destination

Hive View This view displays each device on a single axis, with nodes representing each signal. Lines between nodes on different devices show connections between signals. This view, like the Balloon View, allows multiple source devices to be displayed at the same time.

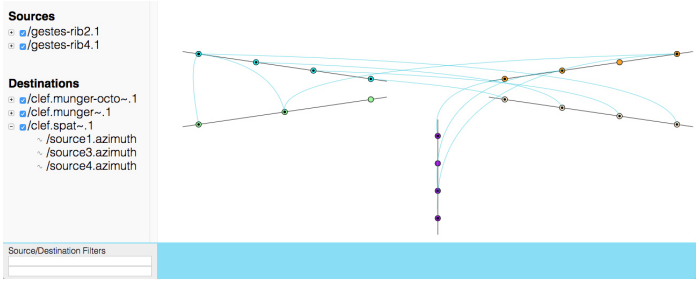

Fig. 9. The Hive View

Similar to the selective rendering features of other views, it is possible to filter out devices and signals. In the Hive View the selection is performed using checkboxes on the left. Signal names are displayed at the bottom of the screen when a particular connection is selected. Figure 10 shows these two features.

Connection Editor The above views provide different methods for visually representing and working with the basic signal connections. Since *libmapper* also provides processing built into the connection itself, a separate interface was implemented to display and modify the mapping expression and range parameters, as shown in Figure 11. This interface allows the signal processing features of *libmapper* to be viewed and modified. A basic interface for saving and restoring

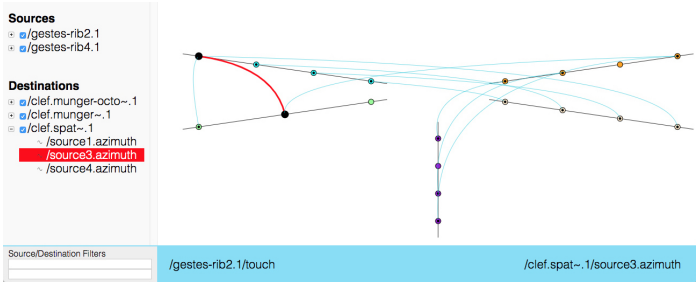

Fig. 10. Selective rendering and labeling in the Hive View

a mapping is also provided. The mapping configurations are saved as JSON[14] files, and when loaded, *Webmapper* will attempt to restore the recorded mapping connections onto the network by triggering the associated *libmapper* commands.

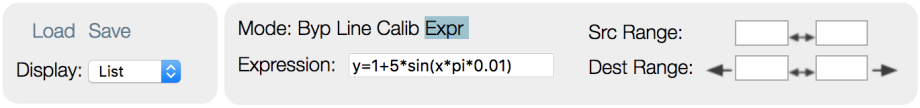

Fig. 11. The load/save and connection editor interface, showing an arithmetic expression applied to the connection.

3.3 User Evaluation

An informal user evaluation was performed on three different views (List, Grid, and Hive)[15] using measurable factors of *time to learn, speed of performance, error rate, and subjective satisfaction* [10]. Three users, who were researchers in the field of DMI design, participated in the evaluation. In the experiment the users were asked to create a mapping for DMIs that they were prototyping using the List View, erase it, and then recreate it using the Grid and then Hive views. The users were observed by the experimenter and followed up with a discussion to investigate the observations and collect general user feedback. A more comprehensive description and discussion on the evaluation is presented in Chapter 5 of [11]. Table 1 contains a summary of the rankings (1=best, 3=worst) of each view for the measured metrics.

The evaluation shows that the alternative interfaces provide quite different methods of interaction and visualization with strengths and weaknesses in different situations. For example, the Hive View was easiest to understand as entire

[14] JavaScript Object Notation
[15] The Balloon View was not yet implemented at the time of evaluation.

Table 1. Measurable human factors for each view

Metric	List	Grid	Hive
Time to learn	2	3	1
Speed	1	2	3
Error rate	1	2	3
Subjective Satisfaction	1	2	3
Ability to visualize	3	1	2

devices and signal connections were presented at the same time, but it was much harder to find a particular signal since it required selecting a signal node before the name of the signal can be revealed. The List View, on the other hand, provided the most straightforward visual representation of connections, but due to the separate tabs for each output device, did not provide for a easy way to obtain an overall picture of the network without switching tabs.

4 Discussion and Future Work

The results of the described evaluation provide a starting point in showing the differences between each view based on a limited number of metrics for a single user performing very specific tasks. In order to further justify the original motivations, additional in depth evaluations should be done to qualify the success in which these different representations have on the original motivations of supporting experimentation and diversity of users. Since the system provides multiple, concurrent representations of the network, the third goal of distributed control is therefore fulfilled by the nature of the implementation.

In terms of the visualizations themselves, thus far we have only implemented the representation of the mapping structure. However, the process of DMI design goes beyond just the connection between signals at a certain point in time: For example, how the mappings are modified over time, as well as the actual signals transmitted through the network are additional factors worth consideration. Integration of version control [12] into the tool as well as live signal visualization (in tools like OSCulator) and analysis can be useful for recalling the temporal progression of the design process, and afford further insight on the current state of the system, respectively.

5 Conclusion

In this paper we have presented *Webmapper*, a visual tool for viewing and manipulating mappings in the context of DMI design. By providing a visual interface for devices and signals on the *libmapper* network with multiple representations and interaction methods, we aim to support different DMI design concepts and workflows, especially in collaborative contexts. It should be stressed that although the views implemented in this application were motivated by specific

perspectives, the overall framework is intended to support the creation of tools to fit a diversity of perspectives. With completely open-source code and a multitude of language bindings it is relatively easy for developers to build additional tools for *libmapper*, and the modular nature of *Webmapper* supports the rapid addition of new views and manipulation strategies. We hope that further usage of the ecosystem demonstrated through the implementation of *libmapper* and *Webmapper* will lead to the development of more interesting perspectives on mapping, both through internal features as well as new tools and use-cases.

References

1. Miranda, E.R., Wanderley, M.M.: New digital musical instruments: control and interaction beyond the keyboard. Volume 21. AR Editions, Inc. (2006)
2. Hunt, A., Wanderley, M.M., Paradis, M.: The Importance of Parameter Mapping in Electronic Instrument Design. Journal of New Music Research **32**(4) (2003) 429–440
3. Steiner, H.C.: Towards a catalog and software library of mapping methods. In: Proceedings of the 2006 Conference on New Interfaces for Musical Expression. NIME '06, Paris, France, France, IRCAM - Centre Pompidou (2006) 106–109
4. Bevilacqua, F., Müller, R., Schnell, N.: MnM : a Max / MSP mapping toolbox. In: Proceedings of the International Conference on New Interfaces for Musical Expression. (2005)
5. Van Nort, D., Wanderley, M.M., Depalle, P.: Mapping Control Structures for Sound Synthesis: Functional and Topological Perspectives. Computer Music journal **38**(3) (2014) 6–22
6. Fiebrink, R., Trueman, D., Cook, P.: A metainstrument for interactive, on-the-fly machine learning. In: Proceedings of the International Conference on New Interfaces for Musical Expression. (2009) 280–285
7. Malloch, J., Sinclair, S., Wanderley, M.M.: Distributed tools for interactive design of heterogeneous signal networks. Multimedia Tools and Applications (2014) 1–25
8. Rudraraju, V.: A Tool for Configuring Mappings for Musical Systems using Wireless Sensor Networks. Masters thesis, McGill University (2011)
9. Krajeski, A.H.: A Flexible Tool for the Visualization and Manipulation of Musical Mapping Networks. Masters thesis, McGill University (2013)
10. Shneiderman, B.: Designing the User Interface: Strategies for Effective Human-Computer Interaction. 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1997)
11. Wilansky, J.: A Software Tool for Creating and Visualizing Mappings in Digital Musical Instruments. Masters thesis (2013)
12. Wang, J., Malloch, J., Chevalier, F., Wanderley, M.: Versioning and Annotation Support for Collaborative Mapping Design. In: Sound and Music Computing Conference. (2017)