Direct Manipulation for Musical Interfaces Based on Visual Software: Design and Evaluation

Jeronimo Barbosa

Direct Manipulation for Musical Interfaces Based on Visual Software: Design and Evaluation

Jeronimo Barbosa

Department of Music Technology
Input Devices and Music Interaction Laboratory
McGill University
Montreal, Canada

July 2019

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

2019/07/09

Abstract

With the popularization of mobile computing and software-distribution platforms like the Apple Store and Google Play, a plethora of remarkable *software-based musical interfaces with dynamic visual feedback* are being designed today. Recent examples such as the Loopy, the Borderlands, the Ocarina illustrate that powerful *visual interfaces* are capable of captivating amateur and expert musicians alike by making *music manipulation* more *direct and straightforward*.

However, while successful examples are available, effectively designing such direct interfaces remains challenging, typically proceeding "as more art than science". For example, unlike acoustic instruments, sound production is physically decoupled from input controls, yielding infinite possibilities for mapping. Moreover, increasing ease-of-use for novices can lead to toy applications that disinterest experts. Finally, analyzing successful crafts remain unusual, and little music interface research is being carried to better understand these interfaces.

This thesis attempts to understand these musical interfaces better, aiming at providing empirical evidence to inform their design and evaluation. Borrowing from human-computer interaction research, I hypothesize that one effective strategy concerns adopting an interaction model known as direct manipulation–characterized by continuous visual representation of the object of interest, physical actions to manipulate the visual representations, rapid visual feedback, and incremental and easily reversible actions.

To investigate this hypothesis, I report exploratory case studies where direct manipulation is designed and evaluated in three specific contexts of music tools: a) *live looping tools*; b) *tools for creating interactive artistic installations*; and c) *music authoring tools*. As a result, I introduce three novel software music tools: *Voice Reaping Machine*; *ZenStates*; and *StateSynth*. Finally, I present a set of interface design strategies and evaluation approaches for effectively designing direct manipulation in musical interfaces based on visuals.

Sommaire

Grâce à la popularisation des plateformes d'informatique mobile et de distribution de logiciels, telles que l'Apple Store et Google Play, de nombreuses *interfaces musicales basées sur des logiciels avec retour visuel* sont en cours de conception. Des exemples récents tels que Loopy, Borderlands et Ocarina montrent que de puissantes *interfaces logicielles visuelles* sont capables de captiver autant les musiciens amateurs que les experts, tout en fournissant une *manipulation musicale* plus *directe*.

Bien que certains exemples soient disponibles, il demeure difficile de conceptualiser de telles interfaces directes, puisque l'accent est porté davantage sur "l'art plutôt que la science". Par exemple, contrairement aux instruments acoustiques, la production du son est physiquement découplée des commandes d'entrée, offrant ainsi des possibilités infinies de mappage. De plus, la facilité d'accès et d'utilisation grandissante pour les novices peut conduire à des applications ressemblant plutôt à des jouets, ce qui désintéresse les experts. Enfin, l'analyse des interfaces musicales existantes reste inhabituelle et peu de recherches sont menées afin de mieux comprendre ces interfaces.

Cette thèse a pour objectif de mieux comprendre ces interfaces musicales dans le but de fournir des bases empiriques qui informeront les principes entourant leur conception et leur évaluation. Tout en me basant sur les recherches conduites dans le domaine de l'interaction personne-machine, je propose l'hypothèse qu'une stratégie efficace consiste à adopter un modèle d'interaction appelé manipulation directe (MD) caractérisé par une représentation visuelle continue de l'objet en question, des actions physiques pour manipuler les représentations visuelles, un retour visuel rapide, et des actions incrémentales facilement réversibles.

Afin d'étudier cette hypothèse, je présente des études de cas exploratoires au sein desquelles la MD est conçue et évaluée selon trois contextes spécifiques reliés à des outils musicaux: a) *outils de bouclage en temps réel*; b) *outils de création d'installations artistiques interactives*; et c) *outils de création musicale*. En conséquence, je présente trois nouveaux outils logiciels de musique: le *Voice Reaping Machine*; le *ZenStates*; et le *StateSynth*. Enfin, je présente un ensemble de stratégies de conception d'interface et d'approches d'évaluation permettant de conceptualiser efficacement la MD pour les interfaces musicales logicielles.

Acknowledgements

This thesis would not exist without the invaluable support of many people. I am immensely grateful to:

- My advisor, Dr. Marcelo M. Wanderley, and my co-advisor, Dr. Stéphane Huot;
- All music technology faculty and my comprehensive exams committee–Dr. Catherine Guastavino, Dr. Gary Scavone, Dr. Isabelle Cossette, and Dr. Eleanor Stubley–, for their insightful comments and early feedback;
- Darryl Cameron and Yves Méthot, for their technical support with the prototypes and setting up experiments;
- All collaborators–especially Dr. Sofian Audry and Dr. Chris Salter–for the opportunity and the exchange;
- Current and past colleagues from IDMIL, Music Tech Area, and CIRMMT;
- My friends in Canada, for the daily support and encouragement, and in Brazil, for their wisdom and endless inspiration. May many carnivals be ahead of us;
- My family, Mamadi, Papadi, and Bocodi, for their love and unconditional support;
- Kamilla de Souza, for the unspoken. What else to say?

I dedicate this thesis to the memory of my old grandpa, Macário Costa. *"O sorriso na chegada da fazenda de meus filhos e amigos, que adoro e quero bem"*.

Contribution of Authors

This document is organized as a manuscript-based thesis, gathering five different papers written during my PhD studies. Overall, it presents two key contributions: 1) Three novel software music tools: the *Voice Reaping Machine*; the *ZenStates*; and the *StateSynth*; and 2) A set of interface design strategies and evaluation approaches for effectively designing direct manipulation in software-based visually-oriented musical interfaces. Following, I detail contributions per manuscript, describing my individual contributions.

Chapter 2

`Jeronimo Barbosa`, Joseph Malloch, Stéphane Huot, and Marcelo M. Wanderley. *"What does 'Evaluation' mean for the NIME community?"* In Proceedings of the 2015 International Conference on New Interfaces for Musical Expression, Baton Rouge, USA.

This chapter presents an exploratory survey on the usage of evaluation studies within musical interface design. Here, I have carried both research and writing. Collaborators have contributed by discussing strategies for data analysis, discussing results, and reviewing early versions of the text.

Chapter 3

`Jeronimo Barbosa`, Marcelo M. Wanderley, and Stéphane Huot. *"Exploring Playfulness in NIME Design: The Case of Live Looping Tools."* In Proceedings of the 2017 International Conference on New Interfaces for Musical Expression, Copenhagen, Denmark.

This chapter describes a case study on using design rationale as process for creating playful-oriented NIMEs–in particular, applied to the specific context of Live looping. A second contribution outcomes from this process: a visual direct manipulation-based live looping tool called the *Voice Reaping Machine*. Here, I have carried all the design research, technical development, and

writing. Collaborators have contributed by reviewing early versions of the design space and the guidelines, as well as providing feedback during the entire manuscript writing.

Chapter 4

`Jeronimo Barbosa`, Sofian Audry, Christopher L. Salter, Marcelo M. Wanderley, and Stéphane Huot. *"Designing Behavioral Show-Control for Artistic Responsive Environments: The Case of the 'Qualified-Self' Project"*.

This chapter contributes with a design study investigating the composition practice of artistic responsive environments, driving the development of two visual behavioral show-control systems (one of them is ZenStates, introduced on Chapter 5). A second contribution concerns lessons learned from an exploratory real-world application of the ZenStates in a professional artistic piece. My individual contributions are four-folded: (1) defining and carrying out the design study; (2) the technical development of the ZenStates prototype; and (3) carrying out the preliminary real-world application; and (4) most of the manuscript writing. Collaborators have contributed with: discussing and taking part of the design study; carrying out the technical development of the ZenTrees prototype; providing feedback on early versions of the ZenStates; and writing parts of the manuscript–especially the sections 'Introducing ZenTrees' and final paragraphs of 'Related Works'.

Chapter 5

`Jeronimo Barbosa`, Marcelo M. Wanderley, and Stéphane Huot. *"ZenStates: Easy-to-Understand Yet Expressive Specifications for Creative Interactive Environments."* In 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Lisbon, Portugal, 167–175. `https://doi.org/10.1109/VLHCC.2018.8506491`

Building upon the last chapter, this chapter introduces a visual specification model for creative interactive environments–the *ZenStates*–and a direct manipulation-based software interface that validates this model. Here, I have carried all research, technical development, evaluation studies, and writing. Collaborators have contributed by discussing the user study protocol, parts of the data analysis, and reviewing early versions of the text.

Chapter 6

`Jeronimo Barbosa`, Marcelo M. Wanderley, and Stéphane Huot. *"Instrumental Music Skills as Input Primitive for Visual Creative Authoring"*.

This chapter proposes the design of interactive music authoring tools centred on instrumental skills of traditional musical instruments. The result is a visual direct manipulation-based software interface called *StateSynth*, that builts upon ZenStates specification model. Here, I have carried all research, technical development, qualitative evaluation, and writing. Collaborators have contributed by discussing the evaluation and reviewing early versions of the text.

Contents

List of Figures

List of Tables

List of Acronyms

CI	Confidence Interval
DIY	Do it Yourself
DM	Direct Manipulation
DMI	Digital Music Instrument
EUP	End-User Programming
GUI	Graphical User Interfaces
HCI	Human-Computer Interaction
LL	Live looping
NIME	New Interfaces for Musical Expression
VRM	'Voice Reaping Machine', name of the live looping tool I have developed.
WIMP	"Windows, Icons, Menus, and Pointers"

Chapter 1

Introduction

> *"With the advent of computers, many of music's past restrictions can begin to fall away, so that it becomes possible for more people to make more satisfying music, more enjoyably and easily, regardless of physical coordination or theoretical study, of keyboard skills or fluency with notation. This doesn't imply a dilution of musical quality. On the contrary, it frees us to go further, and raises the base-level at which music making begins. It lets us focus more clearly on aesthetic content, on feeling and movement in sound, on the density or direction of experience, on sensuality, structure, and shape — so that we can concentrate better on what each of us loves in music that lies beyond the low level of how to make notes, at which music making far too often bogs down"* (Spiegel 1986)

The quote above dates back to the mid 80s. Still, it could have been written yesterday to describe a whole new generation of remarkable musical interfaces based on visual software. With the popularization of mobile computing and software-distribution platforms like the Apple Store and Google Play, a plethora of these interfaces are being designed today. Recent examples such as the Loopy (Tyson 2009), the Borderlands (Carlson 2012), the Ocarina (Wang 2014), illustrate that powerful *visual interfaces* are capable of captivating amateur and expert musicians alike by making *music manipulation* more *direct and straightforward*.

This thesis investigates computational systems like these: *software interfaces* that explore *dynamic visual feedback* as a strategy to make *music interaction* more *direct and straightforward*.

While successful examples are available, effectively designing such interfaces remains challenging, proceeding "as more art than science" (Cook 2017) for many reasons. For example, sound production in digital musical instruments is physically decoupled from input controls, yielding infinite possibilities for mapping. Yet, choosing the right mapping strategy is essential for a successful instrument (Hunt, Wanderley, and Paradis 2003). Moreover, increasing ease-of-use for novices can yield "toys" that disinterest experts. Conversely, sophisticated controls that might engage experts can also scare away novices (Jordà 2003; Wessel and Wright 2002). Finally, analyzing successful examples–which were able to overcome all the challenges–remain unusual, requiring further musical interface research to understand these examples better.

How may one successfully overcome these challenges? In other words, how can we support the design of new software interfaces so that visual feedback may foster direct music interaction?

This thesis attempts to shed some light to this question. Ultimately, it aims at providing some empirical evidence to inform the design and evaluation of new musical interfaces based on visual software.

Borrowing from Human-Computer Interaction (HCI) research, I hypothesize that one effective strategy concerns adopting a visual interaction style known as Direct manipulation (Shneiderman 1983; Shneiderman et al. 2010).

1.1 Direct manipulation

Direct manipulation (DM) is an interaction style introduced by Ben Shneiderman (1983; 2010) to describe visual interfaces, such as video games and spreadsheets, that highly contrasted with the textual command-line interfaces that were back then predominant in human-computer systems. DM is characterized by:

- *"Continuous representation of the object of interest;*

- *Physical actions (movement and selection by mouse, joystick, touch screen, etc.) or labeled button presses instead of complex syntax; and*
- *Rapid, incremental, reversible operations whose impact on the object of interest is immediately visible".*

Shneiderman argues that DM can bring the following benefits:

- *"Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user;*
- *Experts can work rapidly to carry out a wide range of tasks, even defining new functions and features;*
- *Knowledgeable intermittent users can retain operational concepts;*
- *Error messages are rarely needed;*
- *Users can immediately see if their actions are furthering their goals, and, if the actions are counterproductive, they can simply change the direction of their activity;*
- *Users experience less anxiety because the system is comprehensible and because actions can be reversed so easily;*
- *Users gain confidence and mastery because they are the initiators of action, they feel in control, and the system responses are predictable".*

One representative example of DM are *"Windows, Icons, Menus, and Pointers"* (WIMP) Graphical User Interfaces (GUI). In WIMP GUIs, users interact with their object of interest by manipulating a diversity of visual control elements such as windows, buttons, checkboxes, dropdown lists, dialog boxes, among others. Examples include the Xerox's Star operational system (Smith et al. 1982), along with its contemporary counterparts, and the Microsoft Word text editor (Shneiderman et al. 2010), making DM one of the most influential interaction styles in commercial human-computer systems.

Many researchers have proposed accounts for the efficacy of DM interfaces. Here we present some of them–focusing on the most relevant according to our objectives and context.

1.1.1 Syntactic and semantic knowledge

Shneiderman uses the syntactic/semantic model of user behavior to explain the power of DM (Shneiderman 1983).

On the one hand, *syntactic knowledge* relates to the arbitrary syntax that requires memorization for user interaction. One example is the syntax of programming languages, which are often unique: The same software (e.g., a calculator) can be implemented in multiple different languages, each one with a unique syntax (e.g., one can implement a calculator in either C, Python, or Java). To code the calculator, one needs to memorize the specific syntax and to articulate instructions in terms of this specific syntax.

On the other hand, the *semantic knowledge* concerns context-specific concepts related to the domain where the interaction occurs. This knowledge would hierarchically increase from low to high-level concepts where one depends on the other. For the context of music, for example, semantic knowledge in increasing complexity could include notes, timing, dynamics, scales, chords, harmony, and genre-specific repertoire. Unlike syntactic knowledge, semantic knowledge is independent of system-specific rules.

For Shneiderman, the power of DM comes from abstracting away the syntactic knowledge, empowering users to focus instead on semantic knowledge–which is often familiar to them. As Shneiderman explains: *"The object of interest is displayed so that actions are directly in the high-level problem domain. There is little need for decomposition into multiple commands with a complex syntactic form. On the contrary, each command produces a comprehensible action in the problem domain that is immediately visible, The closeness of the problem domain to the command action reduces operator problem-solving load and stress"* (ibid.).

1.1.2 Gulf of execution and gulf of evaluation

Hutchins and colleagues (1985) provided a cognitive account for the efficacy of DM according to two critical problems in interacting with computer systems: the *gulf of execution* and the *gulf of evaluation*.

On the one hand, the gulf of execution concerns the problem of how the user expresses intention using the available controls provided by the interface. In other words, it regards users expectations on how the interface controls may help them to concretize their intentions. Hutchins and colleagues argue that the gulf of execution can be reduced by *"making the commands and mechanics of the system match the thoughts and goals of the user"* (ibid.).

On the other hand, the gulf of evaluation concerns the problem of understanding the results of users actions through the feedback provided by the interface. Hutchins and colleagues argue that this gulf can be reduced by *"making the output displays present a good conceptual model of the system that is readily perceived, interpreted and evaluated"* (ibid.).

Hutchins and colleagues argue that the higher the gulfs (of execution and evaluation) one interface has, the higher is the cognitive effort required by that particular interface requires, making the interface less direct. Therefore, implementing an effective DM system could be achieved by reducing both gulfs.

1.1.3 Instrumental degrees for directness

Human beings use a wide diversity of tools to interact with the real physical world, augmenting the capabilities of the human body. Using this metaphor, Michel Beaudouin-Lafon generalizes DM in terms of an interaction model he called instrumental interaction (Beaudouin-Lafon 2000).

The instrumental interaction is based on two key components: the domain objects (i.e., objects of interest) and the tools[1] (i.e., "instruments"). Domain objects concern the computational representation we interact upon. Tools are actionable computational mediators, that enable us

[1]Beaudouin-Lafon uses the term "instrument" to refer to "tool"–hence the term instrumental interaction. However, for the context of this thesis, I opt to use the term "tool" instead. The purpose is minimizing confusion, as the term "instrument" is already used to denote "musical instrument".

to interact with domain objects, providing feedback while the action happens. For example, in traditional drawing applications, the canvas represents a domain object (i.e., computational representation of a sheet of paper), whereas the pencil and the paint bucket represent tools (i.e., one representing freehand-drawing, another representing painting a particular area).

Beaudouin-Lafon argues that instrumental power can be explained in terms of three properties:

Degree of indirection concerns the spatial and temporal distance between the tools' actions and their perceivable effects on the domain object. More specifically, spatial distance concerns the match between where the action occurs and where its outputs take place. For example, the pencil has a low spatial distance as the place where the drawing is displayed is almost the same where the pencil drawing occurs. Temporal distance concerns the match between when the action occurs and when its outputs take place. For example, the pencil has a low temporal distance as the drawing trace is continuously displayed as drawing occurs. Therefore, the lower the degree of indirection (i.e., the lower the spatiotemporal distance), the higher the directness.

Degree of integration concerns the match between the degrees of freedom provided by the input device with the degrees of freedom provided by the control parameters of the tool. For example, navigating multidimensional data using a two-dimensional mouse will likely result in a low degree of integration, and therefore, in lower directness.

Degree of compatibility concerns the behavioral match between physical actions required by the tools and their perceivable effects on the object domains. As Beaudouin-Lafon exemplifies: *"Using text input fields to specify numerical values in a dialog box, e.g., the point size of a font, has a very low degree of compatibility because the input data type is different from the output data type. Similarly, specifying the margin in a text document by entering a number in a text field has a lower degree of compatibility than dragging a tab in a ruler"* (Beaudouin-Lafon 2000). Therefore, the fewer mediators are necessary for the interaction, the higher the degree of compatibility and the higher the directness.

Beaudouin-Lafon explains that effective directness translates to a low degree of indirection, a high degree of integration, and a high degree of compatibility. Successful DM systems are capable of articulating these characteristics.

1.1.4 Where are we today?

While almost 40 years old, DM remains a relevant research topic in HCI. One recent research direction argues that traditional WIMP GUIs are intrinsically indirect, as these interfaces do not seem to fulfill the required accounts for directness. For example, dialog boxes have arguably a high degree of spatial and temporal indirection, as they shift user attention from the object of interest to the dialog window. Similarly, *"specifying the margin in a text document by entering a number in a text field has a lower degree of compatibility than dragging a tab in a ruler"* (Beaudouin-Lafon 2000). In this direction, some authors have proposed new powerful ways to explore DM—for beyond WIMP GUIs. Here, we cover some examples.

Dragicevic and colleagues (2008) explore DM in the context of video browsing. Instead of relying on time-oriented seeker bars (traditionally used in videos), the authors propose a new interaction technique focused on visual objects of the video, showing how their trajectory flow over time by mouse dragging–implemented in a new tool called the DimP. Evaluation has shown that DimP outperforms traditional seeker bars for navigating video based on specific visual elements, and was characterized as cleaner and easier-to-use by participants.

Recent Ph.D. theses have also investigated DM. Bridging research from information visualization and HCI (Perin 2014), Charles Perin explores DM principles to introduce novel interaction techniques for information visualization. Examples can be found on the 'À Table!', that enables users to directly manipulate sports ranking tables, focusing on how variables (e.g., number of points, victories, and defeats) evolve and compare to each other over time. Interaction occurs by mouse dragging visual elements of the ranking table, instead of relying on interface widgets. A mixed-methods user study with 143 participants suggests that 'À Table!' enabled participants to improve their analysis of soccer championships compared to traditional ranking tables. A similar

example is the Interactive Horizon Graphs, which involves parallel visualization and comparison of multiple time series. A quantitative user study with 9 participants shows that the Interactive Horizon Graphs improved task performance–in terms of time, correctness, and error–for scenarios involving manipulation of a large number of time series.

Another recent Ph.D. thesis applies DM to traditional desktop interaction (Aceituno 2015). Here, Jonathan Aceituno explores different accounts of directness within the HCI community, synthesizing them into a unified theoretical framework for directness. This framework drives the development of novel interaction techniques to extend the expressive capacity of standard input devices for desktop environments. One example is the subpixel interaction, a technique that enriches the mouse to enable continuous direct manipulation in-between discrete screen pixels, resulting in increased mouse accuracy in constrained display areas. Subpixel interaction introduces a wide range of new expressive applications such as editing calendar events (with minute precision) and video frame selection (with frame precision). Another example is the push-edge and the slide-edge, two novel auto-scrolling techniques, that outperform the default scrolling technique used in OS X computers in terms of reduced selection time and reduced overshooting.

Another context that recently explored DM concerns end-user programming. Hempel and Chugh (2016) introduce the Sketch-n-Sketch, an authoring environment for SVG drawing that brings DM to text-based structured programming. With Sketch-n-Sketch, snippets of textual programming code could be created via a WIMP GUI, allowing parameters customization via mouse dragging (e.g., one could drag-and-drop variables to change their values, and verify the impact of this change in the SVG in real-time). As a consequence, non-expert programmers can benefit from basic templates and customization, while experts can create more complex extensions to fit their needs.

Examples can also be found in the context of creativity-support tools and data visualization. Haijun Xia and colleagues explore DM by combining pen and touch inputs to leverage existing drawing skills for personal data visualization (Xia et al. 2018). The authors introduce the DataInk, a web system that enables users to directly manipulate their hand-drawn sketches, allowing dy-

namic data binding, layout automation, and visual customizations (e.g., color). Participants of a user study with eight novices and professional visual designers–who tried the DataInk in a 90 minutes session–reported that the system seemed easy to learn and yet afforded rich and diverse expressive visualizations.

1.2 Goal: Direct manipulation for musical interface design

The previous examples provide evidence for the potential of DM for beyond WIMP GUIs in the broad context of HCI. However, would similar approaches also make sense for the specific context of new interfaces for musical expression (NIME)?

There are reasons to believe so. First, while NIME craft has been traditionally diverse and idiosyncratic, it is common to find digital luthiers approaching structured methods as a way to increase efficiency and innovation in their practice. Also, crafting transparent musical interfaces that affords intimate control is a long-standing ideal within the NIME community (Fels 2004; Fels, Gadd, and Mulder 2002; Wessel and Wright 2002). Therefore, addressing DM in NIME could provide these luthiers with a theoretical framework and empirical evidence to inform their design decisions. Second, WIMP-based DM is widely popular in commercial musical tools (c.f., subsection 'Mainstream WIMP GUIs in Music Tools'), so that potential advances could occasionally find real-world applications. Finally, while HCI and NIME have a relatively old relationship (Holland, Mudd, et al. 2019; Poupyrev et al. 2001), the relationship is often limited to formal task-based evaluation (Wanderley and Orio 2002). There is today an increasing interest in overcoming this limitation and combining the areas in new, different ways (Holland, Garcia, et al. 2016; Holland, Mudd, et al. 2019; Holland, Wilkie-McKenna, et al. 2013) so that DM appears as an exciting opportunity for improving cross-pollination.

Motivated by these reasons, this thesis attempts to follow the same approach of these previous examples, contextualizing discussions on the NIME domain. The overall goal is towards a *theoretical framework of direct manipulation for musical interface design.*

1.3 Musical interfaces based on visuals

This section surveys *musical interfaces based on visual software*, targeting at the NIME literature. Given the large number of these interfaces, the focus is on either prominent historical examples or bodies of academic works describing more than a single artifact.

1.3.1 Iánnis Xenákis' UPIC and hand-sketching metaphor

An early example is Iánnis Xenákis' *Unité Polyagogique Informatique du CEMAMu* (UPIC) developed in mid-70s (Lohner 1986). With the UPIC, composers could directly generate and manipulate sound by sketching on a digitizer tablet with an electromagnetic stylus. Sketches were visually displayed to users in real-time along with the produced sound. As Lohners (ibid.) puts:

> *"The UPIC is a complex system designed to facilitate direct access to sound and musical material by the user (....) an easily accessible system, the basic functions of which a layman can learn in a few hours. (...) Xenakis's idea of UPIC puts forth a unique approach to the development of contemporary music practice on a more general level. Its concept supersedes the standard possibilities of music production by enabling swift and agile translation of thought into music. The operational simplicity of the graphic digitizer actually excludes the need for technical computer science expertise, in contrast to other systems in which such expertise is a prerequisite, requiring armies of assistants. Rather, the mind and creative impetus of the person using the UPIC become the center of attention. The UPIC reveals itself as an ideal incarnation of Hal Chamberlin's paradigm: 'the user sits with eyes glued to the screen, hands manipulating input devices, and mind closing the feedback loop'."*

UPIC's vision remains powerful, influencing a whole new generation of interfaces based on the same 'drawing sounds' metaphor. A few years after the UPIC, the Fairlight CMI synthesizer reached commercial success, being adopted by influential musicians such as Herbie Hancock and Peter Gabriel (Harkins 2015). Recently, Farbood and colleagues introduced the Hyperscore (2004),

that allowed novice musicians to sketch melody and harmony within a typical music timeline. Mark Zadel has developed the Different Strokes (2012), that maps audio samples to user-drawings, enabling non-linear playback control via a variety of input devices (e.g., mouse, multitouch interface). Another example is the Illusio (Barbosa, Calegario, Teichrieb, Ramalho, and Cabral 2013), a live looping tool that enables musicians to sketch their own loop controls in a multitouch interface. Anil Çamci has introduced the GrainTrain (2018), adapting Different Stroke's metaphor for tablet devices and adding multiple voices and further granular synthesis control. Finally, Spencer Salazar's Auraglyph is a sketch-based modular programming environment for tablets (2017).

1.3.2 Laurie Spiegel's Music Mouse

Another early example is the Music Mouse by Laurie Spiegel (1986). The Music Mouse is an early visual interface for music-making in desktop computers, centered on the mouse as an input device. The instrument incorporates music theory knowledge in order to enable users to direct control flow, rhythm, and dynamics of a piano MIDI-instrument. Because of the expert music-knowledge that was built-in to the system, people with no previous background on music can improvise harmonically complex music by moving the mouse. Additionally, music knowledge parameters such as scales, number of voices, tempo, and expressivity (i.e., staccato, legato) can be fine-tuned via keyboard shortcuts, enabling a wide diversity of musical expression.

1.3.3 Mainstream WIMP GUIs in music tools

Following the tradition of commercial human-computer systems, the successful interface type for music tools are WIMP GUIs (c.f., section 'Direct Manipulation'). I also include within this metaphor visual controls that emulate those physical input controls commonly used in hardware music equipment, such as knobs, buttons, and faders. Sometimes, even the 'look and feel' of these hardware pieces are visually emulated. This metaphor has been implemented by the most widely-used and successful commercial software music tools. Today, several examples of Digital Audio Workstations (DAWs), such as Ableton Live, Pro Tools, and Apple's Garage Band, Virtual Studio

Technology (VSTs), and even some audio programming environments (e.g., Reaktor)–to name a few–illustrate this paradigm.

1.3.4 Mobile-based applications

With the popularization of mobile computing and software-distribution platforms like the Apple Store and Google Play, a plethora of remarkable touch mobile-based musical applications are being designed today–contributing only in 2015 to an estimated 28.5 million dollars market of computer music applications (National Association of Music Merchants 2015). A relatively recent survey of these applications (focused only on iOS) can be found on Thor Kell Master thesis (2014), containing 337 tools analyzed according to their interface metaphor and mapping strategies employed. Notable examples such as the Loopy, the Borderlands, the Ocarina illustrate how these applications can be intuitive and powerful, captivating amateur and expert musicians alike. Chapter 3 attempts to go in this same direction.

1.3.5 Visual programming for computer music

Visual-oriented tools are also common in computer music programming. Examples such as Max MSP (Puckette 2002) and Puredata (Puckette 1997)–two dataflow-based visual programming environments–are among the most popular tools in the computer music community. In a recent study (2018), Pošcic and Krekovic surveyed 218 users of five of these tools–including Max MSP and Puredata–and showed that: 1) most developers are male and highly academic trained in music; 2) most of them have first heard of music programming while during university (28%) and from friends (26%); and 3) that most of them learn programming by online written (61%) and video (39%) tutorials.

1.3.6 Golan Levin's painterly interfaces

Another influential work is Golan Levin Painterly Interfaces (2000). In his master thesis, Levin introduced the Audiovisual Environment Suite (AVES): a set of easy-to-learn mouse-oriented mu-

sical interfaces based on the metaphor of Painterly Interface. This metaphor was built based on a critical review of early visual-music systems, composed of the following guidelines:

- *"The system makes possible the creation and performance of dynamic imagery and sound, simultaneously, in realtime;*
- *The system's results are inexhaustible and extremely variable, yet deeply plastic;*
- *The system's sonic and visual dimensions are commensurately malleable;*
- *The system eschews the incorporation, to the greatest extent possible, of the arbitrary conventions and idioms of established visual languages, and instead permits the performer to create or superimpose her own;*
- *The system's basic principles of operation are easy to deduce, while, at the same time, sophisticated expressions are possible and mastery is elusive".*

As a consequence of these guidelines, Levin intended to achieve the following:

> *"My focus is the design of systems which make possible the simultaneous performance of animated image and sound. (...) The design and implementation of a metaartwork–an artwork for creating artworks–whose interface is supple and easy to learn, but which can also yield interesting, inexhaustibly variable, and personally expressive performances in both the visual and aural domain"*

One example of such painterly interfaces was the Yellowtail. Visually, the yellowtail uses the metaphor of a black canvas where users can draw as if their mouse 'stroke' as a white pencil. Once drawn, these strokes are dynamically animated over the screen, brought to life. In terms of sound, these dynamic animations feed a sound spectrogram that was then sonified in real-time via Short-Time Fourier Transforms.

1.3.7 Sergi Jordà's sonigraphical instruments

For over 15 years, Sergi Jordà has crafted what he coined as "sonigraphical instruments"–user-friendly music making systems with high visual feedback component (Jordà 2003). The goal of such visual feedback was to reduce barriers for novice music making, enabling at the same time the development of skills and virtuosity as desired by experts—a concept known as "low entry fee and high ceilings", first introduced to the NIME literature by Wessel and Wright (2002). As put by Jordà:

> *"(...) a simple and complex tool all at once; a tool that would not dishearten hobbyist musicians, but would still be able to produce completely diverse music, allowing a rich and intricate control and offering various stages of training and different learning curves."*

One example of such a system is the F@ust Music On-Line (FMOL) (Jordà 2002). The FMOL is an online collaborative musical instrument that combined sound generators (e.g., oscillators, samplers, Karplus Strong's physical string model) and up to three user-chosen chained sound processors (e.g., Bandpass filters and delay) as the sound engine. Visual feedback consists of horizontal and vertical oscilloscope-based strings. On the one hand, vertical lines could be directly manipulated by users via mouse, dynamically reacting according to sound engine parameters, such as loudness. On the other hand, horizontal strings represented sound processors and, while they did not vibrate, they could be dragged by users. Such integration of visual feedback with the sound engine, along with a simple standard input device, the mouse, was the strategy devised by Jordà to achieve the desired "low entry fee"–similar to what Levin and Spiegel had done before.

Jordà repeats the strategy in a second representative example: the Reactable (Jordà, Geiger, et al. 2007)–co-developed with Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The Reactable builds upon FMOL, replacing the desktop screen by a tabletop multitouch table and the mouse by tangible controls (Fitzmaurice, Ishii, and Buxton 1995). The sound engine is a constrained subset of objects from Max/MSP dataflow audio programming environments, controlled

via dynamic patching (Jordà 2003). Visual feedback is again at the heart of the interaction, characterized by Jordà as functional—that is, avoiding ornamental graphics (i.e., *"any shape, form, line, or animation drawn by the visual synthesizer is strictly relevant and informational"*)—and pro-transparency—that is, allowing users to *"constantly monitoring objects states and internal parameters"*.

1.3.8 Jean-Michel Couturier's graphical interaction for gestural control

Another example is the work of Jean-Michel Couturier with graphical interaction for post-WIMP gestural sound control. As detailed in his Ph.D. thesis (2004), Couturier explores Beaudouin-Lafon's instrumental interaction style (c.f., subsection "Instrumental degrees for directness") to guide the design of a set of post-WIMP musical interfaces. One example is the Scanned-Synthesis instrument (Couturier 2002), based on a synthesis technique with the same name (Verplank, Mathews, and Shaw 2001), where the content of the wavetable is dynamically customized via a physical spring-based model. Parameters of this model can be controlled via different input devices such as touch-screens, Wacom tablets, and others. Another example is the Filtering Strings (Arfib, Couturier, and Kessous 2005), where on a visual representation of a virtual string is attached to a 32-filters bank that can be modified via input controls (again, Wacom tablets and multitouch control).

Along with his colleagues, Couturier argues that visual feedback can be an effective strategy for achieving expressivity in digital instruments (ibid.) if used to convey internal mechanics of how an instrument works for both audience and performers. Similar considerations on the potential of visuals for enhancing DMIs expressivity have been made by other NIME practitioners (Dobrian and Koppelman 2006; Schloss 2003). Couturier also argues that succeeding requires accounting for the flux of visual information provided: too much information could actually make comprehension more difficult.

1.3.9 Thor Magnusson's screen-based instruments

Others are interested in using visual feedback for idiosyncratic, political, and aesthetic-motivated explorations. One example is the case of Thor Magnusson's screen-based instruments (2017). Magnusson borrows the concepts of affordances and constraints from HCI to argue that WIMP GUI interfaces often limit the creative musical practice, by shaping musical outcomes in advance (Magnusson 2006, 2010). Therefore, Magnusson points to the need for developing new idiosyncratic sets of constraints and affordances that could yield novel disruptive musical interfaces interface.

Following this philosophy, Magnusson develops the ixi software, a family of more than ten visual software-based musical interfaces for desktops. One representative example is the SpinDrum (Magnusson 2006), where sample sequencers with different tempo are reified as abstract rotating wheels. Another example following up the same spirit is the ixiQuarks family (Magnusson 2017), which includes instruments such as SoundScratcher—a sampler whose playback can be directly manipulated by the users through different configurable modes—and the artificial-life based Predators—where simulations of predators and preys interacting with one another controls sound generation. Typically, these interfaces communicate via OSC with sound engines written either in SuperCollider or Puredata. For the interaction, they use mouse and keyboard.

1.3.10 Ge Wang's principles of visual design

Another example is Ge Wang, whose work explores the potential of artsy visual design for real-time music making (Wang 2016). Following the example of Perry Cook with his music controllers principles (Cook 2001, 2009), Ge advocates that "visual design for computer music is as much art as it is science" and has synthesized years of experience designing such systems into 13 idiosyncratic design guidelines (Wang 2016). These guidelines are:

"User-Oriented Principles

- *Make it realtime whenever possible. Design sound and graphics in tandem: Neither should be an afterthought; seek salient mappings.*

- *Invite the eye–of experts and newcomers alike.*
- *Hide the technology: Induce viewers to experience substance, not technology.*
- *Do not be afraid to introduce arbitrary constraints.*
- *Reinforce physical interaction using graphics (especially on touch screens).*

Aesthetic Principles

- *Simplify: Identify core elements; trim the rest.*
- *Animate, create smoothness, imply motion: It is not just a matter of how things look but how they move.*
- *Be whimsical and organic: glow, flow, pulsate, breathe; imbue visual elements with personality.*
- *Have an aesthetic; never be satisfied with 'functional'.*

Other Principles

- *Iterate (there is no substitute for relentlessness).*
- *Visualize an algorithm to understand it more deeply and discover new directions;*
- *Glean inspiration from video games and movies."*

One representative example that incorporates many of these guidelines is the Ocarina (Wang 2014), developed in 2008 for the iPhone platform. Named after the ancient wind instrument, the Ocarina is an early example of a mobile-based musical interface that reached relative popularity (ten million users worldwide). The Ocarina integrated multiple smartphone sensors—such as the microphone (where the user had to blow in order to start producing sound), touch control (for choosing notes), and accelerometers (for articulation)—into a simple and cohesive flute-like interaction, largely based on visuals. While functionally "unnecessary" (Wang's word), Wang

argues that these visuals help to compensate the lack of haptic feedback on touchscreens, and was *"essential in order to convey a sense of magic and hide the technology"*.

Despite Ocarina's relative simplicity, Ge's principles are also applied to more complex domains such as audio programming. One example is the Audicle (Wang and Cook 2004), a visual live-environment for editing, compiling, debugging and visualizing ChucK programming language. Integrated with ChucK's engine, Audible allowed 3D real-time visualization of abstract programming elements such as program's structure, global and local variables, concurrency, and time. The aim was making these elements less abstract and more directly 'graspable' to users, facilitating, therefore, the programming process. The same direction is explored in Chapter 5 and Chapter 6.

1.3.11 Virtual reality musical instruments

Another body of work where visual feedback is recurrent is virtual reality musical instruments (VRMIs) (Serafin et al. 2016). VRMIs are a subcategory of graphical visual instruments that focuses on virtual reality technology, where the performer is immersed on 3D visualizations that are at the core of the interaction. VRMIs have a particular history on their own, recently surveyed by Serafin and colleagues (ibid.). Like Ge Wang, Serafin and colleagues also lay out principles to guide the design of VRMIs. Among these, six are visual-related and are quoted below:

- *"Design for Feedback and Mapping;*
- *Reduce Latency;*
- *Prevent Cybersickness;*
- *Consider Both Natural and 'Magical' Interaction;*
- *Create a Sense of Presence;*
- *Represent the Player's Body."*

One example of VRMI where we can find some of these guidelines is the Drile (Berthaut, Desainte-Catherine, and Hachet 2010). The Drile is a live looping tool where the performer is vi-

sually immersed in a 3D world where loops are reified into complex 3D shapes. These sound-shapes are organized into a hierarchical tree data structure—defining an advanced looping technique called hierarchical live looping—that can be directly manipulated by the performer via the Piiver–a bi-manual Wiimote-like input device. Here, visual feedback plays a crucial role: without visuals the instrument becomes unplayable.

However, as in Ge Wang's Ocarina, we can also find examples where visual feedback is functionally 'unnecessary'. That is the case of the augmented-reality based Rouagues (Berthaut, Subramanian, et al. 2013). The Rouages explores visual feedback to highlight internal mechanics of how the instrument works, aiming at increasing audience understanding on the connection between performer's gestures and sound output–following the direction pointed by Couturier and colleagues (Arfib, Couturier, and Kessous 2005). For the performer, however, visual feedback is secondary.

In addition to these works, many other VRMIs can be found in the literature (Serafin et al. 2016). In general, VRMIs are a peculiar type of musical interfaces where visual feedback tends towards realistic 3D representations, aiming at user immersion. Therefore, many guidelines introduced by Serafin and colleagues resonates with past works, and the desire of making music interaction more direct, engaging, and straightforward seems to persist.

1.3.12 Frameworks for musical interface design

While not primarily focused on visual software, many theoretical frameworks have been proposed to support musical interface design. Examples include Pennycook's pre-NIME principles (1985); George Essl and Sile O'Modhrain's enactive framework for NIME tangibility (2006); Fabio Morreale's framework for User Experience (2014); and Ian Hattwick's framework for hardware design in professional artistic productions (2017). Here, the focus is on some representative examples that can be related to DM.

Joel Ryan's remarks at STEIM

Joel Ryan presents considerations on the pro-idiosyncrasy design philosophy used at the STEIM (STudio for Electro-Instrumental Music) (J. Ryan 1991). For many years, the STEIM has supported artists to design new musical instruments. Examples include George Lewis's 'Voyager', and Michel Waisvisz's 'the Hands'.

One particular consideration concerns the role of *effort and expression*. Ryan affirms that physical effort is *"closely related to expression in the playing of traditional instruments"*. Therefore, NIME designs would need to stimulate physical effort, despite the tendency computers have to make things 'effortlessly'. One strategy for enabling effort is increasing the *responsiveness* of the instrument, by providing physical feedback to performers. Such responsiveness, Ryan argues, would enhance feedback loop between musician and musical instrument, enabling the development of skills and musical thoughts over time.

Ryan also affirms that *quick interactive feedback* play a crucial in software development tools used at STEIM. He explains that this feedback allows for a quick iterative search for new intriguing sounds, where discovery happens by trial and error. Furthermore, Ryan claims that quick interactive feedback seems to facilitate learning for technically-struggling artists: *"Those who are writing higher level music language must continue to put more effort into providing more interactive environments"*.

Wessel and Wright's intimate control

In (Wessel and Wright 2002), David Wessel and Matthew Wright summarize years of experience crafting NIME for personal use, pointing out their personal requirements for NIME design. These requirements concern:

1. Initial ease of use for novices (i.e., low-entry fee), coupled with "long-term potential for virtuosity" (i.e., high ceilings);

2. Low and minimally-variable latency for input-to-sound response;

3. Adopting continuous input controls (as if these controls were signals) instead of relying only on discrete events (as provided in MIDI controls);

4. 'Clear and simple' strategies for input-to-sound mapping; and

5. Providing at least one minimal set of essential musical control, namely: a) muting and unmuting the sound stream; b) volume and sound density control; c) mapping the 'size' of input gesture directly to sound output; and d) predictability on control.

These requirements have become influential within the NIME community–especially the idea of low-entry fee and high ceilings (Jordà 2017; Wright 2017). Along with requirements such as simple mapping strategies, low and minimally-variable latency, adopting continuous input controls, the authors argue that these requirements would yield in an 'intimate' musical control.

Perry Cook's principles

The series of articles published by Perry Cook is another representative example. For Cook, the area *"proceeds as more art than science, and possibly this is the only way that it can be done"* (Cook 2001). As such, the author condenses his lifelong expertise on 23 informal principles for creating new musical digital controllers. These principles were initially introduced in (ibid.), later revised on (Cook 2009), and recently commented on (Cook 2017).

Among the 23 guidelines, one arguably relate to DM: the 6th *"Instant music, subtlety later"*. Cook advocates that musical controllers should enable immediate music making, minimizing the time required for setups and configurations–as typically found in prototypes. Moreover, two guidelines—the 22nd and 23rd—explicitly mention visual feedback. In these cases, however, feedback is limited to debugging and repairing controllers (i.e., *"Build in diagnostic features and displays"*), or on simulating input data (i.e., *"Construct controller proxies"*).

Sidney Fels' transparency and intimacy

Sidney Fels introduced the idea of *transparent interfaces* (Fels, Gadd, and Mulder 2002). Transparency concerns a property of an interface related to the *"psychophysiological distance, in the minds of the player and the audience, between the input and output of a device mapping"*. The higher the transparency, Fels argues, the higher the instrument expressivity. This causality would have a reason: transparent interfaces would yield intimate control (Fels 2004). Intimacy describes a property of the performer-instrument relationship where there is a *"perceived match between the behavior of a device and the operation of that device"*, resulting in a sense of embodiment (i.e., the instrument becomes an extension of the performer's body).

Both concepts are explored in the Iamascope: a contactless visual-based musical interface developed by Fels and Mase (1999). Based on a kaleidoscope metaphor, the Iamascope used the raw input of a video camera filming the user to feed both kaleidoscope-like projected visuals and a MIDI synthesizer, so that every tiny movement by the user yields in visual animation and sound. When there is no movement, no sound is generated. Fels credits the responsive whimsy imagery and its coupled sound mapping as critical components for the success of the Iamascope (the authors explains that the Iamscope was enjoyed by more than one thousand people' over different exhibitions).

Fels' discussions on transparency and intimacy seem to resonate with the concepts of gulf of execution and gulf of evaluation (c.f., subsection 'Gulf of execution and gulf of evaluation'). More specifically, the gulf of execution relates to transparency (i.e., one could argue that the lower gulf of execution, the higher the transparency), whereas the gulf of evaluation concerns intimacy (i.e., one could argue that the lower gulf of evaluation, the higher the intimacy). Therefore, using Fels' terminology, effective direct manipulation systems would require transparency and intimacy.

1.4 Evaluating musical interface design

The previous section has presented several musical interfaces based on visual software. However, little has been said so far on how to evaluate these interfaces. The topic is discussed in this subsection.

One natural approach–that borrows from traditional musical instruments–concerns: 1) engaging in continued musical practice with the instrument, by composing pieces and developing repertoires; and 2) its acceptance in real-world, via adoption by other musicians. Many of the instruments presented in the previous section seem to fulfill these requisites, such as the case of Ableton Live, the Fairlight CMI, and Max MSP. However, the example of traditional instrument shows that adoption and repertoire development may take many years–or even centuries–, making this approach unpractical for most cases. As a consequence, many researchers started to engage in structured formal approaches to evaluate NIMEs– an especially challenging direction that has intrigued researchers ever since the origins of the NIME conference (Holland, Garcia, et al. 2016; Poupyrev et al. 2001).

A comprehensive meta-review of these attempts is presented by O'Modhrain (2011), who articulates a theoretical framework organized according to four stakeholders involved in the musical context: The performer, the designer, the audience, and the manufacturer. For each stakeholder, O'Modhrain propose potential methods to assess four criteria: enjoyment; playability; robustness; and achievement of design specifications. Evaluation methods presented are mostly quantitative and qualitative. Each one of these widely-used empirical traditions suits a specific range of problems, as contrasted on Table 1.1.

One representative *quantitative* framework concerns Wanderley and Orio (2002), who propose evaluating the usability of music gestural controllers based on some established HCI-methods– namely quantitative usability testing, taxonomies, and GOMs models. The framework relies on context-dependent musical tasks specially designed to measure features such as learnability, explorability, feature controllability, and timing controllability. The authors argue that outcoming

Table 1.1 Comparing quantitative and qualitative methodologies in terms of strengths, and weaknesses. This table is adapted from a previous work in the context of educational research (Johnson and Onwuegbuzie 2004)

Quantitative	**Qualitative**
Values	
Internal validity, External validity, Reliability, Objectivity (Pickard 2013);	Credibility, Transferability, Dependability, Confirmability (Pickard 2013);
Strengths	
Validating already constructed models, frameworks, or hypothesis;	Can be applied to generate theories in exploratory scenarios, where there are no clear research questions, via the Grounded theory;
Allows generalization for different contexts;	Results reflect participant's vision of the problem;
Allows data-driven predictions;	Suited for complex phenomena, typical recurrent in music, where variables are hard to isolate;
Allows elimination of confounding factors, providing a clearer assessment of cause-and-effect relationships;	Provides in-depth context-specific knowledge about individual cases;
Data collection and analysis is often less time consuming;	Embraces contextual differences in cross-case comparison;
Tends to be less biased by the investigator's perspective, allowing replication;	Flexible to potential changes during the study;
May have more credibility among institutions (e.g. funding agencies); and	Suitable for small populations, such as the DMI, where expert users are rare; and
Applicable to a larger number of participants	No need for laboratory experiments that can intimidate participants, using natural settings instead.
Weaknesses	
Models, frameworks, and hypothesis may not reflect participants perspective of the problem;	Findings may be particular for the study's context (i.e not generalizable);
The possibility of confirmation bias, where results are biased by expectations of investigators;	Unsuited for data-driven predictions;
Not flexible to changes;	Unsuitable to test/verify hypothesis, or models;
Results can be so abstract and general that become hardly useful in practice;	May have lower credibility among institutions (e.g. funding agencies);
Needs a large number of participants, which may be unsuited for the DMI context; and	Data collection and analysis are often time-consuming; and
Defining representative tasks and operationalizing subjective criteria (e.g. expressivity) can be difficult.	Results are inherently biased by researcher's idiosyncrasies

results could be used to inform DMI design decisions. Chapter 5 presents one possible application of this framework, where I compare ZenStates's understandability against two popular alternatives through a lab controlled user study.

One representative *qualitative* framework is suggested by Stowell and colleagues (2009). The authors argue for qualitative methodologies to encompass a more holistic view of music making, avoiding reducing music interaction to separable tasks. A case study illustrates the authors' point, where they use the Discourse analysis method for comparing two different versions of a voice-based NIME–one version with a specific timbre remapping technology developed by Stowell; another version without the technology–with five beatboxers in short-term exploratory sections. The method, the authors argue, enables *"a detailed reconstruction of users' conceptualization of a system"*. An alternative qualitative approach is presented in Chapter 6, where I use a longitudinal study to investigate StateSynth's learnability thresholds and expressive ceilings with two professional musicians.

Alternatively, there are recent discussions on NIME as a practice-based design-oriented research area, where these traditional forms of evaluation are challenged (Dahl 2016). This approach seems aligned with the tradition of research through design (Gaver 2012), that values insightful documentation and annotated portfolio as a way to reflect and learn from one's practice. As such, Gaver advocates that *"design research community should be wary of impulses towards convergence and standardization, and instead take pride in its aptitude for exploring and speculating, particularizing and diversifying, and - especially - its ability to manifest the results in the form of new, conceptually rich artifacts"*. Chapter 3 and Chapter 4 attempts to engage in this debate, by exploring how rich and detailed design processes could potentially work as formative evaluation.

Despite these advances, NIME evaluation remains a complex and challenging topic, discussed with more details on Chapter 2.

1.5 Summary

Section 1.2 surveyed NIME systems and frameworks related to visual software. Table 1.2 presents a descriptive summary of these systems. In addition to basic information such as the **input devices** necessary for interaction, and the **visual display** used for the visual feedback, this table includes:

Sound module complexity: Referring to the sound synthesis used, categorized according to their complexity as: 1) *limited*, where only a few musical parameters of one specific sound synthesis are available; 2) *moderated*, in-between the extremes, with many parameters of one specific type of synthesis, for example; or 3) *complex*, that provides many musical parameters of many types of sound synthesis.

Musical context: Describing the musical context where interaction takes place and the role the instrument in this context. As discussed in previous works (Malloch et al. 2006; Rasmussen 1983), the categories range from: 1) skill-signal based, denoting *"a real-time, continuous response to a continuous signal"* (e.g., acoustic playing; turntablism); 2) rule-sign-based, denoting *"selection and execution of stored procedures in response to cues extracted from the system"* (e.g., interactive music systems; directive sequencing); until 3) model-symbol based, meaning *"a level yet more abstract, in which performance is directed towards a conceptual goal, and active reasoning must be used before an appropriate action (rule- or skill-based) is taken"* (e.g., live coding; algorithmic music).

Visual category: Referring to the type of visual representation used within the interface. According to Blackwell (2013), these visual representations can be: 1) *Typography and text*; 2) *Maps and graphs*; 3) *Schematic drawings*; 4) *Pictures*; 5) *Node-and-link diagrams*; 6) *Icons and symbols*; and 7) *Visual metaphor*.

Table 1.2 Descriptive summary of the surveyed visual software-based musical interfaces

Name	Input device	Visual display	Sound module complexity	Musical context	Visual category
UPIC	Digitizer tablet; Computer keyboard	Monitor	Moderate: Waveforms & envelope control; Sample-based synthesis	Model-symbol	Visual metaphor (hand-sketching)
Fairlight CMI	Light pen; Computer keyboard; Musical keyboard	Monitor	Moderate: Sample-based synthesis	Rule-sign; Skill-signal	Text; Graphs;
Music Mouse	Mouse; Computer keyboard	Monitor	Limited: MIDI	Rule-sign; Skill-signal	Visual metaphor (two-dimensional matrix); Text
Ableton Live (WIMP GUIs)	Mouse; Computer keyboard	Monitor	Complex: Digital Audio Workstation	Model-symbol	Icons and symbols; Visual metaphor (GUIs)
Loopy (Mobile-based apps)	Tablet	Tablet	Limited: Live looping	Rule-sign	Visual metaphor
YellowTail	Mouse; Computer keyboard	Monitor	Limited: Inverse Fast Fourier Transform	Skill-signal	Visual metaphor
Reactable	Multitouch surface; Tangibles	Multitouch surface; Tangibles	Complex: Audio units	Rule-sign; Skill-signal	Node-and-link diagrams
FMOL	Mouse; Computer keyboard	Monitor	Complex: Audio units	Rule-sign; Skill-signal	Schematic drawings
Iamascope	Video camera	Projection	Limited: MIDI	Rule-sign	Visual metaphor (kaleidoscope)
Scanned Synthesis Instrument	Digitizer tablet; Touch input; Computer keyboard	Monitor	Moderate: Scanned synthesis	Skill-signal	Graph; Icons and symbols; Visual metaphor (GUIs)
Filtering Strings	Digitizer tablet; Touch input	Monitor	Moderate: Scanned synthesis	Skill-signal	Graph; Icons and symbols; Visual metaphor (GUIs)
Hyperscore	Mouse; Computer keyboard	Monitor	Simple: MIDI; Sequencer	Model-symbol	Visual metaphor (timeline; hand-sketching)
SpinDrum	Mouse; Computer keyboard	Monitor	Moderate: Sample-based synthesis;	Rule-sign	Visual metaphor (Custom; GUIs)
Soundscratcher	Mouse; Computer keyboard	Monitor	Moderate: Sample-based synthesis; Granular synthesis	Rule-sign	Graph; Visual metaphor (Custom; GUIs)
Ocarina	Tablet	Tablet	Moderate: Physical modeling	Skill-signal	Visual metaphor (game inspired)
Audicle	Mouse; Computer keyboard	Monitor	Complex: Audio units	Model-symbol	Text; Graphs; Visual metaphor (Multiple custom metaphors)
Different Strokes	Mouse; Computer keyboard	Monitor	Limited: Sequencer	Model-symbol; Rule-sign	Visual metaphor (hand-sketching)
Illusio	Touch surface; Footswitch	Touch surface	Limited: Live looping	Rule-sign;	Visual metaphor (hand-sketching)
Drile	Piivert	Projection	Limited: Live looping	Rule-sign;	Visual metaphor (3D models)
Rouages	Hardware music controller	Projection	Limited: MIDI	Rule-sign; Skill-signal	Visual metaphor (3D models; GUIs)
Auraglyph	Tablet; Pen	Tablet	Complex: Audio units	Model-symbol; Skill-signal	Node-and-link diagrams; Visual metaphor (hand-sketching)
GrainTrain	Tablet	Tablet	Moderate: Granular synthesis	Model-symbol; Skill-signal	Visual metaphor (hand-sketching; GUIs)
Max MSP (Visual programming)	Mouse; Computer keyboard	Monitor	Complex: Multimedia programming	Model-symbol	Node-and-link diagrams; Visual metaphor (GUIs)

Section 1.1 presented the theoretical foundations of DM in human-computer systems. Table 1.3 uses these foundations as metrics to analyze the surveyed visual interfaces as follows:

Continuous visual representation of sound: Describing which elements of the sound module are continuously displayed in the interface. In addition to a short summary (here, multiple items are presented according to their visual relevance in the UI), representations are classified as: 1) *Incipient:* if the sound module is underrepresented in visual terms (e.g., core elements do not have visual representations); 2) *Partial:* if only core elements of the sound module are visually represented; or 3) *Advanced:*, if most elements of the sound module are visually represented.

Sound manipulations become immediately visible: Describing how visuals updates occur after sound interactions. They can be: 1) *None* (i.e., no immediate updates occur); 2) *Require interface navigation*, where visual updates are immediate but not displayed all at the same time, requiring users to perform interface navigation to reach the desired sound-visual representation; 3) *All updates at once*, where all sound elements are immediately displayed on the screen at the same time; or yet 4) *Ambiguous*, where the update process is unclear, because, for example, significant sound elements do not have visual correspondents, or because updates are not immediate;

Rapid, incremental, reversible operations for sound: Describing how many rapid, incremental and reversible operations are available to users for sound manipulation, categorized as either *incipient*, *some*, or *many*;

Sound production via physical actions, instead of complex syntax: Describing whether sound production occurs via physical actions (c.f., Joel Ryan discussions on effort and expressiveness), instead of more indirect approaches. They can be: 1) *Incipient*, where little physical actions are necessary for sound production; 2) *Partial*, where physical actions are required for modifying an existing sound stream (which plays independently of physical actions); and 3) *High*, where most sound production happens via physical actions.

Table 1.3 Analysis of the surveyed musical interfaces according to the principles of DM.

Name	Continuous visual representation of sound	Immediately visible sound manipulations	Rapid, incremental, reversible operations for sound	Sound production via physical actions, instead of complex syntax
UPIC	Incipient: Abstract hand-drawn shape (to be associated to musical representation)	None	Incipient	Incipient
Fairlight CMI	Advanced: Several sample-based synthesis parameters	Requires interface navigation	Many	High
Music Mouse	Advanced: Notes currently played, articulation-related parameters, and meta info (e.g., tempo)	All sound controls at once	Many	High
Ableton Live (WIMP GUIs)	Advanced: Numerous and diverse musical parameters, from low level to high level	Requires interface navigation	Many	Incipient at first, but increases after user configuration
Loopy (Mobile-based apps)	Partial: Multiple loops playback, and basic mixing control (e.g., volume)	All sound controls at once	Some	Partially
YellowTail	Partial: Sounds currently being played	All sound controls at once	Some	Partially
Reactable	Advanced: Audio units, their input control parameters, and connections between these	All sound controls at once	Many	Partially
FMOL	Partial: audio units, their input control parameters	All sound controls at once	Many	Partially
Iamascope	Partial: Frequency of the played notes	Ambiguous	Incipient	High
Scanned Synthesis Instrument	Basic: Changes in timbre, playing mode (chord or note), and other scanned synthesis parameters	All sound controls at once	Some	High
Filtering Strings	Partial: Changes in timbre and frequency	Ambiguous	Some	High
Hyperscore	Advanced: Note frequencies, instruments, amplitude, chord changes, key modulations, and meta information (e.g. tempo)	All sound controls at once	Many	Incipient
SpinDrum	Advanced: Sample-based synthesis parameters	All sound controls at once	Many	Partially
Soundscratcher	Advanced: Sample-based and granular synthesis parameters	All sound controls at once	Many	Partially
Ocarina	Partial: Audio input, note played	All sound controls at once	Some	High
Audicle	Advanced: Several low level sound parameters from the Chuck environment	Ambiguous	Many	Incipient
Different Strokes	Partial: Playback position of multiple samples	All sound controls at once	Incipient	Partially
Illusio	Partial: Multiple loops playback, and basic mixing control (E.g. volume)	Requires interface navigation	Some	Partially
Drile	Partial: Multiple loops playback, basic mixing control, and effects	Ambiguous	Some	Partially
Rouages	Advanced: Playback and volume control for loops, synth voices, and scores	All sound controls at once	Some	High
Auraglyph	Advanced: audio units, their input control parameters, and connections between these	All sound controls at once	Many	Partially
GrainTrain	Partial: Grain playback position and five sound parameters	All sound controls at once	Many	High
Max MSP (Visual programming)	Incipient, but may increase to advanced after user customization	Requires interface navigation	Many	Incipient at first, but increases after user configuration

Together, Table 1.2 and Table 1.3 suggest that DM can be a powerful framework for better *understanding* existing visual interfaces for music. But can DM also help us in *designing* novel visual interfaces for specific musical contexts? Are there concrete interface strategies to foster DM? Other interesting open questions include:

1. Most instruments with limited sound modules are used in rule-sign musical contexts providing only a limited number of rapid, incremental, reversible operations for sound (e.g., Loopy; Different Strokes; Illusio). How increasing the number of these operations would impact overall directness?

2. Instruments with complex sound modules (e.g., Ableton Live; Max MSP) often struggle to: 1) display all sound manipulations immediately visible (probably due to the amount of information to be displayed); 2) fully exploring physical actions for sound production without relying on complex syntax–as proposed by Joel Ryan in his discussions on effort and expression. How to overcome these limitations?

3. How to evaluate these instruments? Are DM benefits–traditionally found on human-computer systems–also transferable to musical contexts?

These are some of the questions I attempt to cover in this thesis. It is formatted as a manuscript-based thesis and gathers five different papers written during my Ph.D. studies.

1.6 Thesis structure

This thesis can be divided into three parts. The *first part* presents the theoretical context to be explored, its scope and motivation, comprising Chapter 1 and Chapter 2. Chapter 1 introduces a comprehensive survey of previous works related to DM in the contexts of NIME and HCI, in addition to representative examples of musical interfaces based on visual software. Chapter 2 is dedicated to surveying evaluation studies within musical interface design.

The *second part* brings the theoretical discussion to practice, presenting exploratory case studies where direct manipulation is designed and evaluated in three different contexts of music tools:

- The first case study concerns *live looping tools* (Chapter 3). It introduces a novel direct-manipulation based live looper tool called the **Voice Reaping Machine**. Its design and formative evaluation uses a design rationale approach, comprising four steps: 1) Surveying and analyzing 101 existing live looping tools; 2) Building a design space from this analysis; 3) Exploring potential guidelines using our design space as baseline; and 4) Iteratively prototyping several low-fi and functional prototypes exploring our guidelines.

- The second case study concerns *tools for creating interactive artistic installations* (Chapter 4 and Chapter 5). It introduces the **ZenStates**: a simple yet powerful visual model for interactive installations based on Hierarchical Finite-States Machines–discussed in Chapter 5. ZenStates design (Chapter 4) was iterative and user-centered, based on field studies, interviews, and iterative prototyping. Evaluation of the tool is four-folded: 1) implementing the model in a direct-manipulation based interface; 2) probing what ZenStates can create through 90 exploratory scenarios; and 3) performing a user study to investigate the understandability of ZenStates' model; and 4) a preliminary real-world application with professional artists.

- The third and final case study concerns *music authoring tools* (Chapter 6). It introduces a direct manipulation based music authoring tool called **StateSynth**, where music keyboard expertise becomes a core element for the programming. StateSynth empowers technically-struggling musicians to build personal interactive music tools, by articulating rich interactive behaviors in terms of their embodied instrumental skills. Designed after ZenStates, I evaluate StateSynth's learning thresholds and expressive ceilings in a one week study with expert musicians.

Finally, the *third part* (Chapter 7) returns from practice to the theoretical discussion, by presenting lessons learned from these exploratory case studies. These lessons are synthesized into

a set of interface design strategies and evaluation approaches for effective direct manipulation in musical interfaces based on visual software.

Chapter 2

What does "Evaluation" mean for the NIME community?

The last chapter discussed previous NIME and HCI work related to DM. This chapter focuses on the role of evaluation for musical interface design. It has been previously published as:

> `Jeronimo Barbosa`, Joseph Malloch, Stéphane Huot, Marcelo M. Wanderley. *"What does 'Evaluation' mean for the NIME community?"* In Proceedings of the 2015 International Conference on New Interfaces for Musical Expression, Baton Rouge, USA.

2.1 Abstract

Evaluation has been suggested to be one of the main trends in current NIME research. However, the meaning of the term for the community may not be as clear as it seems. In order to explore this issue, we have analyzed all papers and posters published in the proceedings of the NIME conference from 2012 to 2014. For each publication that explicitly mentioned the term "evaluation", we looked for: a) What targets and stakeholders were considered? b) What goals were set? c) What criteria were used? d) What methods were used? e) How long did the evaluation last? Results show different understandings of evaluation, with little consistency regarding the usage of the word.

Surprisingly in some cases, not even basic information such as goal, criteria and methods were provided. In this paper, we attempt to provide an idea of what "evaluation" means for the NIME community, pushing the discussion towards how could we make a better use of evaluation on NIME design and what criteria should be used regarding each goal.

2.2 Introduction

> "*In essence, while the search for solid and grounded design and evaluation frameworks is one of the main trends in current NIME research, general and formal methods that go beyond specific use cases have probably not yet emerged. Will these be the El Dorado or the Holy Grail of NIME research?*" (Jordà and Mealla 2014)

The paragraph above, quoted from Jordà and Mealla's paper published at NIME 2014, illustrates the high expectations often associated with evaluation in NIME research today. This growing interest can also be statistically observed in the conference proceedings. Based on previous works (Barbosa, Calegario, Teichrieb, Ramalho, and McGlynn 2012; Stowell et al. 2009), we have performed text analysis on the proceedings of the three last NIME conferences (from 2012 to 2014) and tracked how many publications reported to have performed an "evaluation". Considering oral and posters presentations only: In 2012, 34% of the publications that proposed a NIME evaluated the proposed devices; In 2014, the number has increased to 49% of the publications, as shown in Table 2.1.

Table 2.1 Number of "*evaluations*" reported in NIME publications from 2012 to 2014, based on (Barbosa, Calegario, Teichrieb, Ramalho, and McGlynn 2012; Stowell et al. 2009).

Evaluates?	**2012**	**2013**	**2014**
Not applicable	24	41	56
No	39	35	41
Yes	20	29	40
Total	34%	45%	49%

However, as the number of evaluations increases, it appears that the meaning of "*evaluation*"

in the context of NIME or digital musical instruments (DMIs) may not be as evident as it seems. Initial analyses of the content of evaluation-related papers in NIME literature show us that there are different understandings of the meaning of the term "evaluation". It is common to find papers that use the term to denote the process of collecting feedback from users in order to improve a prototype (e.g., publication 14#A#48 in our corpus[1]). It is also common to find others that use the term to assess the suitability of existing devices for certain tasks (Wanderley and Orio 2002), or to compare different devices using common characteristics (Birnbaum et al. 2005). Describing emerging interaction patterns when using the devices may also be found (e.g., publication 13#O#66). And all in all, these different objectives are all hidden behind the same general term of "evaluation".

Furthermore, there are other complicating factors. As pointed out by (Kvifte and Jensenius 2006; O'Modhrain 2011), there are several stakeholders that might be involved in the design of DMIs and the requirements of one may not intersect those of another. Thus, criteria considered as important for one stakeholder (e.g., playability for the performer (Jordà 2005)) might not be as important for another one (e.g., the audience). In addition, depending on the stakeholder and the goal specified for the evaluation, the time window chosen (Hunt and Kirk 2000) and the stakeholder's expertise with DMIs (Ghamsari, Pras, and Wanderley 2013) might also impact the results. In the case of acoustic instruments, for instance, the criteria for evaluating the suitability of a guitar for a beginner might not be the same as for a trained musician.

In this exploratory research, we aim to give insights into how the term evaluation has been more commonly employed in the NIME literature. For this, we have analyzed the proceedings of NIME conference from 2012 to 2014, looking for: a) the most common targets and stakeholders involved in the evaluation; b) the most common goals; c) the most common criteria; d) the most common techniques/methods used for the evaluation; e) the duration of the evaluation.

[1]The identifier follows the format YY#F#ID, where YY denotes the year of publication, F indicates if the publication is a paper ('A') or a poster ('O'), and ID indicates the order in which it was analyzed. The collected data is available at `http://idmil.org/pub/data/dmi_evaluation_nime2012-2014.xlsx`

2.3 Background

The *role of evaluation* has been extensively discussed in the context of HCI (Barkhuus and Rode 2007; Greenberg and Buxton 2008), Creativity Support Tools (Shneiderman 2007) and acoustic musical instruments (Campbell 2014). In the context of DMIs and NIME, discussions are just starting (Johnston 2011) (see, for example, the Workshop on Practice-Based Research in New Interfaces for Musical Expression in NIME 2014[2]). Yet, it is possible to find in literature a large variety of approaches for evaluating DMIs. Here, we provide a brief overview.

Building upon HCI research on the evaluation of 2D input devices (Card, Mackinlay, and Robertson 1991) and on the comparison of input devices for direct timbre manipulation (Vertegaal, Eaglestone, and Clarke 1994), Wanderley et al. proposed to adapt this knowledge to the context of DMIs (Wanderley and Orio 2002). They proposed musical tasks that could allow to quantitatively compare how input controllers perform when considering a certain musical goal.

A different approach, based on the qualitative tradition, was proposed by Stowell and al. (2009). Instead of quantitative comparison, the authors focused on investigating subjective qualities inherent to the musical experience, such as enjoyment, expressivity and perceived affordances. For this, they used semi-structured interviews to collect data with performers, followed by *Discourse Analysis* on the transcribed speech.

Neither do these approaches consider the impact of time on the evaluation (i.e., as time goes by, the more musicians are likely to play and practice with their instruments, and perhaps become better able to express themselves with it). Usually evaluation happens throughout a few sessions, with almost no time interval between them. This issue is addressed by Hunt and Kirk (2000). In their work, they presented an AB Testing based approach (which mixed quantitative and qualitative characteristics) used to evaluate mapping strategies for 3 different DMIs over a period of time.

Another time-related issue is the notion of player's expertise, analyzed both quantitatively and qualitatively by (Ghamsari, Pras, and Wanderley 2013), and its perception by the audience,

[2] `http://www.creativityandcognition.com/NIMEWorkshop/`

as discussed by (Fyans, Gurevich, and Stapleton 2010). Considering the latter, Barbosa et al. presented an evaluation approach that focuses upon the Audience's perspective (Barbosa, Calegario, Teichrieb, Ramalho, and McGlynn 2012). Here, the goal was to assess the participants' comprehension about five components of the instrument, by using an on-line questionnaire.

2.4 Research questions

In order to assess the context of usage of the term "evaluation" by the NIME community, we have set the following research questions:

Question 1: Which targets are evaluated? For example, the whole DMI, its input module, the mapping module, the output module, or the feedback provided by the DMI. In this process, which stakeholders are usually considered?

Question 2: What are the most common goals for DMI evaluation?

Question 3: What criteria are commonly used for evaluating DMIs?

Question 4: What approaches are used for the evaluation (i.e., quantitative, quantitative or both)? What are the most commonly employed techniques/methods?

Question 5: How long do DMI evaluations last on average (i.e., a single session/experiment, or over time)?

2.5 Methodology

We have analyzed all papers and posters available on-line for the last three proceedings of the NIME conference (2012, 2013, 2014). Demos were not considered.

As mentioned before, for each publication we assigned a unique identifier in order to provide practical examples. Then, we collected the following data:

Format: How the work was published (i.e., as oral presentation or poster);

Target: A summary of the main contribution of the publication, using as much as possible the authors' own terminology;

Target category: Classified as: a) DMI; b) Input; c) Mapping; d) Output; e) Feedback; f) Performance. Any other kind of target was classified as "None" as they are outside the scope of this work. One publication can have multiple target categories;

Includes evaluation: Whether or not the authors evaluated the target. For this, we only considered publications in which authors directly used the term "evaluation". If they did not use the term, the publication was not considered.

For those that did evaluate a target (our main interest in this work), we also collected the following data:

Perspective evaluated – According to the stakeholders involved in the design of DMIs (Kvifte and Jensenius 2006; O'Modhrain 2011), what perspective(s) were considered? One publication could address multiple perspectives;

Goal of the evaluation – Here, we tried to use as much as possible the authors' own terminology. However, whenever the name of the target (i.e., the name of the instrument of technology proposed) was mentioned we replaced it with the general term "system";

Criteria considered – Here again, we tried to use as much as possible the authors' own terminology;

Approach – What was the approach chosen towards the evaluation (i.e., quantitative, qualitative, or both)?

Duration – Was the evaluation performed only in a single session/experiment? Or did it occur over time? We did not record specific time durations – if the evaluation lasted several days, weeks, or months, it was categorized as "over time";

Methods – What methods were used to evaluate the target? Here, we tried to add keywords related to the methods employed, with as much details as provided by the authors.

The collected data was gathered in a spreadsheet. For the objective fields (i.e., “Target category”, “Evaluates or not”, “Perspective evaluated”, “Approach”, and “Duration”) we counted the number of occurrences in order to generate tables and graphs. For the more subjective fields (i.e., “Goal of the evaluation”, “Criteria considered”), we initially have employed the word cloud technique as provided by Wordle[3]. In order to extract more details from this data, we did further qualitative analysis. This process is described in the next section.

2.6 Results

From 325 papers analyzed in total, 204 papers were suitable for our purposes (i.e., had DMIs or one of its modules as target). Of these, 89 papers (45 oral presentations & 44 posters) used the term “evaluation” with regards to their target. This result is illustrated in Figure 2.1. The spreadsheet containing all collected and analyzed data is available on-line[4].

In this section we present our results according to each of our five research questions.

2.6.1 Question 1: Evaluated Target

This question regards the most common targets and stakeholders considered in the evaluation. The most common target was the whole DMI (60 publications) and the most common perspective considered was the Performer (52 publications). Results are summarized in Figures 2.2 and 2.3 respectively. In both cases, the classification was non-exclusive (i.e., the same publication could assess different targets and perspectives at the same time).

Regarding the analysis presented in Figure 2.2, it is interesting to note that the number of mapping strategies and output proposed – and consequently evaluated – are low. This might be due to the fact the conference is more focused on “interfaces”, a notion more related to the input

[3] `http://www.wordle.net/`
[4] `http://idmil.org/pub/data/dmi_evaluation_nime2012-2014.xlsx`

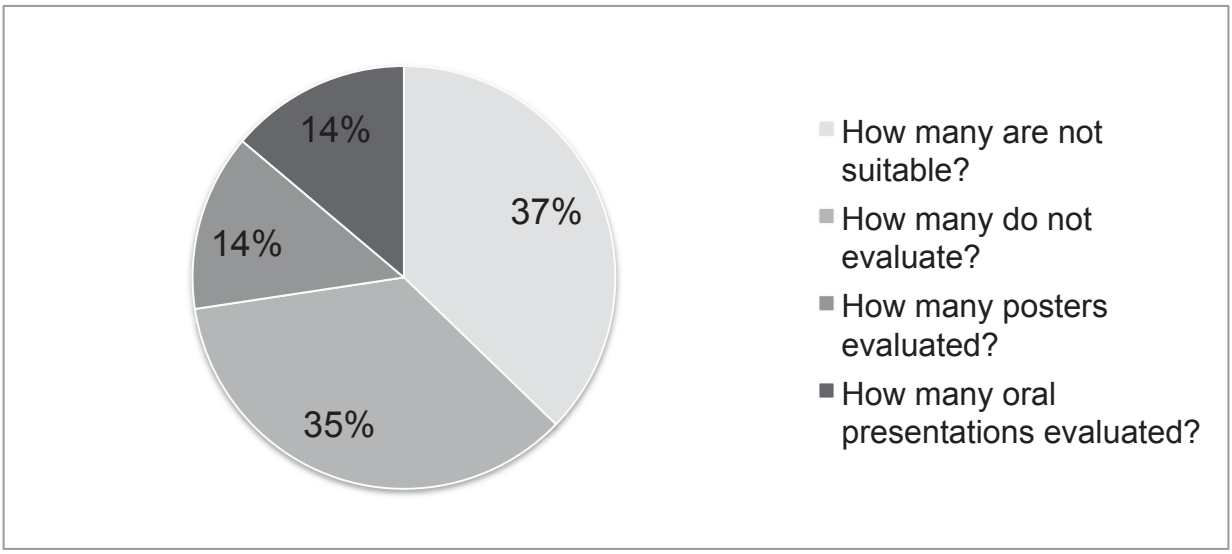


Fig. 2.1 Percentage of reported evaluations according to format (i.e., oral presentations & posters) in NIME proceedings of 2012, 2013 and 2014.

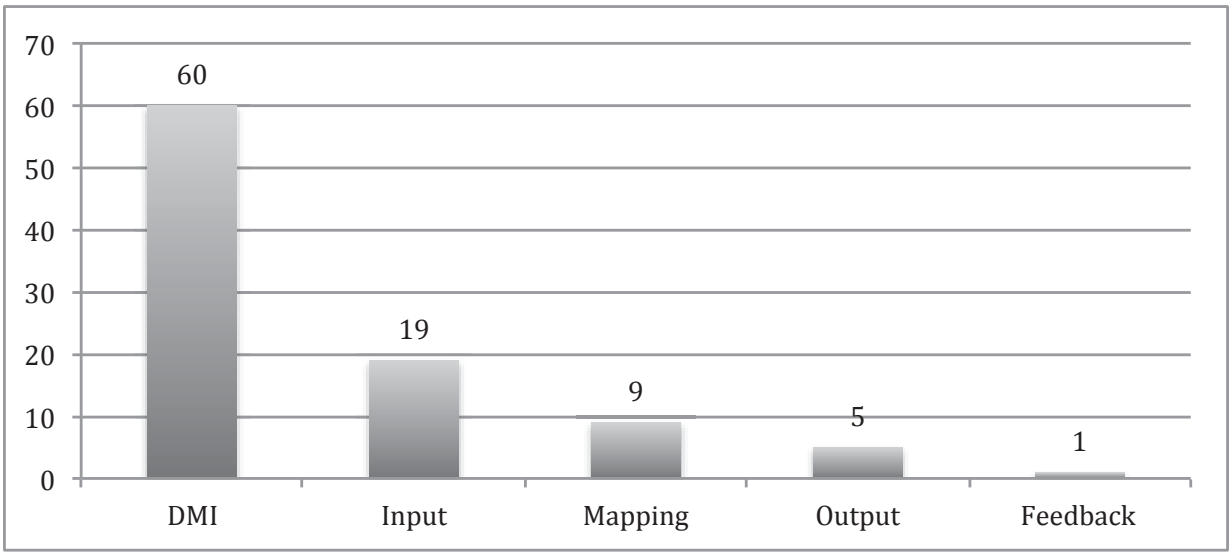


Fig. 2.2 Most common targets used in the evaluations performed.

module, however these numbers are interesting if we consider that mapping has been stated to play a crucial role in the design of new DMIs (Hunt, Wanderley, and Paradis 2003).

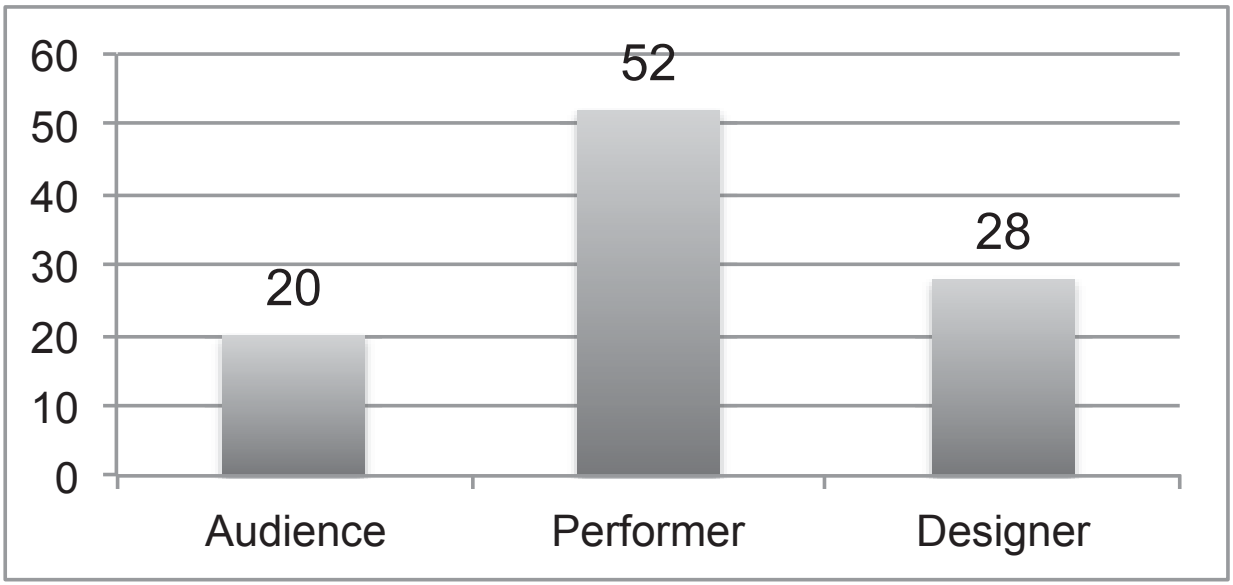


Fig. 2.3 Perspectives considered in the evaluations performed.

As it can be seen in Figure 2.3, the predominance of the *performer's* perspective support the claim that it is the most important stakeholder in musical performance contexts (Birnbaum et al. 2005). The *designer's* perspective is commonly related to the technical aspects of the proposed system (e.g., how effective is a machine learning technique such as in 13#A#47 and 14#O#16, or the frequency response of the sound output such as in 14#O#85). The *audience's* perspective, which is related to how the audience perceives the proposed system (e.g., 13#A#12 and 13#A#11), comes in the last position. These results may indicate that the NIME community tends to under-consider the audience in the design of DMIs, or at least for their evaluation. However, since we consider only papers that report on an evaluation, further investigation is necessary.

2.6.2 Question 2: Goals of the Evaluation

This question addresses the goals the authors aimed with the evaluation. As it can be seen in a word cloud based on collected data (see Figure 2.4), a large variety of terms were employed.

This led us to investigate qualitatively the nature of the chosen goals. We came up with six non-exclusive categories related to the general purpose of the evaluation, defined as follows:

A Investigate how the target performs according to specific pre-defined criteria (e.g., 13#A#7);

B Collect feedback in order to improve the target (e.g., 14#A#48);

C Compare the target with similar systems as baseline (e.g., 14#O#39);

D Verify specific hypothesis about the evaluated target (e.g., 14#O#128);

E Describe interesting (emerging) behaviors while testing the target (e.g., 13#O#66);

F Not specified or **different** from the previous (e.g., 13#O#90).

The goals were then classified according to these categories. The same thing was done separately for each stakeholder perspective. The results are presented in Table 2.2.

Table 2.2 Goals classified according to the six non-exclusive categories proposed. Results also presented for each stakeholder perspective.

Goal	Occurrences	Perf.	Aud.	Des.
A	47	21	10	23
B	18	15	4	1
C	23	12	4	10
D	12	8	3	2
E	25	20	8	4
F	5	3	1	0

We note that 'A' is the most common goal used for all stakeholders. However, it is very common to find goals that are combinations of the above-mentioned categories (e.g., investigate specific predefined criteria and then use this result to compare the target to similar systems, such

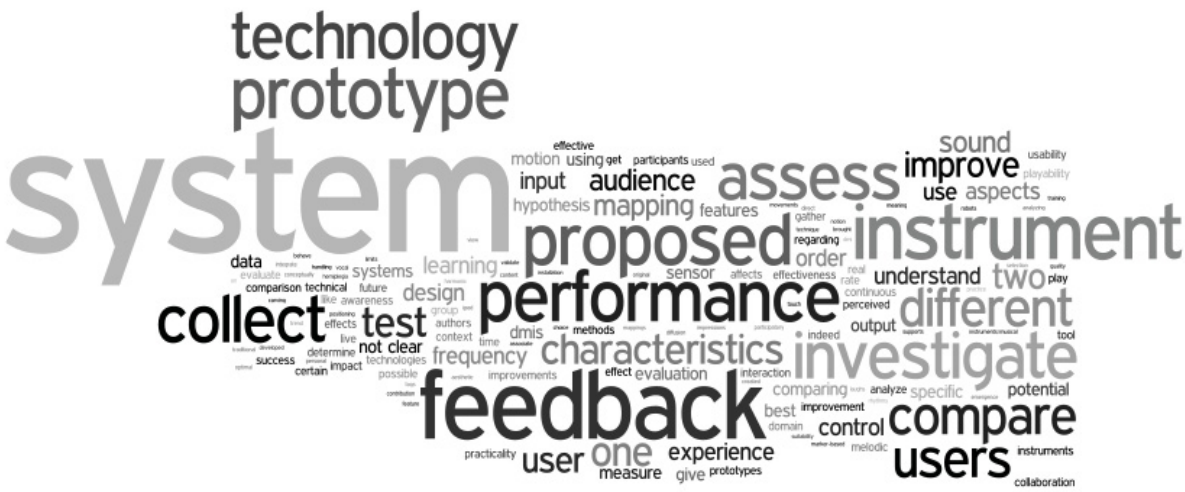


Fig. 2.4 The most common goals found in the reported evaluations. Terms are scaled relative to their frequency in the analyzed text.

as in 13#A#27 and 14#A#10). We also highlight that the same publication can have multiple stakeholders. Finally, it is interesting to note how diverse are the goals hidden behind the term “evaluation” on NIME literature.

2.6.3 Question 3: Criteria

This question involves the most common criteria used for the evaluation. At first, for each stakeholder perspective, we have built a word cloud based on the collected data. As shown in Table 2.5, a large amount of publications omitted this information, and the term “not clear” was very large, hiding the rest of our data. This motivated us to remove it from the word cloud, as presented in Figures 2.5, 2.7, 2.6. At the same time, the fact seems representative, as it illustrates the lack of consistency regarding evaluation criteria in the NIME community.

Considering the Performer’s perspective (Figure 2.5), we can note that some terms emerge despite the large diversity. Most part of them were already addressed in the literature, such as *‘engagement’* (Wessel and Wright 2002), *‘effectiveness’* (Jordà 2005), and *‘expressiveness’* (Arfib, Couturier, and Kessous 2005). However, these criteria are still subjective in the context of DMIs

Fig. 2.5 The most common criteria according to the Performer's perspective ("not clear" excluded).

and there is no consensus on how to measure or analyze them. Considering the Designer's perspective (Figure 2.6), objective terms like *'precision'*, and *'latency'* emerged. Considering the Audience's perspective (Figure 2.7), there was no significant difference regarding the usage of the terms (i.e., terms such as *'focus'* and *'intention comprehension'* were mentioned only twice). In all three cases, the large diversity of terms should be highlighted.

In order to investigate the nature of these criteria, we further performed a qualitative investigation of the data. We classified the criteria as *objective* (i.e., there is a clear understanding on how to measure these criteria), *subjective* (i.e., there is no clear understanding on how to measure these criteria) or *both*. The result is summarized in Table 2.3. Subjective criteria and "not clear" were the most commonly found categories. Once again, the lack of consistency regarding criteria is apparent.

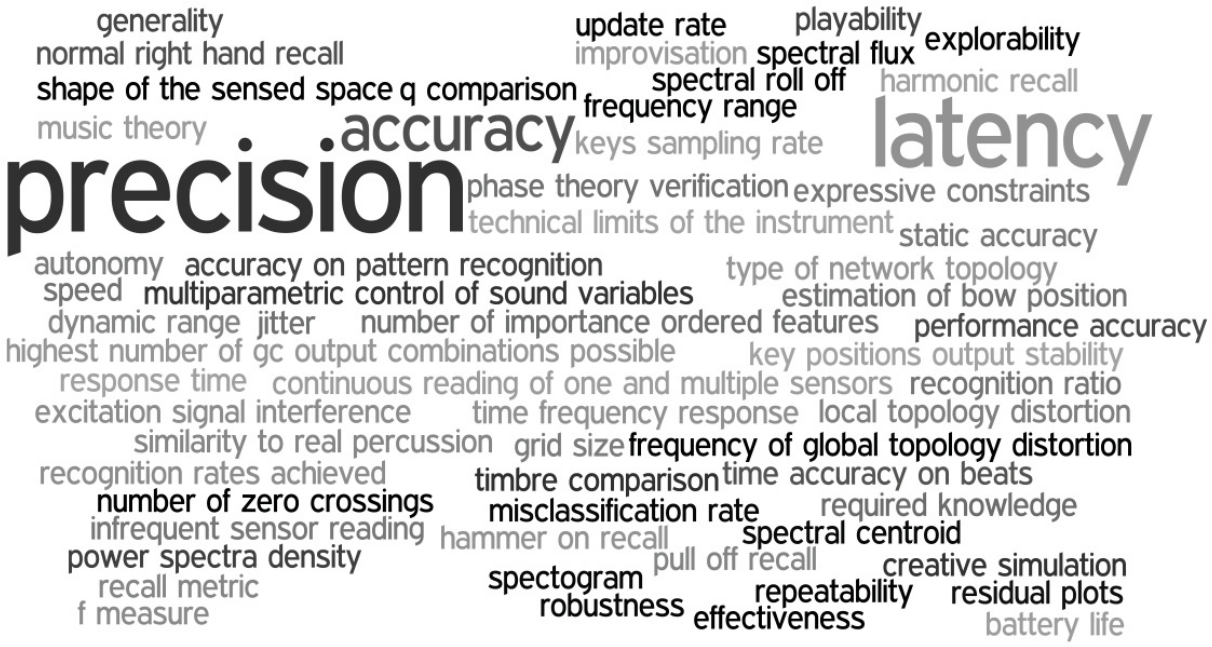


Fig. 2.6 The most common criteria according to the Designer's perspective ("not clear" excluded).

Table 2.3 Criteria classified in subjective, objective, both or "not clear" (i.e., not able to determine).

Criteria	Number of occurrences
Subjective	29
Objective	27
Both	7
Not clear	25

Fig. 2.7 The most common criteria according to the Audience's perspective (“not clear” excluded).

2.6.4 Question 4: Approach

This question investigates the approach chosen (i.e., *qualitative*, *quantitative* or *both*) and the most common techniques or methods employed. Table 2.4 summarizes the results. Regarding the techniques/methods, once again, we have built word clouds based on the collected data. The results are presented in Figures 2.8, 2.9, 2.10.

Table 2.4 Results regarding the evaluation approaches chosen.

Approach	Occurrences	Perf.	Aud.	Des.
Qualitative	28	23	8	1
Quantitative	36	9	5	23
Both	13	11	7	1
Not clear	12	9	0	3

Fig. 2.8 The most common methods/techniques employed according to the Performer’s perspective

Although the quantitative approach was the most commonly used, we found less variety among the qualitative related methods (e.g., questionnaire, and interviews). It is also interesting to note that qualitative approaches were more common when evaluating the Performer’s perspective. On the other hand, quantitative approaches were preferred when evaluating the Designer’s perspective.

2.6.5 Question 5: Duration

The last question assessed the duration of the evaluation (i.e., single session/experiment, or over time). Regarding this, most part of the evaluation (66%) seems to be performed in a single

Fig. 2.9 The most common methods/techniques employed according to the Audience's perspective.

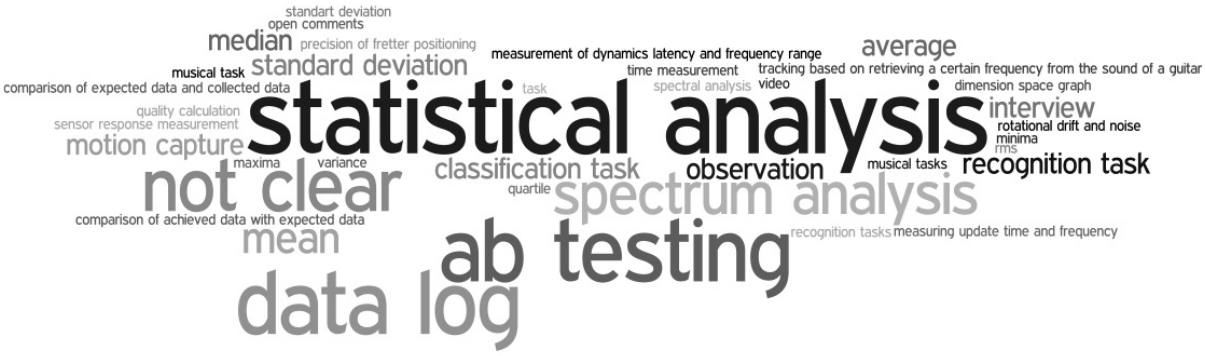


Fig. 2.10 The most common methods/techniques employed according to the Designer's perspective.

session. Evaluations over time occurred in some cases (19%), but they were much less common. The remaining (15%) were not clear about the subject.

2.7 Discussion

Surprisingly, we can notice a significant number of publications that employ the term “evaluation” without giving any detail about criteria (31%) or methods (19%). In some rare cases (4%), even the goal is not clearly stated. This result is shown in Table 2.5.

We believe that this is the most important issue and that it deserves attention – especially from NIME reviewers. It is completely acceptable to not evaluate the DMIs we create, however, if one wants to “evaluate” something, it is essential to provide basic information such as the goal of

Table 2.5 Quantifying omitted information.

Total of evaluations	**89**
Do not inform which methods were used	17
Do not inform which criteria were used	28
Do not inform goals for the evaluation	4

the evaluation, how it was performed (e.g., what methods and criteria were used) and what results were achieved. Otherwise, the information provided is not meaningful to the community. More importantly, this is harming the validity of the evaluation and prevents its replicability (Greenberg and Buxton 2008).

In addition, these results provide us a clearer view of different approaches towards "evaluation" by the NIME community. The data allowed us to picture the profile of a typical evaluation (i.e., evaluates the DMI according to the performer's perspective, in a single qualitative experiment), for which literature offers several different possible approaches. However, how can we address the remaining cases, such as the *audience* perspective, or evaluation over time?

Another interesting issue concerns the lack of agreement about criteria and goals used for evaluation. This provides us with some interesting questions for future research, for example: considering a given goal and stakeholder perspective, would it be possible to find a common understanding about the most important criteria? Will it be possible for us to find a consensus approach to analysing subjective criteria, such as playability, engagement, and expressiveness?

2.8 Problems & Limitations

During our analysis, we faced some issues, the most relevant ones being that:

- It was sometimes hard to classify a target according the categories we were looking for (i.e., DMI, Input, Mapping, Output, and Feedback), such as in 14#A#48;
- The difference between evaluation and experiment (in which hypotheses needed to be demonstrated) is not clear, such as in 14#A#49;

- In the process of extracting subjective fields (i.e., the goals) of the evaluations, some bias may have been introduced since the data we were looking for were not always clearly described (e.g., difficult to say what method/goal/criteria the authors used). We tried to minimize this bias by using the authors' own terminology as much as possible. This problem will be difficult to solve, as it is related to the way the evaluations were reported in the publications;

- Considering publications with multiple stakeholders (such as 12#A#23), we did not differentiate which methods and criteria were set for each stakeholder. This fact introduced some noise to our cross-question analysis (i.e., Questions 3 and 4, in which we have created one word cloud for each stakeholder perspective).

In addition, we stress that the results presented and discussed in this paper are still preliminary. Going forward, further years of NIME proceedings should be considered, as well as other relevant venues, such as the ICMC.

2.9 Conclusion

We have investigated how the term "*evaluation*" has been employed in the NIME literature. The results give us a better idea of: a) the most common targets and stakeholders considered during the evaluation; b) the most common goals set; c) the most common criteria set; d) the most common techniques/methods used for the evaluation; and e) how long the evaluation lasts.

In case one is interested in evaluation within a certain context (e.g., what would be the most used techniques for evaluating mapping considering audience's perspective?), we highlight that cross-relating results (like we did in Questions 3 and 4) can provide a richer analysis scenario.

Finally, although "*there is no one-size-fits-all solution to evaluating DMIs*" (O'Modhrain 2011) and more precisely "*the choice of evaluation methodology - if any - must arise from and be appropriate for the actual problem or research question under consideration*" (Greenberg and Buxton 2008), this work may help us to assess different evaluation profiles in order to find the most suitable techniques considering different goals, criteria and stakeholder's perspectives. Thus, we hope

to contribute by going beyond discussing whether the NIME community should or should not evaluate their creations, focusing instead upon how could we make better use of evaluation and what criteria should be used for the evaluation.

2.10 Acknowledgments

The authors would like to thank: The NSERC Discovery Grant, Inria/FRQNT/the Embassy of France in Canada (Équipe de Recherche Associée *MIDWAY*), for the funding; the anonymous reviewers, for their valuable comments and suggestions; and Johnty Wang, Carolina Medeiros, and John Sullivan.

Chapter 3

Exploring Playfulness in NIME Design: The Case of Live Looping Tools

Chapter 2 addressed evaluation and its role for NIME design. From now on, this thesis starts introducing three unique case studies where direct manipulation is designed and evaluated in different musical contexts. The first case study, presented in this chapter, deals with live looping. This chapter has been previously published as:

> `Jeronimo Barbosa`, Marcelo M. Wanderley, and Stéphane Huot. *"Exploring Playfulness in NIME Design: The Case of Live Looping Tools."* In Proceedings of the 2017 International Conference on New Interfaces for Musical Expression, Copenhagen, Denmark.

3.1 Abstract

Play and playfulness compose an essential part of our lives as human beings. From childhood to adulthood, playfulness is often associated with remarkable positive experiences related to fun, pleasure, intimate social activities, imagination, and creativity. Perhaps not surprisingly, playfulness has been recurrently used in NIME designs as a strategy to engage people, often non-expert,

in short term musical activities. Yet, designing for playfulness remains a challenging task, as little knowledge is available for designers to support their decisions.

To address this issue, we follow a design rationale approach using the context of Live Looping (LL) as a case study. We start by surveying 101 LL tools, summarizing our analysis into a new design space. We then use this design space to discuss potential guidelines to address playfulness in a design process. These guidelines are implemented and discussed in a new LL tool–called the "Voice Reaping Machine". Finally, we contrast our guidelines with previous works in the literature.

3.2 Introduction

From childhood to adultness, playfulness is often associated with remarkable positive experiences related to fun, pleasure, intimate social activities, imagination, creativity.

In the NIME context, in particular, playfulness has been used as a strategy to engage people, often non-expert, in musical activities (Keating 2007; Robson 2002; Troyer 2012). In addition, at least two other reasons make us believe that playfulness could be a relevant topic for NIME research.

Firstly, playfulness has been linked to several positive aspects potentially useful in music, such as creativity (Russ 2003; Zabelina and Robinson 2010)–arguably essential for any artistic activity. Similarly, in the context of computers, some potential benefits include improved performance, potential to improve learning, higher user satisfaction and attitudes, and positive subjective experiences (Woszczynski, Roth, and Segars 2002).

Secondly, for the particular case of NIME, because classical approaches towards learning (e.g. pedagogical methods, teachers, and schools) are almost inexistent, we believe playfulness could be a useful strategy for engaging people in practice, yielding in the development of skills in the long term. This direction is suggested by Oore (2005) when sharing his personal experiences in learning two NIMEs. Similarly, in sports and psychology literature, some authors highlight the importance of play–in addition to practice–for the development of expertise (Côté, J. Baker, and Abernethy 2012).

Despite this relevance, designing for playfulness remains a challenging task, as little knowledge is available for designers to support their choices. For example: in case we want to design a playful NIME, how should we proceed? How can we make sure we are properly addressing playfulness in our design? In other words, what kind of design decisions should one make so that the resulting NIME is playful?

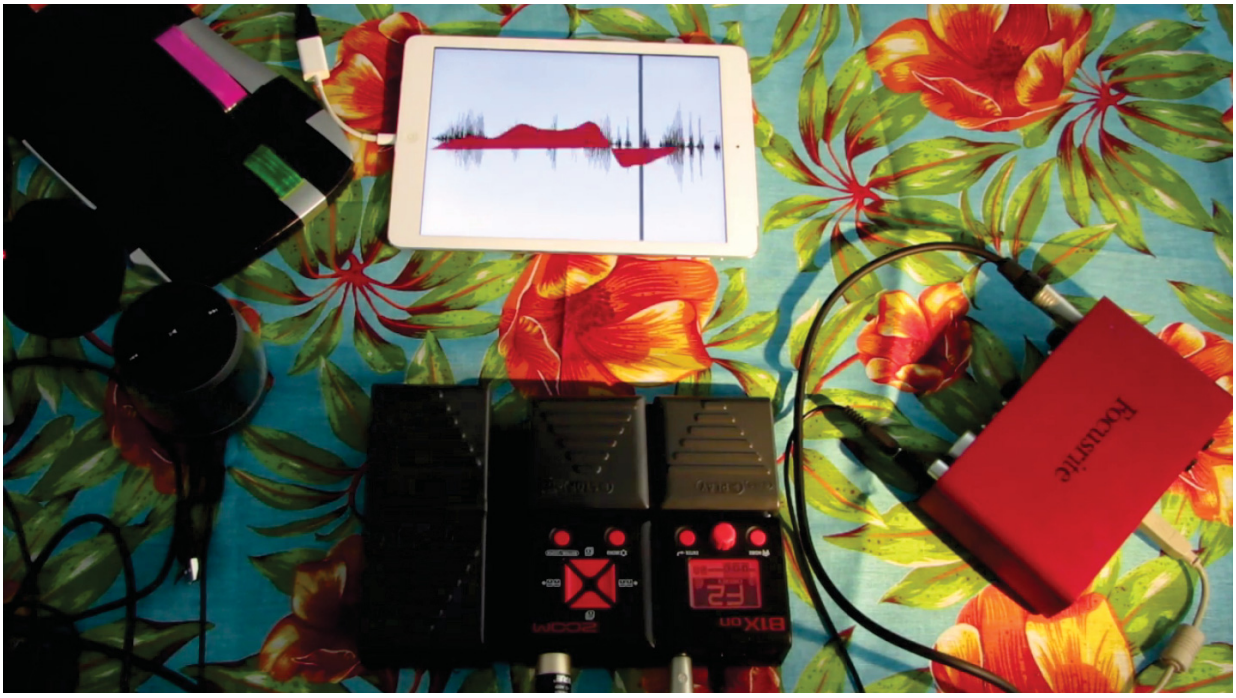


Fig. 3.1 The Voice Reaping Machine.

These are the questions we try to address in this paper by exploring the notion of playfulness in the NIME context. For this goal, we follow a design rationale approach, focusing on Live Looping (LL) as a case study. To start, we survey 101 LL tools, summarizing our analysis into a new design space. We use this design space to discuss potential guidelines to address playfulness in a design process. These guidelines are implemented and discussed in a new LL tool–called the "Voice Reaping Machine", presented in Figure 3.1. Finally, we contrast our guidelines with previous works in the literature.

3.3 Background

Playfulness is an ambiguous term with different usages in the literature. For instance, Nijholt (2014) defines that interfaces are playful when "users feel challenged or are otherwise persuaded to engage in social and physical interaction because they expect it to be fun." Alternatively, video game researchers (Boberg et al. 2015; Lucero et al. 2014) address playfulness as a quality that enriches products' market value via a set of pleasurable user experiences, such as sympathy, relaxation, or nurture.

For the context of this work, we define playfulness as *a context-specific tendency that leads people to engage in voluntary, inventive, and imaginative interactions with a system* (Webster and Martocchio 1992; Woszczynski, Roth, and Segars 2002). In this sense, it relates to open-ended interactions driven by individuals' intrinsic motivation, where enjoyment is the ultimate goal.

The notion of playfulness has been extensively discussed in the Human-computer Interaction (HCI) context. A detailed survey is beyond the scope of this research–we recommend (Woszczynski, Roth, and Segars 2002) for further details. Here, we focus on a central issue: the state/trait nature of playfulness.

An early study to address playfulness–as defined here–in the context of HCI is (Webster and Martocchio 1992). The construct is defined as a trait–a pattern in an individual's behavior that is recurrent over time. In other words, playfulness was considered as a characteristic present in certain personalities that yields a predisposition to be playful–no matter the tools used in the interaction or the context. This characteristic is also known as "autotelic personality" (ibid.).

On the other hand, some authors argue that playfulness could potentially be addressed as a state–a short-timed condition, where several context–specific factors related to the interaction (e.g. the difficulty of the task executed, the ability of the individual in dealing with that tasks) could impact the achievement of this playful state. In this case, the assumption that the context of the interaction has little or no impact on playfulness is no longer valid: in fact, here the context becomes critical.

Considering playfulness as a state, (Webster, Trevino, and L. Ryan 1993) suggest that playfulness could be investigated through the lenses of the flow theory (Csikszentmihalyi 2000). This theory relies on the flow state: an optimal condition, characterized by a high degree of enjoyment and involvement, in which people become totally immersed in performing an activity due to a perfect balance between the challenges offered by the tasks and the skills possessed by the individual. If the activity is too easy, it yields boredom; if too challenging, it yields anxiety. Dimensions that compose flow include: (1) Control, defining the sense of control users feel over the process and the activity's outcome; (2) Attention focus, defining to what extent users were distracted or absorbed while performing the activity; (3) Curiosity, representing the degree of imagination and curiosity stimulated while performing the activity; and (4) Intrinsic interest, defining to what extent users were voluntarily engaged and motivated by the activity.

3.3.1 Playfulness in NIME

There is a little amount of research dedicated to playfulness–as defined here–in the context of NIME. In addition, for some of the cases that do address playfulness (Keating 2007; Troyer 2012), little effort is made to either justify design decisions or discuss which characteristics have made that particular design playful. Here, we cover two works that go beyond this limitation.

The first one is (Robson 2002). In this practice-led research, playfulness was used as central guideline for designing simple fun–focused musical interfaces (i.e. toy-like instruments), aiming at non-musicians.

Four projects are presented: (1) The Piano cubes, two square jam jar embedded with tilt sensors, each one mapped to the direction and the tempo of a four-notes piano arpeggio; (2) The Bullroarer, a digital version of this ancient musical instrument composed of a piece of wood and a rope; (3) The Stretch, a latex rubber surface embedded with slider variable sensors in a square frame; and (4) The "When I think of heaven", a wall-sized square instrument that combined four different Stretch interfaces with two drum pads. For each of these, the author discusses motivation and evolution of the designs, summarizing his experience in three key conclusions: (1) Novices

tend to enthusiastically explore playful interfaces when these are open ended and hide user's lack of expertise; (2) Social collaboration can be useful to encourage play; and (3) Mapping is one of the most challenging steps in designing playful interfaces.

It is important to note that playfulness here is limited to toy-like interfaces which, although fun and simple to use, could potentially be quickly mastered and forgotten by users. This characteristic is often undesirable in the context of NIME (Wessel and Wright 2002).

Another example, more recent, is (A. P. McPherson et al. 2016). The author introduces the D-Box, a simple and straightforward (i.e. only three basic sensors are available for performers to interact with) DMI that, despite its simplicity, was purposely designed to be opened, and modified (i.e. hacked) by its users, aiming at playful engagement in this hacking process. In this sense, the author follows the idea of designing for ludic engagement, as defined by (Gaver et al. 2004).

This DMI was investigated over two workshops in the UK with 17 diverse-backgrounded participants focused on understanding how they use the D-Box. Results–presented according to three stages: before participants opened the D-Box, during the hacking, and after D-Boxes were anonymously exchanged among participants–focused on issues such as: (1) Sense of ownership after the hacking (i.e. participants reported connection to their own hackings, and disappointment with the one received in the exchange); (2) Patterns in the exploratory behavior when hacking the instrument (i.e. the caution random walk); and (3) The limited initial affordances. Little is said, however, about the design process of the instrument, and about how playfulness was built throughout this process.

3.4 Methodology

Our work has four basic underlying assumptions:

1. Playfulness is a state, and because of that, people can be more prone to playfulness depending on the characteristics of the context;

2. Performers can achieve a state of playfulness in the context of musical practice with NIME;

3. The NIME itself plays a role in achieving this playful state (i.e. it is possible to design instruments that are more "playfulness inducers" than others); and

4. We can foster this playfulness by addressing the conditions that might lead to *flow*.

Framed by these assumptions, our goal is to explore how to design NIMEs that effectively facilitate playfulness among performers. For this, we decided to use a design rationale-inspired methodology (MacLean, Young, and Moran 1989), aiming at describing and reasoning our decisions throughout our design process. Our methodology is based on five steps: a) *choose a case study*; b) *survey of existing tools*; c) *create a design space*; d) *exploring potential guidelines for achieving playfulness*; and e) *iteratively prototyping solutions*. These steps are presented in the following sections.

3.4.1 Choose a case study

For the scope of this study, we decided to focus on the context of Live looping (Berthaut, Desainte-Catherine, and Hachet 2010). Live looping is a musical technique based on looping audio samples recorded in performance time by the performer himself/herself.

We believe that choosing LL as case study is beneficial for two reasons.

First, because LL tools share a standard set of core functionalities (e.g. record, play, stop, and overdub), different LL tools might allow performers to achieve the same kind of musical results (i.e. one performer could likely replicate the same musical excerpt in different LL tools). How these functionalities are implemented, however, (e.g. a pedal, a desktop application, etc.) is specific to each individual tool. We believe this restriction is essential to allow comparison between different implementations.

Second, artists such as Reggie Watts[1] and Dub Fx[2] demonstrate how LL tools afford a new particular set of skills, built upon their skills with their musical instruments (in the case of these

[1] `https://reggiewatts.com/`
[2] `https://dubfx.com/`

artists, the voice). This new set of skills suggests that LL tools can be considered as musical instruments by themselves, and are therefore representative as case study.

3.4.2 Survey of existing tools

There is a wide variety of devices that implement LL. As a first step, we have surveyed and analyzed live looping tools produced by the music technology industry, academic studies, and independent developers. In total, 101 tools were surveyed. The result of this survey and analysis is available on Appendix A.

This survey allowed us to get a sense of how designers approached the design of new LL tools, especially concerning similarities and differences of each one. This allowed us to develop the design space (Birnbaum et al. 2005; Graham et al. 2000) introduced in the next subsection.

3.4.3 Create a design space

From our survey of LL tools, we identified five dimensions representing relevant aspects of LL tools. We planned to make these dimensions orthogonal, with little or no intersection between them (the only exception concerns the visual feedback). These dimensions are:

Looping capabilities: Defines the range of musical possibilities provided by the looping device. It is a continuous scale that ranges from *basic* (set of standard functionalities consisting of record, overdub, play, stop, and delete, as implemented in the Boss RC-1) to *advanced* (e.g. individual layer control, as in the Loopy) functionalities;

Input capacity: Defines the amount of standard input controls visible to the user for the interaction (e.g. buttons, knobs, touch screen, etc). This dimension relates to the notion of input capacity as defined by (Graham et al. 2000), concerning the capabilities of an input device for capturing information from user interaction. Here, it is represented by a continuous scale from *low* (as in the Ditto Looper, which has only a foot-switch and a knob) to *high* (as in the Roland MC-09, which provide approximately 51 buttons, 8 knobs, and 4

sliders);

Mapping directness: Defines how the looping functionalities are made accessible for the user via the input controls. This accessibility can be *direct* (i.e. similar to an one-layer mapping (Hunt, Wanderley, and Paradis 2003)), when a certain functionality is directly accessible to users when they use a input control (e.g. pressing a foot-switch to record and overdub in the Vox Lil'Looper). Contrarily, this accessibility can be *indirect* (i.e. similar to a multiple-layered mapping (ibid.)), when users need to navigate in the interface until the point they are able to either enter a "looper mode" or find the desired functionality (as found in multi purpose software such as the Ableton Live);

Visual feedback role: Defines what role visual feedback plays in the looper. This role can be: a) *Limited*, where visual feedback–if present–happens only when user interacts with input controls (e.g. the Digitech DL-8); b) *transparent*, where visual feedback allows users to quickly infer the current status of the system (e.g. the Boss RC-1); and c) *ornamental*, where visual feedback works as aesthetic decoration for the device, with no correspondent in terms of functionality (e.g. the drawing aspects of the Illusio);

Visual feedback intensity: Defines how much of visual feedback the device can provide, ranging from *low* (e.g. the single small LED provided by the TC Ditto Looper), to *high* (e.g. the full monitor screen visual interface of the Freewheeling).

3.4.4 Explore potential guidelines for achieving playfulness

Considering this design space, our surveyed tools, and previous works in the literature, how can we design live loopers that facilitate playfulness? We propose three key guidelines presented in the following subsections.

Advanced looping capacity

All live looping tools share a basic set of functionalities that are at the core of live looping performance (i.e. record, overdub, play/stop, and clear). Few are, however, the tools that go beyond this basic set, expanding the musical possibilities of live looping.

In order to address the conditions that may lead to flow (and therefore promoting playfulness), we argue that providing advanced looping capacity is essential. Curiosity raised by the new musical possibilities might yield exploratory use, which has been linked to flow's intense concentration and enjoyment (Ghani and Deshpande 1994). Furthermore, advanced functionalities may afford the development of new skills for expert users, without compromising the basic functionalities for novice users. This aspect can help users find their own balance between challenge and skills–essential for achieving the flow state.

One straightforward design strategy for implementing high looping capacity is either to incorporate extra functionalities provided by existing tools or to brainstorm innovative functionalities not yet addressed these existing tools. Another strategy is to "absorb" expert techniques inside the tool–as suggested by Cook's third principle (Cook 2001).

Low input capacity and direct mappings

Providing low input capacity means reducing the number of standard input controls immediately visible to users for the interaction. Additionally, providing direct mappings means that functionalities–both basic and advanced–should be directly accessible via input controls.

The motivation is trying to make the device easier to get started with by: a) reducing confusion that a high number of input controls may cause to new users; and b) coupling the reduced number of input controls with the usage of direct mappings (e.g. arguably, providing a single button for navigating many different functionalities could result in more challenging initial ease of use). This aspect, again, could help users with different levels of expertise to find the balance between challenge and skills required for flow.

Another argument is that, because mappings are direct, performers could spend less time dealing with actions that do not have a direct impact in the musical outcome (e.g. interface navigation). As a result, we allow them to better focus their attention on the musical activity–another key requirement for the flow state.

It is important to note that this guideline does not necessarily yield NIMEs that are "easy to use" or "easy to master". The number of input controls can be low and their mapping direct, but still it may be challenging to meaningfully control them in a musical sense. A practical example is the theremin, which, despite intuitive and easy to get started, is arguably hard to master. This idea of instruments with "low entry fee and high ceilings", discussed in (Wessel and Wright 2002).

Transparent and intense visual feedback

This guideline means that visual feedback should be provided and guided towards: a) allowing user and the audience to infer what is going on inside the device (i.e. visual feedback should be transparent); and b) exploring highly visible visual displays as output, so that they can be easily perceived by performer and audience.

The importance of visual feedback for NIME has been discussed by several previous works (Dobrian and Koppelman 2006; Schloss 2003). We believe this guideline may facilitate playfulness and flow because it may help in promoting transparency–that is, how much people (mainly the audience and non-expert users) perceive the connection between the performer's gestures and the sounds produced (Fels, Gadd, and Mulder 2002). Furthermore, visual feedback also affords potential to make the tool more intuitive (Jordà 2003), contributing for initial ease of use.

Some concrete strategies on how to implement this guideline can be found in the literature. Examples include using visual metaphors, and exploring perceptual sound parameters (e.g. loudness) (Arfib, Couturier, and Kessous 2005).

3.4.5 Iteratively prototype solutions

The previously mentioned guidelines led us to develop low-fi prototypes of new live looping devices–represented in Figure 3.2[3]. Some of them evolved to the "Voice reaping machine", that is presented in the next section.

3.5 Implementation

The LL tool designed to implement these guidelines is called the Voice Reaping Machine (VRM)–already presented in Figure 3.1. The tool is composed of: a) an iPad application developed in C++/Openframeworks; and b) a modified-keyboard that works as two foot-switches.

In the following subsections we discuss how we implemented each guideline in this particular prototype.

3.5.1 Advanced looping capacity

In addition to the standard basic functionalities, the VRM presents three innovative functionalities when compared to existing LL tools: a) the capacity of easily setting the playback position of the looping; b) the capacity of easily resetting a new looping area inside the original loop; and c) the capacity of creating additional voices to the loop, by combining either two playback positions or two looping areas playing together at the same time. This process is shown in Figure 3.3.

The combination of these functionalities makes the VRM unique when compared to existing LL alternatives, providing it with a peculiar advanced looping capacity.

3.5.2 Low input capacity and direct mappings

Concerning the standard basic functionalities, the VRM emulates the foot pedal-based interaction style used by the most simple loopers in our survey. As a consequence, all standard basic functionalities are directly accessible via the modified keyboard. For example, to record, the user

[3]Videos of these prototypes can be found in: `https://youtu.be/7OpCP26LXxA`; `https://youtu.be/CAiVWvVFaqI`; and `https://youtu.be/oRpVfqern6s`

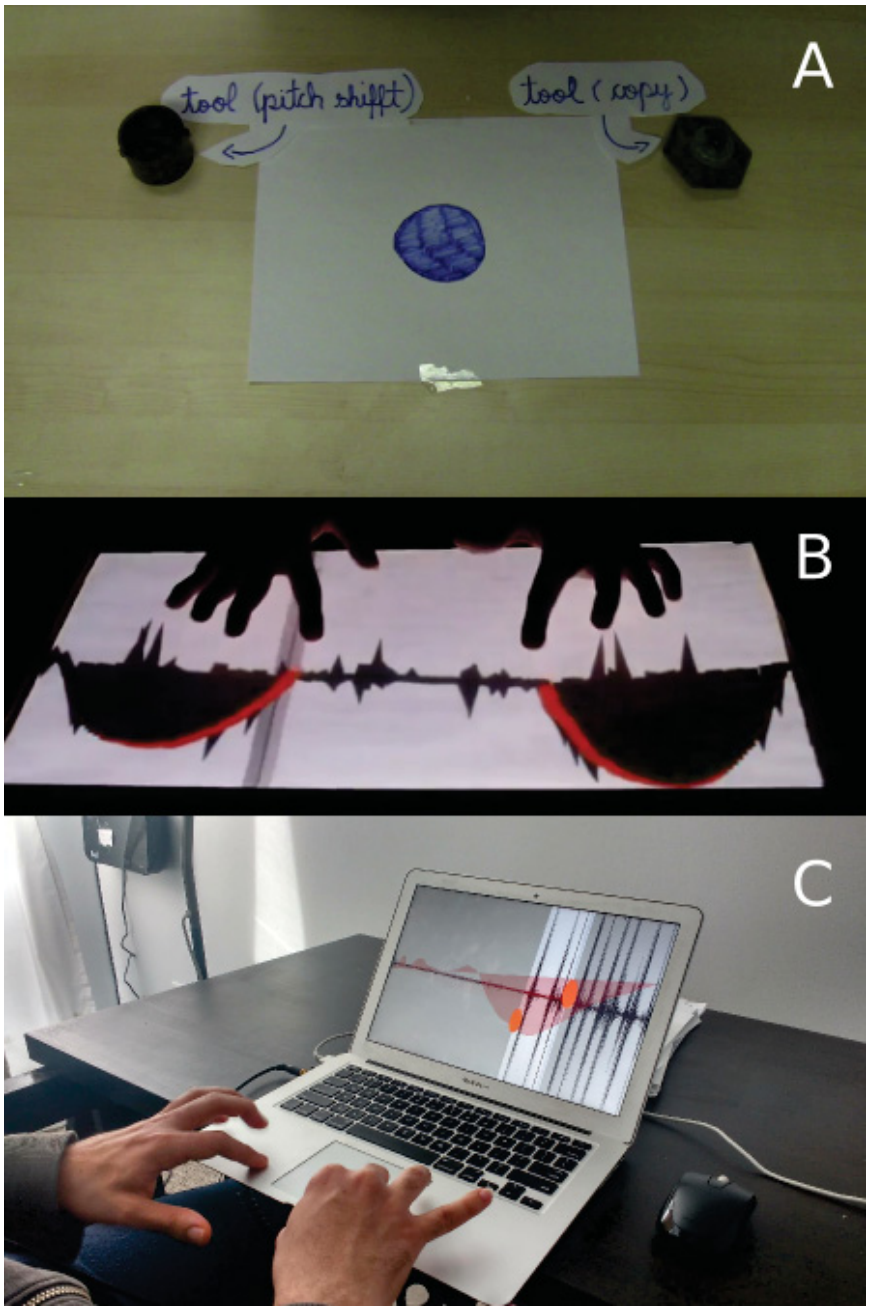


Fig. 3.2 Several prototypes were designed for exploring our guidelines. In addition to paper sketches, we also built (A) a video prototype; (B) a functional prototype using a DIY multitouch table; and (C) another functional prototype using the computer's trackpad.

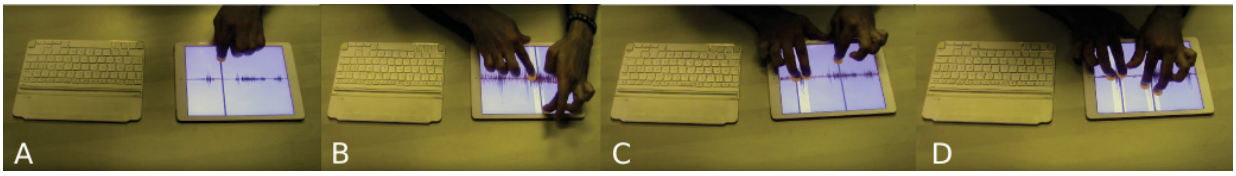


Fig. 3.3 The advanced functionalities introduced by the VRM represented in terms of its incremental touch interaction: (A) one finger in the tablet screen results in control of the playback position; (B) two fingers result in redefining the looping area; (C) three fingers result in controlling the playback position *and* redefining the looping area; and (D) four fingers result in two looping areas playing in parallel.

Table 3.1 Contrasting different guidelines for playful NIMEs

Work	NIME	Guidelines
Robson (2002)	*The Piano cubes*; the *Bullroarer*; the *Stretch*; and the *"When I think of heaven"*	(1) Novices tend to playfully explore interfaces when these are open ended and hide user's lack of expertise (2) Social collaboration can be useful to encourage play (3) Special focus on mapping because it is one of the most challenging steps in designing playful interfaces.
Troyer (2012)	The *DrumTop*	(1) Explore everyday gestures and objects for interaction (2) Explore the natural feedback provided by these physical objects
McPherson et al. (2016)	The *D-Box*	(1) Initial simplicity and limited input capacity (2) Purposely designed to be opened, and modified (i.e. hacked) by its users, affording more complex interaction
Our guidelines	The *Voice Reaping Machine*	(1)Provide advanced looping capacity (2) Provide low input capacity and direct mappings (3) Provide transparent and intense visual feedback

needs to press the foot-switch at the beginning and again at the end points of the musical phrase to be looped. The same actions allow overdubbing if executed whenever some musical material is looping. Play and stop can be also be directly accessed via the foot-switch. Clearing is possible by holding the main foot-switch by two seconds.

Regarding the three advanced functionalities, they were made accessible via incremental touch interaction and direct manipulation of the object of interest (the sample), as follows:

One finger added: The user is able to control the playback position of the looper (the finger X position), and the volume of the playback (finger Y position);

Two fingers added: The user is able to set a looping subarea, where the begin position is the left-most finger X position, and the end position is the right-most finger X position. In both case, volume is defined by interpolation of the Y position of both fingers;

Three fingers added: In this case, users can perform both actions defined above at the same time. In other words, they can: a) set and control a looping subarea (as in 'two fingers added'); and b) control the playback position of the looper (as in 'one finger added');

Four fingers added: Here, users are able to select two looping subareas (as a doubled 'two-fingers added').

3.5.3 Transparent and intense visual feedback

Visual feedback is at the core of the VRM, and was designed in order to highlight the high level mechanisms of the looper (Berthaut, Subramanian, et al. 2013). It basically consists of a timeline showing the waveform of the recorded loop and a gray line indicating the playback position of the loop. In addition, live audio input is provided by the interface, in order to allow input monitoring.

All elements are responsive to user actions (e.g. recording or overdubbing changes to background color to red, areas outside a looping subareas are made gray, and so on), allowing performer and audience to infer accurately what is happening inside the device (i.e. transparency). For this

goal, all the tablet screen is dedicated to the interface, in order to maximize visual feedback intensity.

3.6 Discussion

We note that there is still some open questions raising from our work. For instance: If the VRM is more playful than other LL tools, is it because of the strategies we used? What consequences would this playfulness bring for the LL practice–in particular, for user engagement, willingness for practice, and skill development?

To clarify these questions, further empirical investigations are needed. Such studies would complement the *formative evaluation* used here, derived from our design rationale-inspired methodology, and would allow us to concretely assess strengths and weaknesses of the VRM.

Finally, our guidelines were contrasted to other guidelines from the literature. This contrast is summarized in Table 3.1.

Despite the different contexts (musical toys, live looping, etc), it is interesting to note how the idea of *simple interaction* seems somehow always present–by allowing novices to simply produce musical results above their capacity (Robson 2002); by exploring everyday objects potentially familiar to users (Troyer 2012); or by providing a low input capacity as proposed here. We believe further research is needed in this direction.

3.7 Conclusion

In this paper, we have explored the notion of playfulness for NIMEs, specifically in the context of live looping tools. Our main contributions are: a) a survey and analysis of 101 existing LL tools; b) the definition of a design space for LL; c) a set of design guidelines for playful LL tools, using our design space as baseline; and d) a practical implementation of these guidelines in a new LL tool, the "Voice Reaping Machine". In the larger picture, we hope these contributions can provide some preliminary knowledge on how to effectively address playfulness in a NIME design.

3.8 Acknowledgements

The authors would like to thank: The NSERC Discovery Grant, Inria/FRQNT/the Embassy of France in Canada (Équipe de Recherche Associée *MIDWAY*), for the funding; the anonymous reviewers, for their valuable comments and suggestions; and Catherine Guastavino, Gary Scavone, Isabelle Cossette, and Eleanor Stubley.

Chapter 4

Designing Behavioral Show-Control for Artistic Responsive Environments: The Case of the 'Qualified-Self' Project

The last chapter addressed DM for live looping, my first case study. This chapter introduces a second case study, which deals with tools for creating interactive artistic installations. The chapter is a full manuscript ready to be submitted as:

> `Jeronimo Barbosa`, Sofian Audry, Christopher L. Salter, Marcelo M. Wanderley, Stéphane Huot. *"Designing Behavioral Show-Control for Artistic Responsive Environments: The Case of the 'Qualified-Self' Project"*.

4.1 Abstract

This paper investigates the challenge of designing novel show-control technology for enabling artists to explore rich interactive behaviors within *responsive environments*–tackling long-standing limitations of existing show-control tools. Exploring this direction, we present one case study on a collaborative 2-year research-creation project entitled the 'Qualified-Self'–aimed at exploring

collective physiological signals for artistic responsive environments composition. Our main contributions are: (1) carrying out a formative design study investigating collaborators' context and their needs; (2) designing two new software systems aimed at fulfilling those needs; and (3) performing as evaluation a preliminary real-world application of one of these systems in the design of a professional interactive installation called the 'Other/Self'. We detail and discuss our contributions, hoping to provide general insights for future works on behavioral show-control and creative authoring.

4.2 Introduction

The so-called "digital revolution" has enabled a whole new generation of unique, exciting, radical, and complex computer-based art forms. One of such art forms is *artistic responsive environments*. Responsive environments are immersive physical spaces that can dynamically react to the presence of visitors, based on a wide diversity of input sensors (e.g., biosensors and video cameras) and actuators (e.g., lightning systems, video projection and haptic devices) (Krueger 1977). To artistically craft these environments, long standing show-control technology–traditionally used in the contexts of theater and scenography–are limited and ill-suited: to map these inputs to actuators in a way that is aesthetically meaningful, media artists, composers, and scenographists need to *program custom software*. As such, programming directly impacts on the artistic outcomes, so that high programming capacity yields finer artistic control of the environment. While such reliance on programming is known, little research has focused on understanding the idiosyncrasies and challenges of this peculiar creative context, and on how show-control technology could be better designed to support such idiosyncrasies.

This paper presents a 2-year case study on designing new show-control tools for the composition of artistic responsive environments. This research has been carried within the context of the *'Qualified-Self'* (hereafter, *QS*), a multidisciplinary collaborative research-creation project described as:

> *"The project aims to study the phenomenon of interactional synchrony in a large group of people using biometric/physiological signals such as respiration, heart-rate/heart rate variability (ECG) and skin conductance (GSR). (...) The QS is a unique chance to collaborate with experts in signal processing, music technology, neuroscience and engineering to artistically explore a novel and emerging research field: the use of collectively produced, real time biometric signals to influence the behavior of media in an environment."*

More specifically, we present three key contributions: (1) carrying out a formative design study investigating collaborators' context and their needs; (2) designing two new software prototypes aimed at fulfilling those needs and enabling them to explore rich interactive behaviors within their compositions; and (3) performing a preliminary real-world application of one of these prototypes– the ZenStates–in the design of a professional interactive installation called the 'Other/Self'. We hope these contributions can help future works on behavioral show-control and creative authoring.

4.3 Related work

There is today a plethora of software tools for show control of artistic responsive environments. We can group these tools within four categories: *dataflow*, *textual*, *timeline*, and *augmented timeline* solutions.

Dataflow-based creative coding environments are arguably the most popular solutions, becoming widely adopted over the years. One example is the visual "Max paradigm" (Puckette 2002), found in tools such as Max/MSP, and Puredata, that allows users to build interactive prototypes by visually combining elementary building blocks, each one performing specific real-time operations. Because everything is designed for real-time, managing control flow (e.g., conditional, loops, routines) is often hard with these tools (ibid.). Similar examples include the vvvv, Isadora, and QuartzComposer.

A similar category concerns *textual* creative coding environments and libraries (Noble 2012),

such as Processing, OpenFrameworks, and Cinder. These tools also successfully lower the floor required to get started in programming while allowing for rich control possibilities. Despite their power, these tools still require users to develop general purpose programming skills, such as mastering language and API syntax, and learning abstract control structures.

Alternative approaches to creative coding environments exist that focus more on the control flow. A popular approach is the *timeline* paradigm which is common in video and sound editing softwares. In this paradigm, the user schedules a sequence of cues and contents that happen at specific points in time–usually allowing for different components to overlap, for example using separate "tracks". These systems, which are increasingly prevalent for theatrical and performance control (e.g., QLab, Vezér, ShowCueSystem), are simple and intuitive, and allow for fine-grained control of media content such as audio, video, and lighting. However, because of their rigid temporal structure, they thus make up only partially for the deficiencies of dataflow systems. In particular, they offer no support for real-time interaction (conditionals, loops).

Some recent tools attempt to overcome these limitations by combining characteristic of previous categories, here-grouped as new category called *augmented timelines*. This is the case of the Score, whose power relies on attaching structured programming code to a graphical timeline (Celerier, Desainte-Catherine, and Couturier 2016). Others, such as the Iannix, relies on relational links among different geometrical shapes in 2D and 3D spaces to sequence multimedia cues over time (Jacquemin and Coduys 2014).

Despite their relevance, none of these studies qualitatively probes the context surrounding show control for artistic responsive environments (e.g., What are the stakeholders? Are there recurrent challenges? Are there different stages for composition?). Such qualitative probes could provide researchers with empirical evidence to better design for the particularities of the context.

Finally, these works seem to ignore a rich tradition in the field of new media art, where behaviors do not rely on a pre-defined sequence of events, nor on stateless function-like input-output mappings, but on a much more flexible paradigm of *behavior aesthetics* (Penny 2000). This paradigm is based on the metaphor of an active autonomous or semi-autonomous "agent"

that acts upon the world in realtime. Once designed by artists, these agents are left to evolve on their own. Previous artistic works illustrate how such agent-based composition paradigms–commonly used in robotic art and game design–can open novel aesthetic possibilities that are not possible in traditional new media approaches (Audry 2016; Downie 2005).

4.4 Design study

Motivated by the lack of previous works on the topic, we decided to carry a formative design study investigating the context of responsive environments composition in the media arts. Our goal was to (1) better understand this peculiar practice and challenges faced with current development tools, and (2) support the iterative search for potential solutions.

4.4.1 Requirements elicitation & Survey

In this stage, we first met collaborators to discuss potential requirements for a new ideal development tool oriented towards behavioral show-control. We then used these requirements to review a survey of some popular tools used in new media arts practice. The result is presented on Table 4.1.

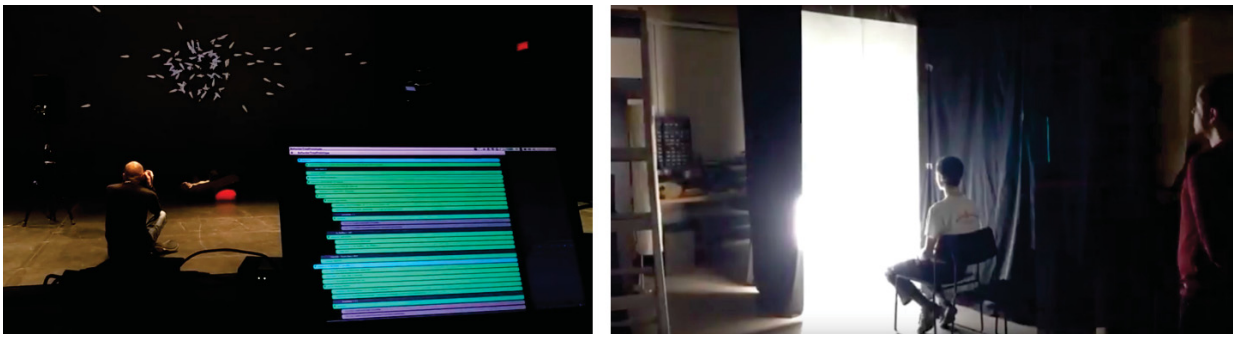


Fig. 4.1 We followed the compositional process of two responsive environment-based pieces created during the QS project. On the right, we show the 'Cepheids', an installation that explores the relationship between the visitor's heartbeats and many virtual agents projected to a large screen. On the left, we show the 'Other/Self', which is detailed in the section *Preliminary 'real world' application*.

Table 4.1 Survey and analysis of some popular tools used for show-control of responsive environments

Name	**Metaphor**	**Interaction**	**Does it support straightforward behavioral control for novices?**	**Experts have incremental fine-grain control of behaviors?**	**Does it support standard technology in new media practice?**
Cue Player	Cuelist; Timeline	Widgets	No	No	Yes
Duration	Timeline	Widgets	No	No	Partial
IanniX	Augmented timeline	Widgets/ Coding	No	Partial	Yes
iMiX16 Pro	Cuelist	Widgets	No	No	Yes
Ossia Score	Augmented timeline	Widgets/ Coding	No	Partial	Yes
Libmapper	Mapping oriented	Widgets	No	No	Partial
Max MSP	Dataflow	Coding	No	Yes	Yes
Openframeworks	Text	Coding	No	Yes	Yes
Processing	Text	Coding	No	Yes	Yes
Pure data	Dataflow	Coding	No	Yes	Yes
Qlab	Timeline; Cuelist	Widgets	Partial	No	Yes
ShowCueSystem	Cuelist	Widgets	No	No	Yes
ShowCueWeb	Cuelist	Hotkeys	No	Yes	Partial
Vezer	Timeline	Widgets	No	No	Yes

4.4.2 Field observation & Interviews

In this stage, we observed the compositional process of two expert artists–both QS collaborators–when composing new responsive environment-based pieces, presented on Figure 4.1. This compositional process involved paper sketching, discussions, and early functional prototyping. Excerpts of this stage are available online[1,2].

We also ran semi-structured interviews with three expert media artists–all QS collaborators–addressing their practice in creating immersive multimedia installations and performances. Questions addressed included:

1. How is your practice? How do you compose and implement these multimedia pieces? What technological tools (hardware and software) do you use? how do you use them?
2. How did you learn this practice and the tools you use?
3. Take your time to think about the following issues. Could you share with us: (a) A big frustration/challenge you've had when composing; (b) What characteristics would have an ideal tool for your practice?

Illustrative scenario

Insights from field observation and interviews are summarized in the scenario of a hypothetical artist called Emma.

Emma is an artist whose expertise is in sound and lighting design. While her composition practice is dynamic and idiosyncratic, it can be roughly organized as presented in Figure 4.2. In this practice, she always tries to maximize the time dealing with aesthetic aspects of her work–stages (a), (b), and (d), e.g., are these lights appropriate for the effect we want to achieve in our audience? what would be the best sounds to match our narrative?. At the same time, she also

[1] 'Other/Self' early tests: `https://youtu.be/PuTLdEo2Vzo`
[2] 'Cepheids' early tests: `https://youtu.be/KwQnkRS5FxE`

tries to keep to "minimum necessary" the time spent dealing with technical aspects–stages (c), (f), and (g), e.g., what is the most efficient way to connect cables inside the stage?.

Emma is preparing a new responsive environments-based piece. For the *narrative scoring* (stage (d)), she aims at creating an interactive room that "behaves" as follows:

1. *Initially, the room is empty.* In this situation, the room is dark (i.e., no lights) and there is a low volume ambient soundscape playing in the background;

2. When a *visitor comes into the room*, a low volume sound starts to play combined with the first. Simultaneously, a 'phantasmagoric' light starts flashing on the other side of the room–at the opposite from where the visitor comes in–inviting the frightened visitor to approach to better see what is going on;

3. As the *visitor moves toward the light*, the room dynamically reacts to his motion. Sound and light brightness increase as the visitor approaches the flashing light. Under the visitor feet, the floor also start to increasingly vibrate. As the general intensity increases, the whole environment becomes increasingly uncomfortable for the visitor;

4. As the *visitor decides to leave the room*, light, vibrations, and sounds slowly shift back to their original state;

5. When the *visitor is gone*, the conditions complete their return to their initial state.

Key challenges

Emma has limited knowledge of computer programming. Ideally, she would team up with technical collaborators in order to carry out technical stages such as (c), (f), and (g). However, this option is often not available (e.g., costs of hiring specialized developers). In this case, she would have to develop these stages by herself and two popular solutions would be available for her: one involving textual (structured imperative) programming, such as Processing (Reas and Fry 2006); another,

visual dataflow programming, such as Max/MSP (Puckette 2002). In this case, key challenges for Emma include:

- **Translating narrative to code:** Investigating how to translate the subjective aesthetic-driven narrative to concrete non-ambiguous programming code. For example, how to code a 'phantasmagoric light'? Related to stages (d) and (f);
- **Language and API knowledge:** Learning proper syntax and logic of imperative or dataflow programming (despite "lowering the floor required to do programming", these tools are still "programming" tools). In addition, it is also necessary to get to know the required APIs. For example, how to produce sound via programming code? How to control the lights? What are the most suitable libraries? Related to stages (c) and (f);
- **General control structure:** Implement abstract concepts to keep track of the current status of the room, and what behavior should be followed (e.g., the code would need to support different states, such as "no one in the room", "visitor enters the room", and so on). Related to stage (f);
- **Input/Output mapping:** When general control structure is done and APIs are mastered, it is necessary to detail what is going to happen inside the room in terms of inputs (e.g., visitors movements) and actuators (e.g., light, sound, vibrations), and how one relates to the other. In certain creative contexts, this mapping challenge is considered critical (Hunt, Wanderley, and Paradis 2003). Related to stage (f);
- **Fine tuning:** Finally, after the first rough sketch is completed, it is necessary to fine-tune the mapping between input and outputs in a way that aesthetically works for the artist. In general, this requires modifying the code on the fly, taking into account the specific characteristics of the physical space where the installation will be. Related to stage (g).

4.4.3 Ideation

Once we had a better understanding of the context, we started exploring possible technical solutions to address the challenges mentioned on the last section. This exploration was mostly composed by quick-and-dirty iterative prototyping, presented in Figure 4.3. As the prototypes became more sophisticated (ranging from paper prototypes to functional prototypes), we started including potential users (i.e., novices and expert artists) in the cycle. These users tested our prototypes in think-aloud protocol sessions, presented in Figure 4.4. In these sessions, potential users provided us with concrete feedback on how to improve our solutions. Raw documentation of some of these prototypes and think-aloud sessions are available online:[3].

This iterative prototyping led to four key technical solutions that could make Emma's challenges easier: (a) visual approach to authoring; (b) adopting straightforward specification models for agent-based authoring; (c) making inputs, actuators, and the most common control structures & variables directly available to users; and (d) encapsulating technical details into high level parameters that can be easily modified by the user.

As an example, a general scaffold on how Emma could approach her interactive room using these solutions is presented in Figure 4.5. In this case, only two states ('empty room', and 'visitor comes in') and two transitions (i.e., '$tracking' and '!$tracking', both blackboard variables represented by the "$" sign) would be enough to efficiently (yet subjectively) communicate Emma's idea. Adding a few tasks (three to the 'empty room', and six to the 'visitor comes in') would add real behaviors to the initial scaffold. Finally, Emma could fine tune these behaviors in practice using high level parameters (i.e., 'intensity', and 'pan') and variables (i.e., '$visitor_x_position').

[3] https://github.com/jeraman/zenstates-paper-vlhcc2018/wiki/Documenting-the-design-process

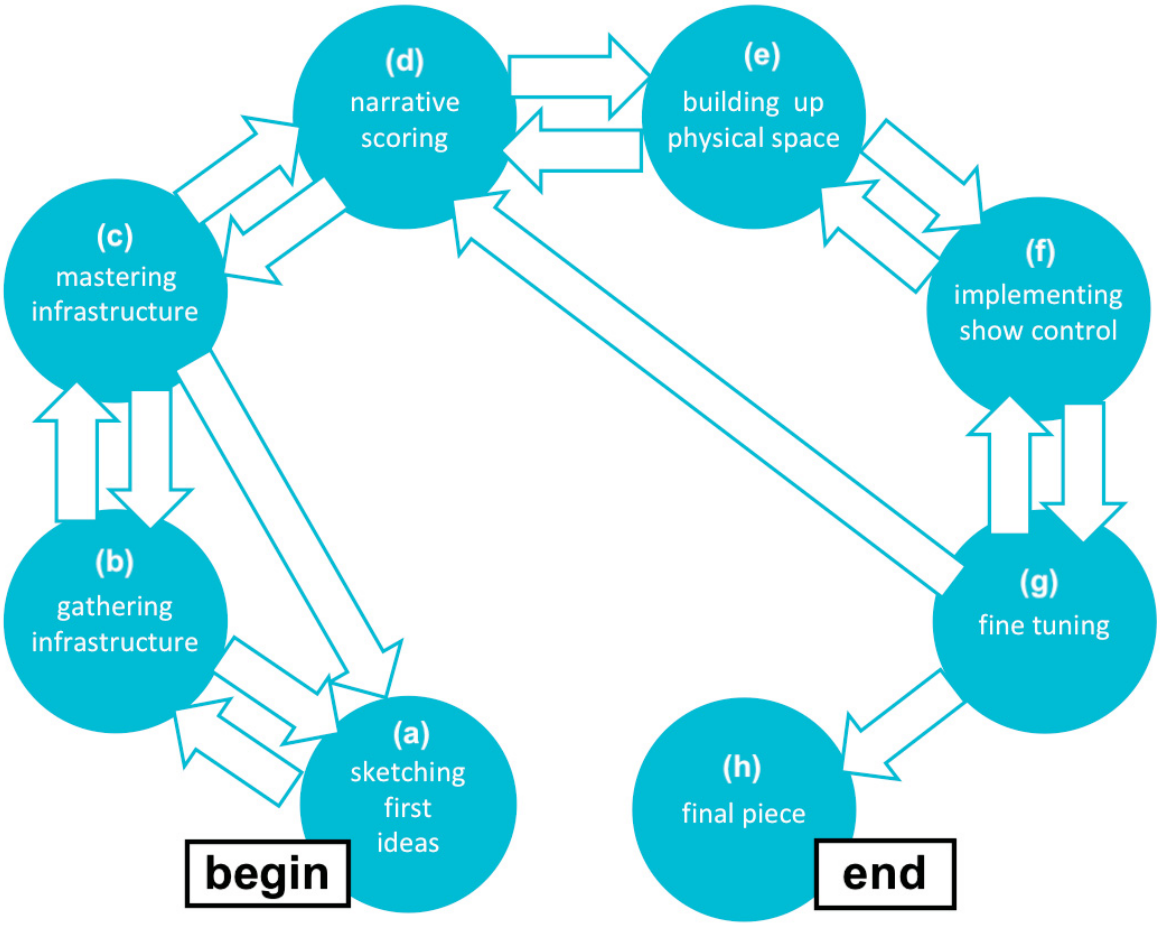


Fig. 4.2 Emma's compositional process can be roughly organized as: (a) *sketching first ideas* about the concept of the piece; (b) *gathering infrastructure*, that is, looking for possible sensor inputs, actuators and materials that might be helpful for the piece; (c) *mastering infrastructure*, hands-on experiments verifying the practical suitability of the gathered infrastructure; (d) *narrative scoring*, where an aesthetic narrative (i.e., a score) is created telling how the mastered infrastructure should behave over the piece; (e) *building up the physical space* aligned with the scored narrative (e.g., size and shape of the room, where to put cables and power outlets, color of the walls); (f) *implementing the show control*, that is, using software to author how the infrastructure should behave to match the intended aesthetic narrative (e.g., how to code a 'phantasmagoric light'?); and (g) *fine-tuning*, adjusting mappings between sensor input and actuators in a aesthetically pleasing way, fitting the physical space profile, and aligned with the scored narrative. The result of this process is (h), the *final piece*.

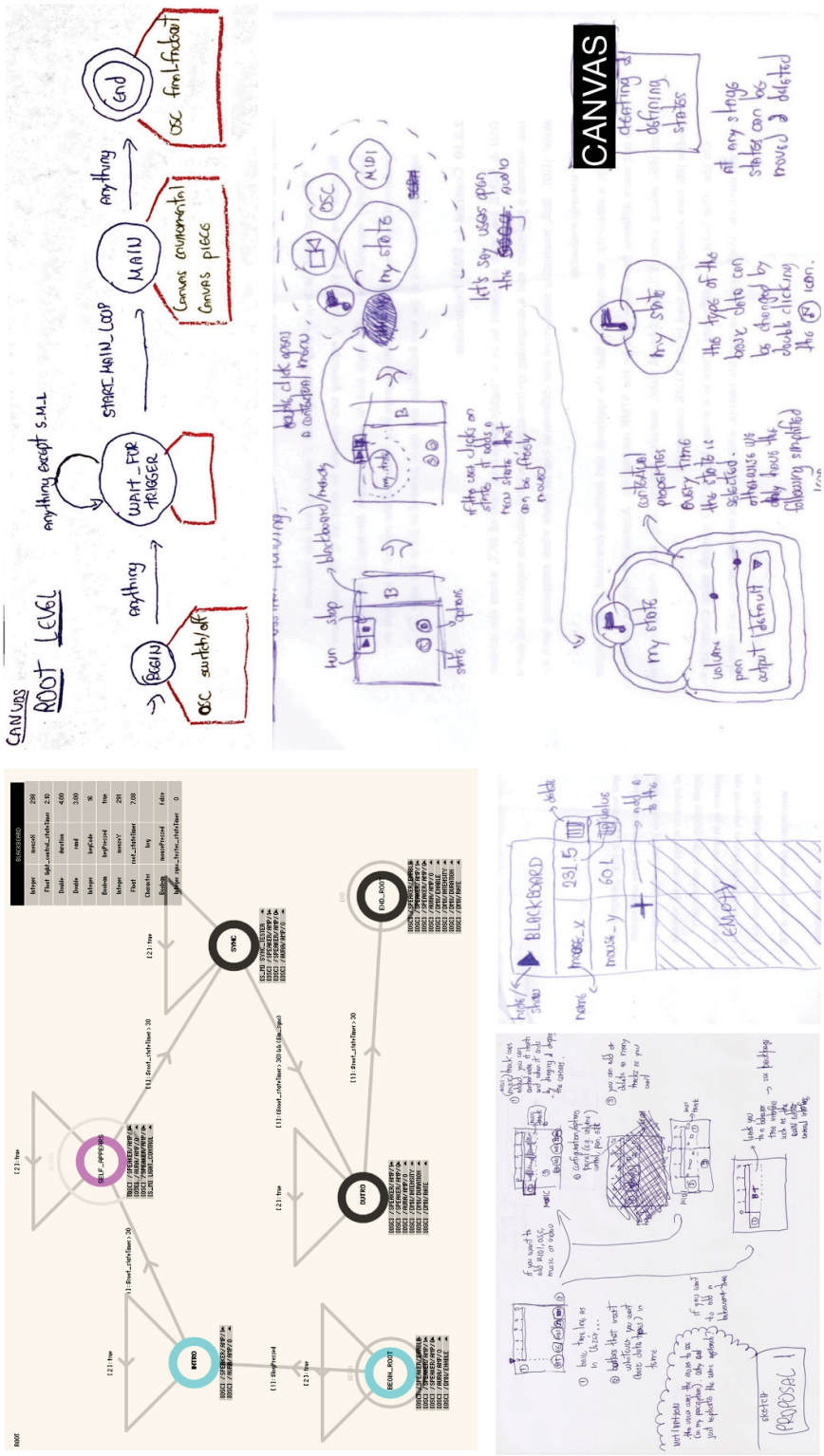


Fig. 4.3 Different prototypes developed during the ideation stage, ranging from early paper prototypes to functional prototypes.

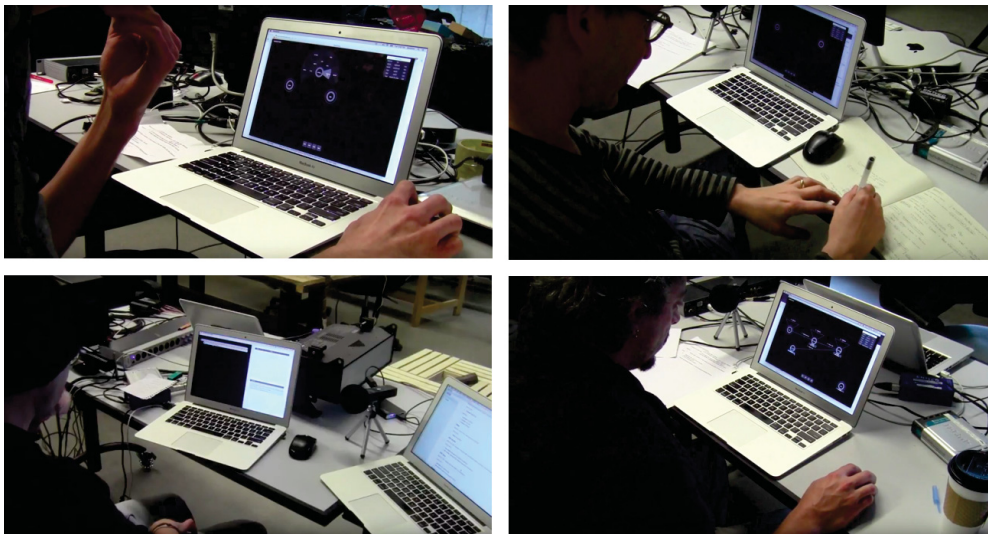


Fig. 4.4 Different think-aloud protocol sessions where our functional prototypes were tested, including both novices and expert artists.

4.5 Designing behavioral show-control

The proposed technical solutions led to the development of two different visual tools for designing behavioral show-control. The first, called *ZenStates*, explores a specification model arguably suited to the reasoning of non-programmers, the *state machines* (Myers, Park, et al. 2008; Pane, Ratanamahatana, and Myers 2001). The second, called *ZenTrees*, explores *behavior tree* as specification model (Colledanchise and Ögren 2018), motivated by limitations of state machines in the context of artificial intelligence in video games (Colledanchise and Ögren 2017; Cutumisu and Szafron 2009).

Both tools have been prototyped in Java, using Processing[4] as library for the graphics. Their source code is available online[5]. These tools are introduced in the next subsections.

[4] `http://processing.org/`
[5] `https://github.com/qualified-self/cue-control`

Fig. 4.5 ZenStates allows users to prototype behavioral responsive environments by directly manipulating *States* connected together via logical *Transitions*. *Tasks* add concrete actions to states, and can be fine-tuned via high level parameters (e.g., 'intensity' and 'pan')*Blackboard* global variables (prefixed with the '$' sign) can enrich tasks and transitions.

4.5.1 Introducing ZenStates

ZenStates is a visual-based behavioral show-control tool centered on *State machines*. Its related technical research and evaluation have been fully reported elsewhere (Barbosa, Wanderley, and Huot 2018). Here we summarize key elements.

In ZenStates, a *State machine* is defined as a set of abstract *States* to which users might add *Tasks* that describe (in terms of *high level parameters*) what is going to happen when that particular state is being executed. These states are connected to one another via a set of logical *Transitions*. Execution always starts at the "Begin" state, following these transitions as they happen until the end. At any moment, inside a Task or a Transition, users can use *Blackboard* variables to enrich behaviors (e.g., use the mouse x position to control the volume, or to trigger certain transitions).

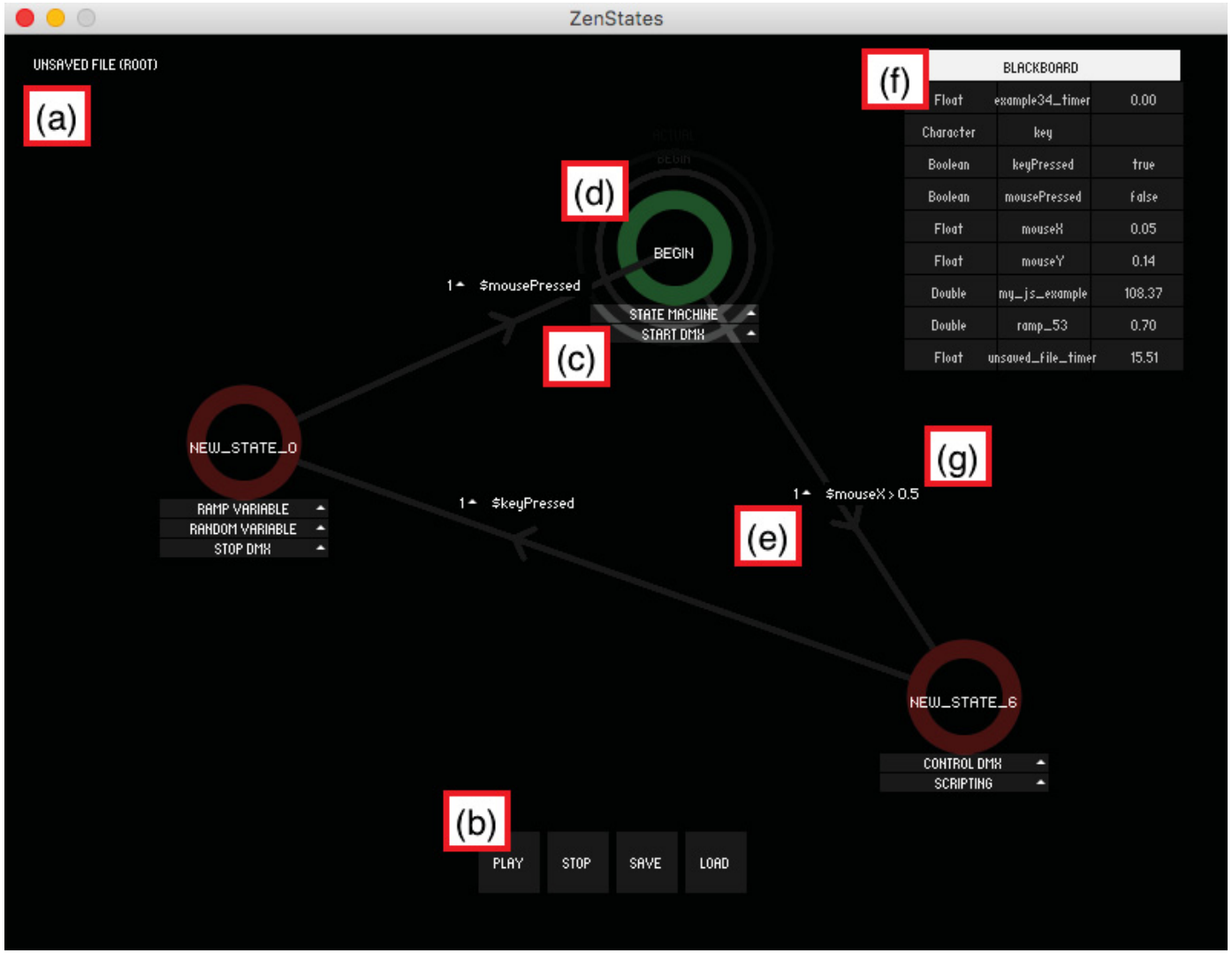


Fig. 4.6 ZenStates' interface: (a) the state machine name; (b) control buttons; (c) tasks associated to a state; (d) the initial state (in this case, also the one which is currently running, represented in green); (e) a transition priority; (f) the blackboard; and (g) a transition.

All these abstract elements (namely the state machine, states, tasks, transitions, and the blackboard) are reified–according to the principle of reification introduced in (Beaudouin-Lafon and Mackay 2000)–inside the interface to graphical objects that can be directly manipulated by the user (Shneiderman et al. 2010). These visual representations become the core of the user interaction and also allow users to infer the current state of the system at any time during execution: the current running state is painted green; states already executed are red; inactive states are gray (see Fig. 4.6).

When combined, these features result in a powerful hierarchical model of state machines. This model provides *organization* (i.e., state machines are potentially easier to understand), *modularity* (i.e., subparts of a system could be independently designed and later merged into a larger structure), and *expressiveness* (broadcast communication allows enhanced description of complex behaviors) when compared to the traditional states machine approach.

4.5.2 Introducing ZenTrees

ZenTrees is a visual-based behavioral show-control tool centered on *Behavior trees*–a goal-oriented programming paradigm commonly used to design agent behaviors in robotics and video games (Colledanchise and Ögren 2018). ZenTrees combines this behavior trees paradigm with sequential composition tools, as provided in timeline-based cue-control systems such as the QLab, with some interaction elements of textual creative coding environments. Therefore, by using hot keys, textual input, and autocompletion, users can add control mechanisms typically found in digital art installations such as sequences, selections, priorities, parallelism, and randomization.

These control mechanisms are integrated in a *Tree structure* that provides a hierarchical representation of the work's dynamics–as shown in Fig. 4.7. This tree structure is composed by a set of *Nodes*, each of which corresponds to a certain task to be accomplished. When a node is called, it attempts to accomplish the task and returns one of three results: *success* (if the attempt worked), *failure* (if it failed, typically because a condition was not verified), or *running* (if it still has not finished doing the task). These results are represented in real-time in the graphical interface,

providing realtime visual feedback on the current state of the system.

ZenTrees provide two kinds of nodes: *Leaf* and *Composite*. Leaf nodes correspond to simple atomic tasks such as sending an Open Sound Control (OSC)[6] or Digital MultipleX (DMX)[7] message, setting a blackboard variable, playing a sound file, introducing a delay, etc. Composite nodes define specific control mechanisms over their children nodes, allowing the user to combine nodes to accomplish more complex tasks (e.g., initializing some variables, then playing a sound file while adjusting the volume according to some sensor input values). For example, the *sequence* composite node runs through all of its children until one of then fails; the *selector* node stops as soon as one of its children succeeds; and the *parallel* node runs all of its children at the same time. These composite nodes can be expanded to reveal their children, or collapsed to hide the details.

To increase control and flexibility, ZenTrees also provides a set of node attachments called *Decorators* which refine the behavior of a node. For example, the *while* decorator will continuously call the node it is attached to as long as a condition is checked; whereas the *randomize* decorator will call the children of a composite node in random order. Finally, as in ZenStates, ZenTrees also provides a *Blackboard* of global variables that can be used inside the nodes.

4.6 Evaluation

As *formative evaluation*, both ZenStates and ZenTrees have been constantly tested, and iteratively redesigned over the course of the QS project–by the means of its user-centered design process (c.f. section 'Formative design study').

Moreover, as *summative evaluation*, we have tried to apply one of our tools, the ZenStates, in real life for the show-control of a professional artistic installation. Our goal here has been to assess the potential of ZenStates as a tool that fulfills collaborators' needs in artistic responsive environments show-control.

[6] `http://opensoundcontrol.org/introduction-osc`
[7] `https://en.wikipedia.org/wiki/DMX512`

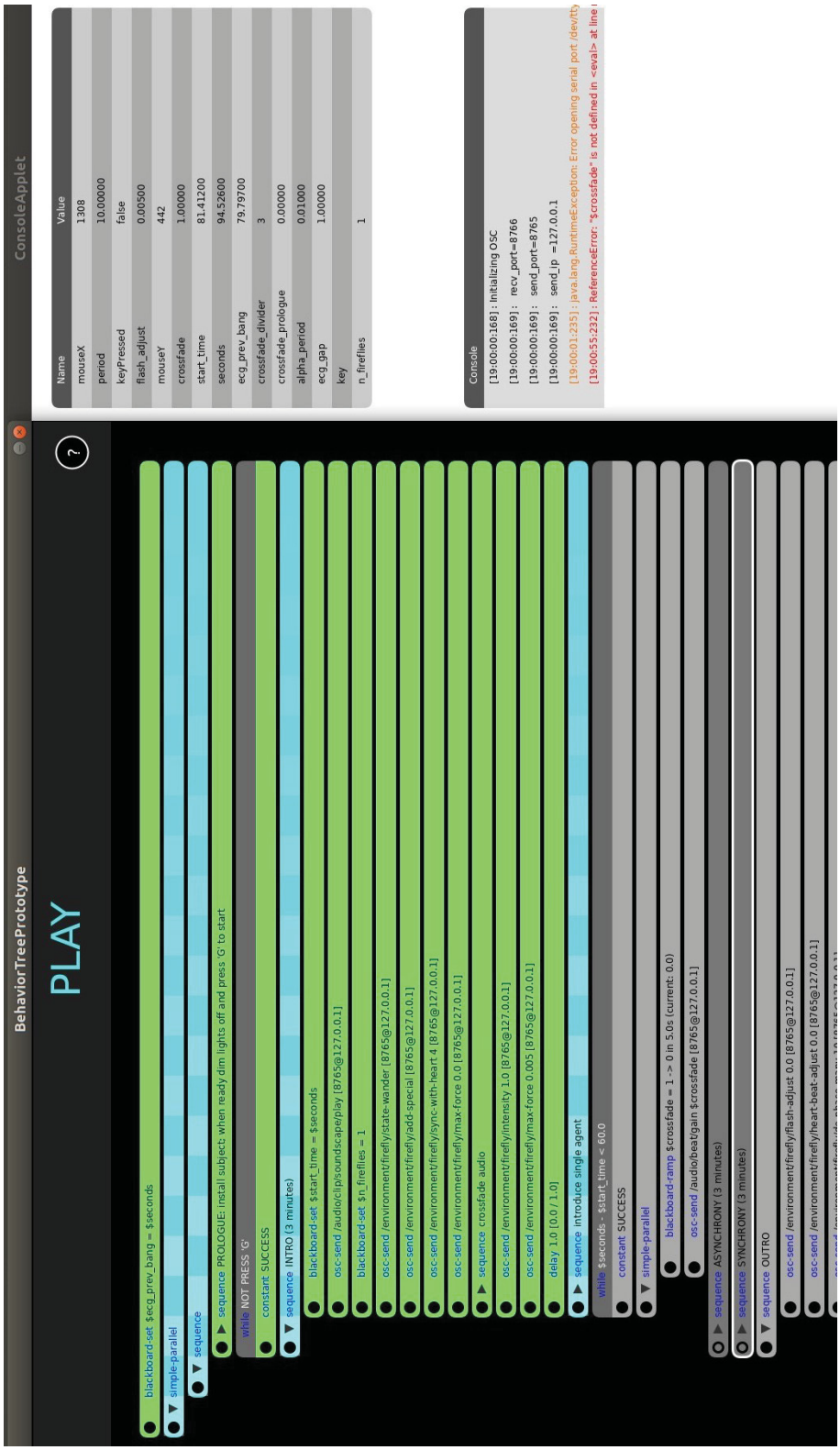


Fig. 4.7 A view of ZenTrees' interface running "Cepheids". ZenTrees allows users to prototype behavioral responsive environments by visually interacting with *Nodes* modified using *Decorators*. *Leaf* nodes implement concrete actions which are controlled using *Composite* nodes and *Decorators*. *Blackboard* global variables (prefixed with the '$' sign) are used to store input values and persistent properties.

Fig. 4.8 Final version of the "Other/Self" piece, by the QS collaborators Chris Salter, TeZ and Luis Rodil Fernández, premiered in the STRP Biënnale that took place in Eindhoven, the Netherlands, in March 2017. On the left image (credits: Hanneke Wetzer), the entry of the responsive environment, where two visitors at time were invited to wear customized suits. The right image (credits: xmodal) shows the interior of the responsive environment, where visitors had to sit one in front of the other and the piece took place.

4.6.1 Preliminary "real world" application

ZenStates has been used in the making of a new interactive installation called "Other/Self"–Figure 4.8–by the QS collaborators Chris Salter, TeZ and Luis Rodil Fernández. As described by the artists:

> *"Other/Self is a performative installation exploring the tension between a vision of the self as 'quantified' versus a 'qualified' self that emerges through the embodied and sensorial presence of the other. The installation consists of an immersive sound, light and haptic experience directly modulated by one of our most intimate and felt bodily experiences: our heartbeat. Two visitors at a time enter a brightly lit enclosed room and put on a set of wearable sensors and actuators. Sitting side by side but facing in opposite directions, the visitors are suddenly plunged into total darkness. Very gradually, each individual begins to barely feel the heartbeat of the other, transferred and transformed*

> *into various luminous, acoustic and haptic patterns at varying levels of intensity. As the visitors' breathing and heartbeats come into synchrony, the sensorial and corporeal encounter begins to change. Rapid, abrupt changes of light, sound and vibration build to an uncanny climax in which the border between self and other, self and environment dissolves."*

The "Other/Self" has premiered in the STRP Biënnale 2017[8] that took place in Eindhoven, the Netherlands, in March 2017.

In technical terms, as media outputs, the installation is based on two DMX controlled spotlights, two DMX controlled stroboscopes, six audio tracks in stereo sound, two aurasound tactile sound actuators, and two custom portable hardware actuators. As sensor input, the piece uses a photoplethysmogram (PPG) heartbeat sensor. This setup was integrated into a media server developed in Max/MSP, that could be entirely controlled via OSC messages. This media server was then properly integrated to ZenStates' tasks.

During the making of the piece, artists, the authors, and other collaborators worked together. The initial plan was to develop the installation exclusively using ZenStates for the logical control. However, as the development advanced, software issues kept appearing, slowing down the progress. Because of the timely schedule and the amount of time required to fix these bugs, we ended coding the logic control in Max/MSP. In a later moment, after the premiere, we were able to fix the bugs and to implement the whole control with ZenStates.

Despite the fact the integration was only partial, this preliminary experience provides us with several insights–derived from observations–about the usage of ZenStates in a real scenario. In the next subsection, we will address three of these insights, focusing on how ZenStates improved the general development of the "Other/Self".

[8] `https://strp.nl/en/`

From ideas to implementation

The first observation concerns how ZenStates made easier the transition between the subjective narrative developed by the artists to the direct prototyping of the installation.

Before starting the development, the artists focused on creating a narrative describing both behavior and effects they wanted to achieve over time–stage (d) from Figure 4.2. This narrative was organized into a flexible timeline describing what was going to happen in each moment of the work–as in an open-ended score. This score was written in a spreadsheet presented in Figure 4.9.

The transition between this subjective score to a concrete responsive environment was direct and straightforward with ZenStates–stage (f) on Figure 4.2. Basically, the whole installation became a state machine where each movement was a state. Nested state machines were added to these states, allowing each movement to be modularized and designed separately. Inside each nested state, sequences were also transformed to states. Finally, atomic media behaviors were translated to their corresponding tasks. Throughout these steps, blackboard tasks were added to allow the specification of the behaviors intended by the artists.

The resulting ZenStates implementation for the movement shown in Figure 4.9 can be found in Figure 4.10. As a contrast, the same movement coded in the same level of details (i.e., only the logic control) in Max/MSP is presented in Figure 4.11.

Fine-tuning aesthetics

Implementing the general logic control, with their states and transitions, is only the first step towards implementing the piece. After this first step, a significant amount of work has to be dedicated to fine tune the installation in a way that it fulfills the aesthetic aims of the artists–stage (g) on Figure 4.2.

Basically, this process involves numerous repetitions of the same movement by playing it several times. Repetitions are experienced by the artists, generating concrete feedback on how to modify parameters in order to match the intended effect. These changes often requires modifying the

	A	B	C	D	E	F
1	**MOVEMENT 1**		**setup**	**sequence 1.1**	**sequence 1.2**	**end**
2		LIGHT (PAR)		On - 0 ->	10 %	out
3		LIGHT (STROBE)				Flash (2-5 second) @ 20%
4		SENSORS	assistant registers pulse and then pushes button to initiate sequence	Pulse (cooked R-R)	Pulse	
5		BODY (HAPTICS):				
6	**PERSON 1**	CHAIR:	led in bench (OFF WHEN PULSE REGISTERED)	Aura (Pulse) - barely perceivaable	Aura (Pulse) perceivable	out
7		SOUND:	n/a	(aura follows heart)		Transition drone begins
8						
9		LIGHT (PAR)		On - 0 ->	10 %	out
10		LIGHT (STROBE)				Flash (2-5 second) @ 20%
11		SENSOR S	assistant registers pulse and then pushes button to initiate sequence	Pulse (cooked R-R)	Pulse (cooked)	
12		BODY (HAPTICS):				
13	**PERSON 2**	CHAIR:	led in bench (OFF WHEN PULSE REGISTERED)	Aura (Pulse) - barely	Aura (Pulse)	

Fig. 4.9 Part of the spreadsheet score created by the artists. The piece is divided in four movements, each one composed of smaller sequences (represented by columns in the image) that together form the narrative. In each sequence, the artists had a specific vision on how the media elements should behave over time (represented by rows). This image corresponds to the first movement.

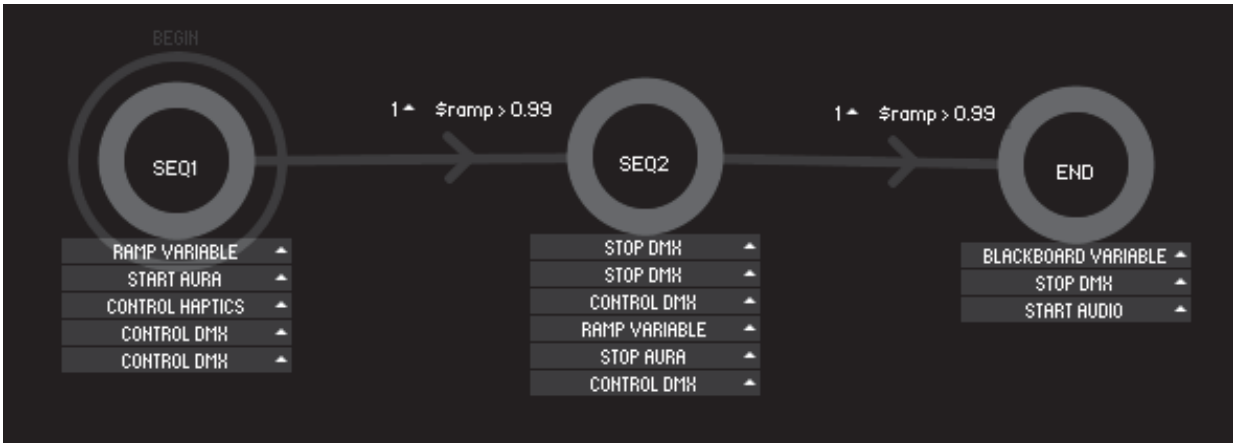


Fig. 4.10 The first movement already described in Figure 4.9 implemented in ZenStates.

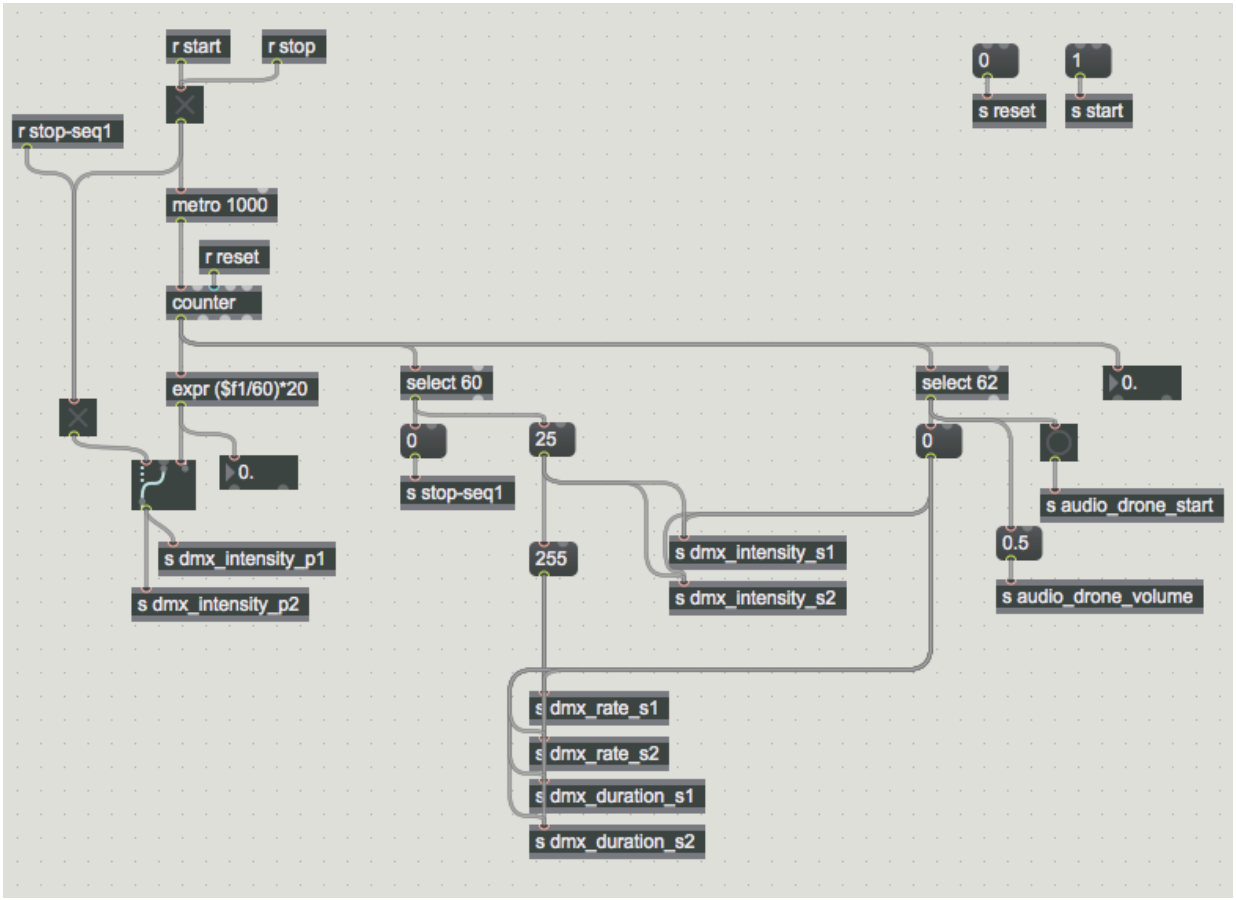


Fig. 4.11 The first movement now implemented in a dataflow language (Max/MSP) by an expert user.

implementation on the fly, inside the physical environment where the installation is installed. Examples of changes like these include: *"in movement 3, light strobes should not ramp from 13 to 50, they should be instead from 0 to 200"*; *"heartbeat sound should follow the same range of values"*; or yet *"movement 1 is too long, we need to make it shorter and more intense"*.

ZenStates makes this process easier and faster due to its enhanced organization. Fine-tuned changes can be made by altering tasks' high level parameters to match the intended behavior. Conversely, tools such as Max/MSP delegate this organization to the user: if the user is not a skilled and experienced programmer and the code is not well organized, changes could become nightmares. Furthermore, the live development also helps by allowing the artists to experience changes as they happen.

Encapsulation of technical details

ZenStates provided a clear separation of narrative versus the low-level technicalities that were not directly relevant to artists when composing. This is achieved by encapsulating these low-level technicalities inside the task and inside the media server. By reducing the technical effort required to explore responsive environments (stage (f) on Figure 4.2), we enable artists to keep their focus on the narrative-level instead (stage (d)). Additionally, by articulating this separation, we incorporated modularity and reuse principles to the practice of non-expert programmers, resulting in improved organization and clarity throughout the implementation.

4.7 Discussion

Despite still preliminary and non-conclusive, we believe that our real-world application shows some initial evidence that, as a behavioral show-control tool, ZenStates–and potentially also ZenTrees–proposes a good balance between *"low thresholds"*–describing if novices can easily get started with the system–and *"high ceilings"*–describing if the system provides the advanced features required by experts (Shneiderman 2007; Wessel and Wright 2002). Finding a balance between "low thresholds" and "high ceilings" is difficult: Focusing too much on the former can yield limited ceilings; Focusing

too much on the latter can yield high-thresholds (Myers, Hudson, and Pausch 2000).

On the one hand, results suggest that ZenStates is more low-thresholded than some its alternatives (i.e., both *dataflow* and *textual* creative coding environments). In particular, discussions in *'From ideas to implementation'*, *'Fine-tuning aesthetics'*, and *'Encapsulation of technical details'* exemplify how ZenStates could make the interactive behavior authoring easier within artistic responsive environments.

Such initial evidence is corroborated by the later studies performed outside the scope of the QS project (Barbosa, Wanderley, and Huot 2018). Quantitative evidence suggest that ZenStates better conveys specifications of interactive environments than its two alternatives. ZenStates model was on average 1.4 times faster than dataflow model, and 1.8 times faster than the structured model in terms of decision time. Such results were achieved despite the fact participants had no previous knowledge about the ZenStates system, whereas they were familiar with the other models. ZenStates' superiority in conveying specifications is also confirmed by subjective user preferences, since 8 out of 12 participants assessed ZenStates as the easiest alternative to understand.

On the other hand, while their ceilings are not as high as some of their alternatives–ZenStates is context-dependent and cannot be extended to other contexts without further coding; *dataflow* and *textual* creative coding environments are flexible programming languages with larger variety and finer grain control of outputs–they are high enough to provide sophisticated creative outputs. One example is how collaborator's subjective score has been directly and straightforwardly transformed into a concrete responsive environment with ZenStates (Figures 4.9, 4.10, and 4.11). This finding seems aligned with evidence from a previous work (ibid.), where 90 unique distinct audiovisual scenarios were authored within ZenStates using a constrained set of inputs and outputs.

These results, we argue, point towards the potential of our prototypes as behavioral show-control tools with low thresholds and relatively high ceillings, aligned with our collaborators' needs. As pointed by (Myers, Hudson, and Pausch 2000), *"lessons of past tools indicate that in practice high-threshold systems have not been adopted because their intended audience never makes it past the initial threshold to productivity, so tools must provide an easy entry and smooth path to*

increased power". Considering this quote, our approaches might be a step in the right direction.

4.7.1 Contrasting ZenStates and ZenTrees

Finally, we point out that while ZenStates and ZenTrees share several common characteristics (e.g., visual-based tools, the blackboard, encapsulation), they also seem to significantly differ in terms of thresholds and ceilings. Our experience suggest that ZenStates seems more straightforward for novices without training, while ZenTrees seems sometimes counter-intuitive, as the conceptual model of behavior trees seems more challenging to grasp than the one of state machines. On the other hand, ZenTrees seems to enable more fine-grained agents-like behaviors, whereas achieving the same level of granularity in ZenStates may require more effort. Combining advantages and drawbacks of both approaches into one integrated solution requires further empirical investigation.

4.8 Limitations & Future work

Some limitations of ZenStates as specification model has already been pointed in previous works (Barbosa, Wanderley, and Huot 2018). Examples include the blackboard, the physical environment alignment, and interface usability and stability–all of which apply equally to ZenTrees. Furthermore, when compared against existing solutions such as traditional timelines, we point out that both prototypes makes manipulation of rigid time structures more challenging. Supporting smooth transitions between different states (i.e., by attaching time duration to state transitions, where individual state configurations may be automatically interpolated while transition happens) is another interesting direction for improvement.

Finally, we stress that while our prototypes support important stages of responsive environments composition practice–namely, stages (f) and (g) from Figure 4.2–, other relevant stages–namely (a), (b), (c), (d), and (e)–remains largely unexplored. Similarly, existing show-control solutions also seem to give little attention to these stages. Investigating how the show-control environment could support these stages is a promising direction for future works.

4.9 Conclusion

We have presented three contributions from the collaborative research-creation project the Qualified-Self. These contributions were: (1) carrying out a formative design study investigating collaborators' context and their needs; (2) designing two new software technologies aimed at fulfilling those needs; and (3) performing as evaluation a preliminary real-world application of the ZenStates in the design of a professional interactive installation called the 'Other/Self'. On a higher-level, we have investigated the challenge of design of novel show-control technology that overcome long-standing limitations of show-control tools, enabling collaborators to express rich interactive behaviors in show-control.

4.10 Acknowledgment

We would like to thank Ivan Franco and all collaborators involved with the QS project, for their time and support.

Chapter 5

ZenStates: Easy-to-Understand Yet Expressive Specifications for Creative Interactive Environments

The previous chapter started introducing my second case study: DM in tools for creating interactive artistic installations. Here, I continue investigating the topic by focusing on the ZenStates and its evaluation instead. This chapter has been previously published as:

> Jeronimo Barbosa, Marcelo M. Wanderley, and Stéphane Huot. *"ZenStates: Easy-to-Understand Yet Expressive Specifications for Creative Interactive Environments."* In 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Lisbon, Portugal, 167–175. © 2018 IEEE[1]. https://doi.org/10.1109/VLHCC.2018.8506491

[1]In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of McGill University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

5.1 Abstract

Much progress has been made on interactive behavior development tools for expert programmers. However, little effort has been made in investigating how these tools support creative communities who typically struggle with technical development. This is the case, for instance, of media artists and composers working with interactive environments. To address this problem, we introduce ZenStates: a new specification model for creative interactive environments that combines Hierarchical Finite-States Machines, expressions, off-the-shelf components called *Tasks*, and a global communication system called the *Blackboard*. Our evaluation is three-folded: (a) implementing our model in a direct manipulation-based software interface; (b) probing ZenStates' expressive power through 90 exploratory scenarios; and (c) performing a user study to investigate the understandability of ZenStates' model. Results support ZenStates viability, its expressivity, and suggest that ZenStates is easier to understand–in terms of decision time and decision accuracy–compared to two popular alternatives.

5.2 Introduction

Over the last years, several advanced programming tools have been proposed to support the development of rich interactive behaviors: The HsmTk (Blanch and Beaudouin-Lafon 2006) and SwingStates (Appert and Beaudouin-Lafon 2008) toolkits replace the low-level event-based paradigm with finite-states machines; ConstraintJS (Oney, Myers, and Brandt 2012) introduces constraints declaration and their implicit maintenance as a way to describe interactive behaviors; InterState (Oney, Myers, and Brandt 2014) and Gneiss (Chang and Myers 2014) are dedicated programming languages and environments. These advances are primarily focused on programmers and are important because:

- They can make interactive behavior *easier to understand*–sometimes even by non-experts programmers. This is the case, for instance, of the SwingStates toolkit (Appert and Beaudouin-Lafon 2008). SwingStates was successfully used by graduate-level HCI students to implement

advanced interaction techniques despite of their limited training, when other students with similar skills were unsuccessful in implementing the same techniques with standard toolkits. In addition, better understanding of the implementation of interactive behaviors can make it easier to reuse and modify (Blanch and Beaudouin-Lafon 2006);

- These tools do not sacrifice *expressiveness* to make programing faster or understanding easier. Examples such as (Oney, Myers, and Brandt 2012) and (Chang and Myers 2014) illustrate how such tools can potentially implement a wide variety of complex interactive behaviors, which would have been more costly in existing alternatives.

Despite these advantages, little effort has been made to investigate how these advances could support end-user programmers (EUP) (Ko, Myers, Rosson, et al. 2011; Paternò and Wulf 2017) from creative communities who typically struggle with coding such as artists and designers (Bailey, Konstan, and Carlis 2001; Myers, Park, et al. 2008). Making these behaviors *easier to understand* could make them easier to learn, reuse, and extend. At the same time, addressing *expressiveness* (i.e., going beyond standard and well-defined behaviors) could improve the diversity of creative outcomes, helping in the exploration of alternatives. All in all, these characteristics have the potential to foster one's creative process (Shneiderman 2007).

One example of such community is composers and media artists working with *creative interactive environments* (Krueger 1977). These environments are immersive pieces that react to the presence of visitors, based on a wide diversity of input sensors (eg. video cameras) and actuators (eg. sound, lightning systems, video projection, haptic devices). Creating easy to understand and expressive development tools for these environments is relevant and challenging for two reasons. First, interactive environments' setups are often more complex than standard interactive systems (such as desktop or mobile computers), as they (i) need to deal with multiple advanced input/output devices and multiple users that both can connect/disconnect/appear/disappear dynamically; (ii) need to be dynamic, flexible and robust. Therefore, by tackling more complex setups, some of its unique features could potentially transfer to more standard setups (e.g. desktop). Secondly,

because programming directly impacts on the artistic outcomes, increasing programming abilities and possibilities can likely yield finer artistic results, by reducing the gap between artistic and technical knowledge.

These are the context and the challenge we address in this paper. Here, we investigate innovations brought by powerful development tools for expert programmers (Chang and Myers 2014; Harel 1987; Oney, Myers, and Brandt 2014), aiming at bringing them to the context of creative interactive environments. The result is ZenStates: an easy to understand yet expressive specification model that allows creative EUPs to explore these environments using Hierarchical Finite-States Machines, expressions, off-the-shelf components called tasks, and a global communication system called the blackboard. This model has been implemented in a visual direct manipulation-based interface that validates our model (Fig. 5.1). Finally, as an evaluation, we also report: (a) 90 exploratory scenarios to probe ZenStates' expressive power; and (b) a user study comparing ZenStates' ease of understanding against the specification model used by two popular interactive environments development tools: Puredata and Processing.

5.3 Related work

One of the most basic approaches for interactive environments development is trigger-action programming (Ur et al. 2014). This approach–used, for instance, in the context of smart homes–describes simple conditional "if-then-else" behaviors that can be easily customized by users. Despite being intuitive and effective, this approach limits interactive environments to simple one-level behaviors. For instance, any example involving temporal actions cannot be implemented.

Other approaches can be found within the context of creative communities. This is the case of the visual "Max paradigm" (Puckette 2002), found in musical tools such as Max/MSP, and Puredata. These dataflow-inspired software environments have become widely adopted over the years by creative EUPs. They allow users to build multimedia prototypes by visually combining elementary building blocks, each performing a specific real-time operation. However, because everything is designed for real-time, little support is provided for managing the flow of control of

Fig. 5.1 ZenStates allows users to quickly prototype interactive environments by directly manipulating *States* connected together via logical *Transitions*. *Tasks* add concrete actions to states and can be fine-tuned via high-level parameters (e.g. 'intensity' and 'pan'). *Blackboard* global variables (prefixed with the '$' sign) can enrich tasks and transitions. © 2018 IEEE.

application (e.g. conditionals, loops, routines) (Puckette 2002).

Another approach is the case of textual programming environments and libraries (Noble 2012), such as Processing, openFrameworks, and Cinder. However, although these tools lower the floor required to get started in programming, they still require users to deal with abstract concepts whose meaning is not often transparent, especially to novices (Victor 2012).

To overcome this understandability issue, direct manipulation (Shneiderman 1983) has been applied in this context to make programming less abstract and therefore more tangible to creative users. Examples include Sketch-N-Sketch (Hempel and Chugh 2016)–where direct manipulation is combined with textual programming language–and Para (Jacobs, Gogia, et al. 2017)–direct manipulation combined with constraint-based models. While the strategy is helpful, the solution is kept on the interface level (i.e., that could potentially be applied to every specification model), whereas the specificion model itself remains unexplored. Furthermore, both examples are limited to static artistic results (i.e., not capable of reacting to inputs over time).

5.3.1 Non-programmers expressing interactive behaviors

Given our interest in exploring more accessible specification models, we need to understand first how non-programmers deal with expressing interactive behaviors. A few works have dealt with this topic.

In (Pane, Ratanamahatana, and Myers 2001) for instance, two studies focused on understanding how non-programmers express themselves in solving programming-related problems. In one study, 10-11 years old children were asked to describe how they would instruct a computer to behave in interactive scenarios of the Pac-man game. In another, university-level students were asked to do the same in the context of financial & business analytical problems. Results suggest that participants tended to use an event-based style (e.g., if then else) to solve their problems, with little attention to the global flow of the control (typical in structured imperative programming languages, for example). This approach to problem-solving arguably has similarities to the approach that state machines have in problem-solving.

Some work also presented practices and challenges faced by professional multimedia designers in designing and exploring rich interactive behaviors derived from series of interviews and a survey (Bailey, Konstan, and Carlis 2001). The results suggest that in the case of designing interactive behaviors: a) current tools do not seem to fulfill the needs of designers, especially in the early stages of the design process (i.e., prototyping); b) texts, sketches, and several visual "arrow-centered" techniques (e.g. mind map, flowcharts, storyboards) are among the most used tools to communicate ideas; and c) designers prefer to focus on content rather than "spending time" on programming or learning new tools. These findings resulted in a new system, DEMAIS, which empowers designers to communicate and do lo-fi sketch-based prototypes of animations via interactive storyboards.

Similar findings were reported by (Myers, Park, et al. 2008). In a survey with 259 UI designers, the authors found that a significant part of participants considered that prototyping interactive behaviors was harder than prototyping appearance (i.e., the "look and feel"). Participants reported struggling with communicating interactive behaviors to developers–although this communication was necessary in most of the projects–when compared to communicating appearance. In addition, participants reported necessity for iteration and exploration in defining these behaviors, which generally involved dealing with changing states. The authors conclude by reporting the need for new tools that would allow quicker and easier communication, design and prototyping of interactive behaviors.

These works suggest that state machines could be suited to the development of interactive behavior by non-programmers, and, we believe, by creative end-user programmers who struggle with technical development.

5.3.2 Experts programming interactive behaviors

Several tools have been developed to support expert programming of interactive behaviors (Cuenca et al. 2015). Examples include Gneiss (Chang and Myers 2014), ICon (Dragicevic and Fekete 2004), and OpenInterface (Lawson et al. 2009). Among these, a significant number of tools use

state machines to specify such interactive behaviors. A complete survey is beyond the scope of this research. Here, we focus on works we consider more closely related to our research.

Perhaps one of the most well-known example is (Harel 1987), which introduced StateCharts: a simple yet powerful visual-based formalism for specifying reactive systems using enriched hierarchical state machines. StateCharts' formalism was later implemented and supported via a software tool called StateMate (Harel et al. 1990), and is still used (more than 30 years later) as part of IBM Rational Rhapsody systems for software engineers[2].

Examples are also notable in the context of GUIs, for instance: HsmTk (Blanch and Beaudouin-Lafon 2006), a C++ toolkit that incorporates state machines to the context of rich interaction with Scalable Vector Graphics (SVG); SwingStates (Appert and Beaudouin-Lafon 2008), that extends the Java's Swing toolkit with state machines; and FlowStates (Appert, Huot, et al. 2009), a prototyping toolkit for advanced interaction that combines state machines for describing the discrete aspects of interaction, and data-flow for its continuous parts.

More recently, research has been done on how to adapt state machines to the context of web applications. ConstraintJS (Oney, Myers, and Brandt 2012) proposed an enhanced model of state machines that could be used to control temporal constraints, affording more straightforward web development. This idea is further developed by the authors in (Oney, Myers, and Brandt 2014), resulting in InterState, a new full programming language and environment that supports live coding, editing and debugging.

These works introduce advances that make interactive behaviors programming faster and easier to understand. However, as a drawback, these tools are hardly accessible for creative EUPs who would still need to develop strong programming skills before benefiting from them. To address this problem, we have built upon these works, aiming at making them accessible to creative EUPs. The result, a new specification model called *ZenStates*, is introduced in the next section.

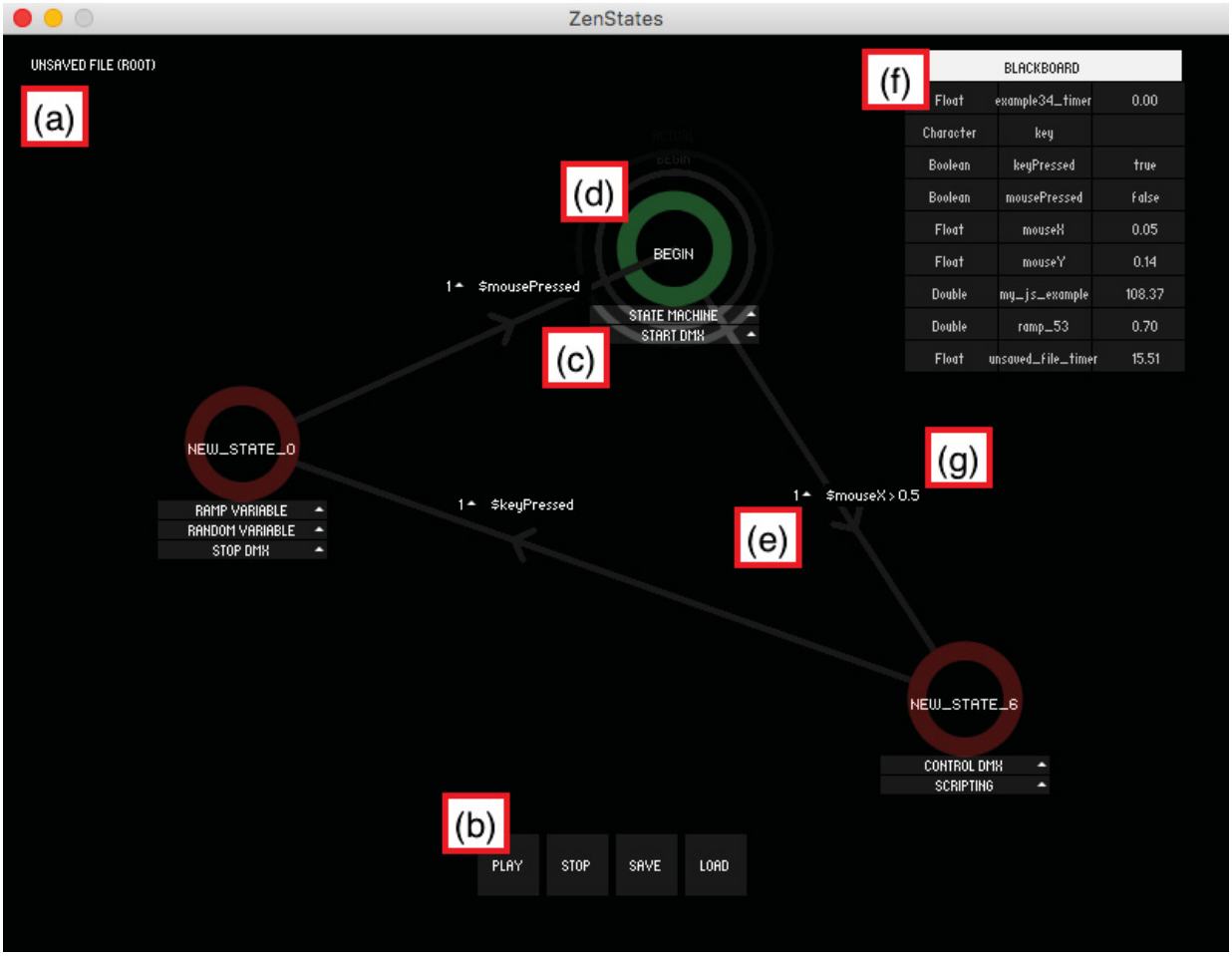


Fig. 5.2 ZenStates' components: (a) the state machine name; (b) control buttons; (c) tasks associated with a state; (d) the initial state (in this case, also the one which is currently running, represented in green); (f) the blackboard; and transitions, composed by a guard condition (g) and its execution priority (e). © 2018 IEEE.

5.4 Introducing ZenStates

ZenStates is a specification model centered on the idea of *State machines*. A *State Machine* is defined as a set of abstract *States* to which users might add *Tasks* that describe (in terms of *high-level parameters*) what is going to happen when that particular state is being executed. These states are connected to one another via a set of logical *Transitions*. Execution always starts at the "Begin" state, following these transitions as they happen until the end. At any moment, inside a Task or a Transition, users can use *Blackboard* variables to enrich behaviors (e.g., use the mouse x position to control the volume, or to trigger certain transitions).

5.4.1 Enriched state machines as specification model

ZenStates borrows features from the enriched state machines model proposed by StateChart (Harel 1987) and StateMate (Harel et al. 1990). These works overcome typical problems of state machines–namely, the exponential growth of number of states, and its chaotic organization–by introducing:

- **Nested state machines (clustering):** One state could potentially contain other state machines;
- **Orthogonality (parallelism):** Nested state machines need to be able to execute independently and in parallel whenever their parent state is executed;
- **Zooming-in and zooming-out:** A mechanism athat allows users to navigate between the different levels of abstraction introduced by the nesting of state machines;
- **Broadcast communication:** That allows simple event messages to be broadcasted to all states, having the potential to trigger other states inside the machine.

When combined, these features result in a powerful hierarchical model of state machines. This model provides improved *organization* (i.e., state machines are potentially easier to understand),

[2] `https://www.ibm.com/us-en/marketplace/rational-rhapsody`

modularity (i.e., subparts of a system could be independently designed and later merged into a larger structure), and *expressiveness* (broadcast communication allows enhanced description of complex behaviors) when compared to the traditional states machine approach.

5.4.2 Key features

Building upon this model, ZenStates proposes five key features described in the following subsections.

Extending communication: the Blackboard

The blackboard–in the top-right of Figure 5.2–is a global repository of variables that can be defined and reused anywhere inside a state machine (i.e., inside states, nested states, tasks or transitions). Therefore, we extend the notion of broadcast communication introduced by (Harel 1987) because variables can be used for other purposes, and not only triggering transitions. In addition, because the blackboard is always visible on the interface, users can easily infer what inputs are available on the environment, as well as their updated values.

Users can interact with the blackboard using *pull* and *push* operations. Pull operations are accessed by using the variable's name (as in Gneiss (Chang and Myers 2014)). Push operations can be performed by tasks (see next subsection).

Some useful context-dependent variables can be automatically added to the blackboard and become directly accessible to users. In interactive environments, examples include mouse coordinates, key presses, various sensor inputs, incoming Open Sound Control (OSC[3]) messages, etc.

Making behaviors concrete: the Tasks

One challenge we needed to address concerns specifying concrete behaviors to happen inside the states. In the case of tools for developers, this behavior could be easily specified via programming language. But how could we make this easier for creative EUPs?

[3] `http://opensoundcontrol.org/`

We solve the issue by introducing the notion of tasks: simple atomic behaviors representing actions (if it happens only once) or activities (if it keeps occurring over time) and that can be attached to states as off-the-shelf components (Lawson et al. 2009).

Tasks work in the following way: Once a certain state is executed, all associated tasks are run in parallel. Each individual task also has its own set of idiosyncratic high-level parameters, that allow users to fine-tune its behaviors. In Figure 5.1, for instance, these high-level parameters are the intensity of the vibration, and the panning of a stereo audio-driven bass shaker. The UI controls to edit these parameters can be easily hidden/shown by clicking on the task, allowing users to focus their attention on the tasks they care the most.

Tasks are normally context-dependent, and need to be changed according to the domain of application. In creative interactive environments, for example, potential tasks could be: sound-related tasks (e.g., start sound, control sound, stop sound), light-related tasks (start, control, and stop light), and haptics related tasks (start, control, and stop haptics)–as shown in Figure 5.3.

There are however two categories of tasks which are not context dependent: the blackboard-tasks, and the meta-tasks.

Blackboard-tasks relate to creating new variables within the blackboard so that they can be later used anywhere inside the state machine. Our motivation is to increase reuse by providing users with a set of recurrent functionalities often found in interactive environments. Examples include oscillators, ramps, and random numbers.

Meta-tasks relate to extending ZenStates in situations where currently-supported functionalities are not enough. For example, OSC tasks allow communication with external media tools via the OSC protocol. JavaScript tasks allow custom JavaScript code to be incorporated to states. Finally, we have State Machine tasks, that allows nesting as shown in Figure 5.3.

Enriching transitions

We enrich transitions by incorporating two functionalities.

First, transitions make use of priorities that define the order they are going to be checked.

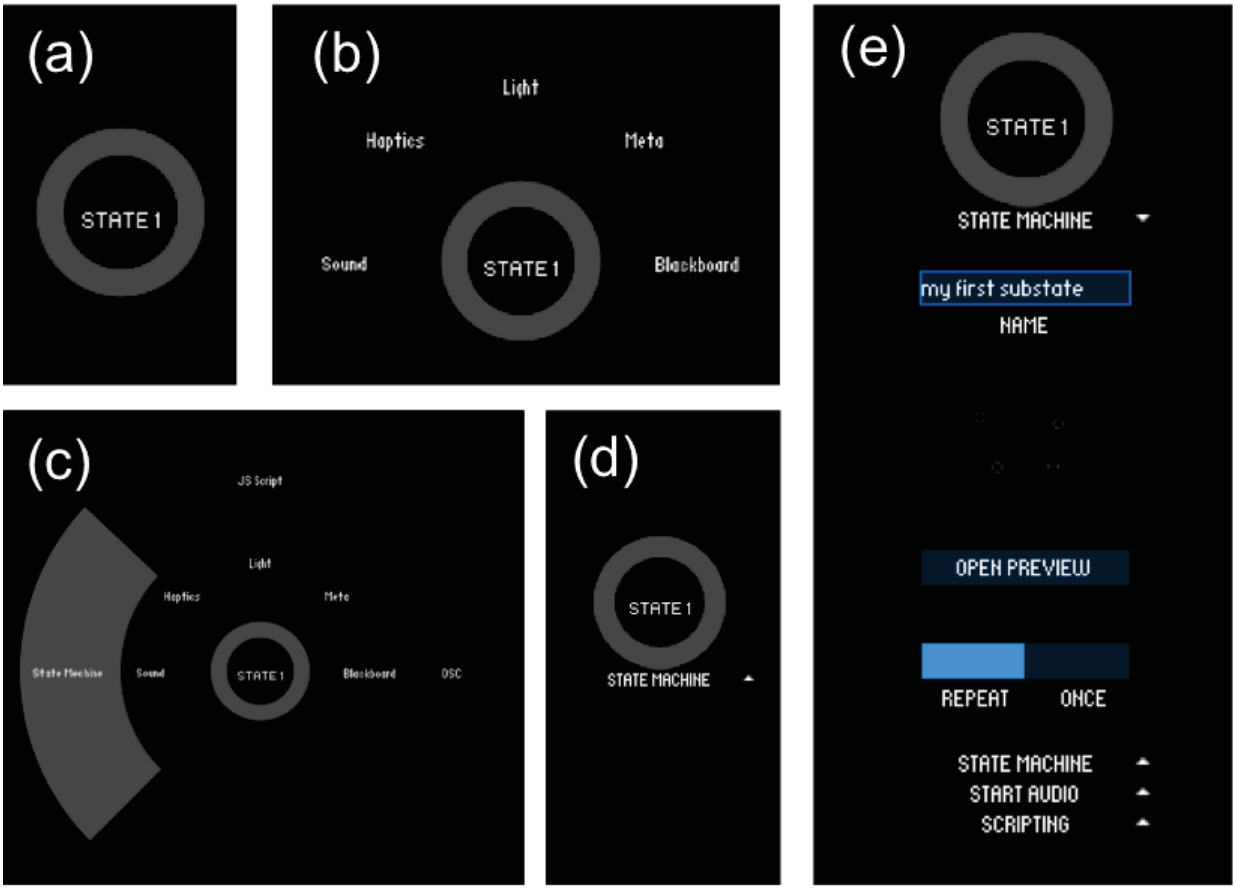


Fig. 5.3 Tasks inside states are reified with a contextual pie menu, as shown in the sequence above–(a), (b), (c), and (d). High-level parameters related to each task can be found by clicking on the task–as shown in (e). © 2018 IEEE.

This means that one state can have multiple transitions evaluated one after the other according to their priority (see Fig. 5.4). This allows users to write complex logical sentences similar to cascades of "if-then-else-if" in imperative languages. It also avoids potential logical incoherences raised by concurrent transitions.

Second, transitions support any Javascript expression as *guard conditions*, expressed as *transition and constraint events* as in (Oney, Myers, and Brandt 2014). In practice, this functionality combines logical operators (e.g. '`&&`', '`||`', '`!`'), mathematical expressions, and blackboard variables used either as events (e.g. '`$mousePressed`') or inside conditions (e.g. '`$mouseX > 0.5`'). For instance, it is possible to set that if someone enters a room, the light should go off. Or if the mouse is pressed on a certain x and y coordinates, another page should be loaded.

Self-awareness

Self-awareness describes a set of meta characteristics belonging to states and tasks that are automatically added to the blackboard and become available to users. For example, states can have a status (e.g., is it running? Is it inactive?), sound tasks can have the current play position, the volume, and the audio pan as properties, etc. This feature can be useful in several ways. For example, we can set a transition to occur when all tasks within a certain state are completed (e.g. '`$State.Done`'). Or setting the volume of an audio file to increase as its playback position increases.

Live development & Reuse

Finally, ZenStates also borrows two additional features from (ibid.) and (Chang and Myers 2014):

- **Live development:** Any element (e.g. tasks, transitions, and states) can be modified at runtime, allowing quicker process of experimentation and prototyping;
- **Reuse:** Previously defined interactive behaviors can easily be reused via nested state machines (that can be saved into files, exchanged, and then later imported). Here, we also

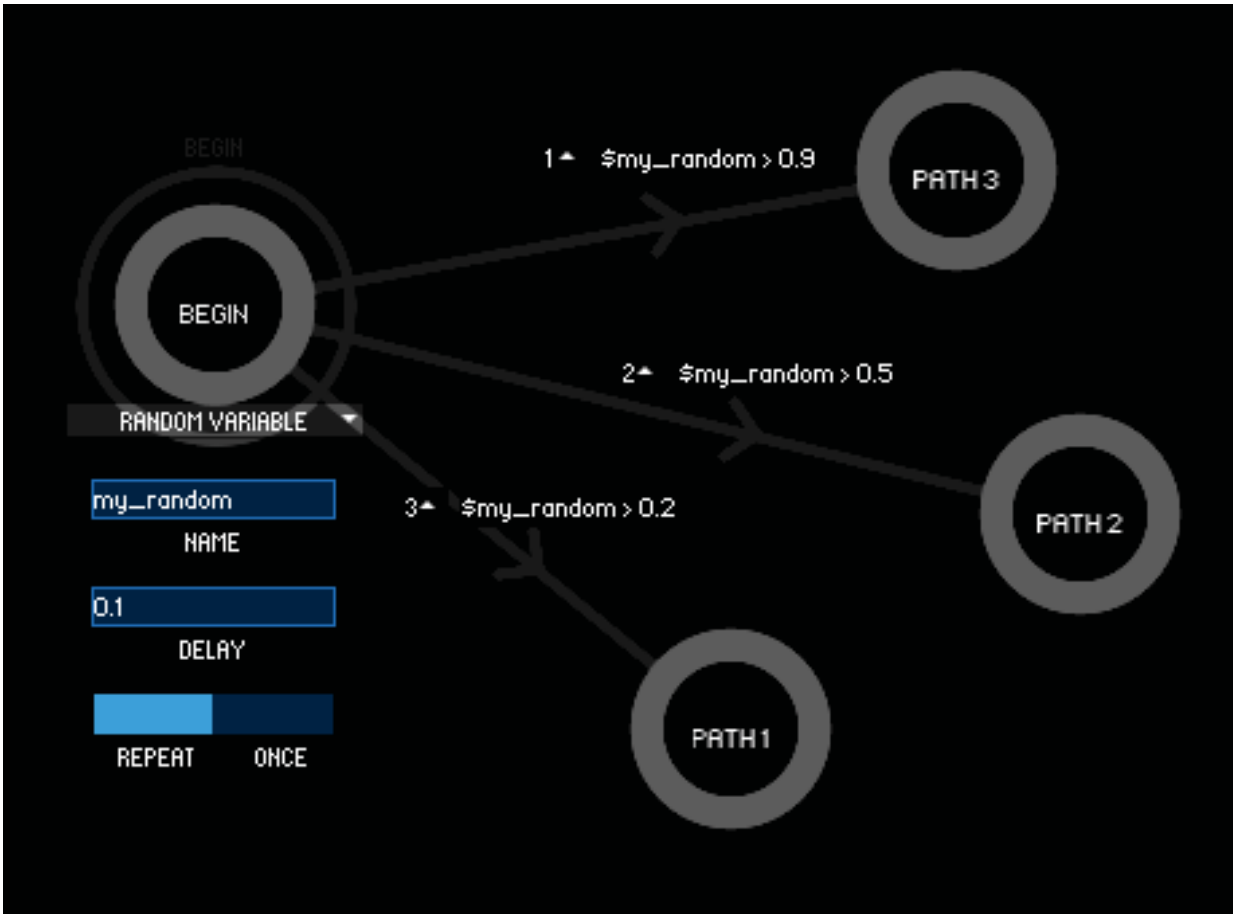


Fig. 5.4 Transitions have priorities attached to them. In this case, the blackboard variable '`$my_random`' generates a random number (between 0 and 1) which is then used to select a random path. The first transition to be checked is '`$my_random > 0.9`' because of its priority '1'. If false, transition with priority '2' would then be checked. This process goes on until all transitions are checked. In case none is satisfied, the current state is executed once again before the next round of checking (repeat button is enabled). Transitions can be added, edited, and deleted via direct manipulation.

expect to align with the principle of reuse as introduced by (Beaudouin-Lafon and Mackay 2000).

5.5 Implementation

The design process of ZenStates has been iterative and user-centered with media artists. Steps included: a) *analysis of existing alternatives*; b) *interviews with expert users*; c) *paper prototypes*; d) *development of scenarios to assess the potential of the tools in solving problems*; e) *observing the compositional process of two new media works*; f) *functional prototypes*; and g) *user-interface refinement over the course of think-aloud protocol sections with novice and expert users*. While a full description of this process is beyond the scope of this paper, its direct result is the specification model as presented in this paper.

To validate this model, we implemented all features described here in a functional software prototype. In this prototype, abstract elements (namely the state machine, states, tasks, transitions, and the blackboard) are reified–according to the principle of reification introduced in (ibid.)–to graphical objects that can be directly manipulated by the user (Shneiderman 1983). These visual representations become the core of the user interaction and also allow users to infer the current state of the system at any time during execution: the current running state is painted green; states already executed are red; inactive states are gray (see Fig. 5.2). All images presented on this paper are actual screenshots of this functional prototype, also demonstrated in the supplementary video.

The prototype has been implemented in Java, using Processing[4] as an external library for the graphics. The project is open-source and the source code is available online[5]. We stress that, as a prototype, this system is sill under development and its usability needs improvement.

[4] `http://processing.org/`
[5] `https://github.com/jeraman/zenstates-paper-vlhcc2018`

5.6 Evaluation

We argue ZenStates proposes an expressive (i.e., allow to develop a wide diversity of scenarios) yet easy-to-understand specification model for creative interactive environments. We have tested these claims in two steps. First, we have probed ZenStates' expressive power by designing 90 *exploratory scenarios*. Second, we have conducted a *user study* investigating ZenStates specification model in terms of its capacity to quickly and accurately describe interactive environments.

5.6.1 Exploratory scenarios

To explore the expressive power of ZenStates, we have developed 90 unique exploratory scenarios. These scenarios implement atomic audiovisual behaviors with different levels of complexity, based on a constrained set of inputs (either mouse, keyboard, or both), outputs (either background color of the screen, the sound of a sinewave, or both), and blackboard variables. The chosen behaviors are often used in the context of music/media arts. Examples include: Sinusoidal sound generators with user-controlled frequency (c.f., the project Reactable [6]) as implemented in the scenario *'mouse_ click_ mute_ mousexy_ freq_ amp'*; and contrasting slow movements and abrupt changes to create different moments in a piece (c.f., 'Test pattern' by Ryoji Ikeda [7]) implemented in the scenario *'random_ flickering_ random_ wait_ silence'*) . The full list of exploratory scenarios–with implementation and video demonstration–is available as supplementary material[8].

While small, these atomic scenarios can be further combined to one another via hierarchical state machines and transitions. Therefore, it is possible to create a potentially infinite number of new scenarios–much more complex than the atomic ones. This diversity illustrates the expressive power of ZenStates–especially considering the constrained set of input, outputs and blackboard variables that were used.

[6] `https://www.youtube.com/watch?v=n1J3b0SY_JQ`
[7] `https://www.youtube.com/watch?v=JfcN9Qhfir4`
[8] `https://github.com/jeraman/zenstates-paper-vlhcc2018/tree/master/scenarios`

5.6.2 User study

We investigated if ZenState's specification model makes the development of interactive environments easier to understand. This model was compared to the specification model used by two popular interactive environments development tools: Puredata (hereafter, the *dataflow* model), and Processing (hereafter, the *structured* model).

Hypothesis:

Our hypothesis is that Zenstates allows users to understand interactive environments specifications more *accurately*–that is, ZenStates would reduce the number of code misinterpretations by users–and *faster*–that is, ZenStates would reduce the amount of time necessary to understand the code–compared to the alternatives.

Subjects and Materials:

We recruited 12 participants (10 male, 1 female, and 1 not declared), aged 29 years old on average (min = 22, max = 46, $median$ = 26, SD = 7.98). Participants were all creative EUPs with previous experience with interactive environments tools (e.g. media artists, composers, designers, and technologists), either professionals or students. Their expertise on computer programming ranged from novices to experts (min = 2 years, max = 23 years, $mean$ = 7.67, $median$ = 6, SD = 5.48). The most familiar languages for them were Python and Max/MSP (both with 4 people each), followed by Processing (2 people).

Regarding the experimental material, we have selected 26 of our exploratory scenarios representing a wide diversity of usage scenarios (i.e., either using blackboard variables; single input; and multimodal input). Each scenario was then specified using the three evaluated specification models (ZenStates, dataflow, and structured), resulting in 78 specifications. All these scenarios and their specifications are available online[9].

[9] `https://github.com/jeraman/zenstates-paper-vlhcc2018/tree/master/data%20collection/html/scenarios`

Procedure:

Our experimental procedure is based on (Kin et al. 2012), adapted and fine-tuned over a preliminary pilot with 6 participants. The resulting procedure is composed of 3 blocks–one block per specification model tested–of 6 trials.

In each block, participants were introduced to a specification model as presented in Figure 5.5. Participants were also given a small printed cheatsheet containing all possible inputs (i.e., mouse and keyboard), actuators (i.e., screen background color, and a sinewave), and language specific symbols that could appear over the trials. At this point, participants were allowed to ask questions for clarification, and were given some time to get used to the model.

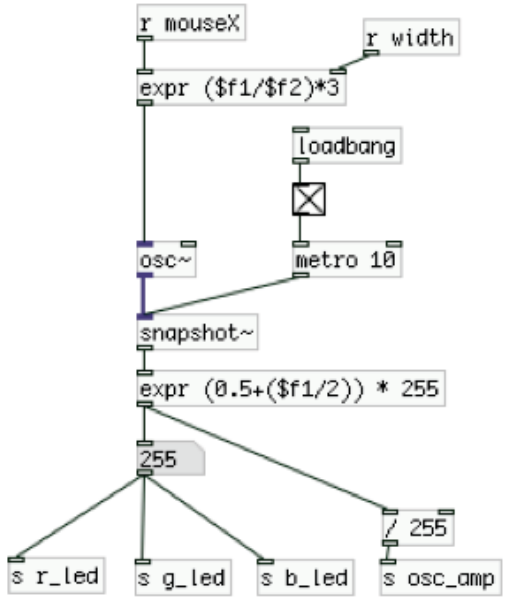
After this introduction, participants were introduced to one interactive environment specification–either ZenStates, dataflow, or a structured language–and to six different interactive environments videos showing real behaviors. Their task was *to choose which video they thought that most accurately corresponded to the specification presented*, as shown in Figure 5.6. Participants were instructed to be the most accurate, and to chose a video as quick as they could–without sacrificing accuracy. Only one answer was possible. Participants were allowed two practice trials and the cheatsheet could be consulted anytime.

Participants repeated this procedure for all three evaluated alternatives. Presentation order of the models was counterbalanced to compensate for potential learning effects. Similarly, the order of the six videos was randomized at each trial.

Our measured dependent variables were the *decision accuracy* (i.e., the percentage of correct answers), and the *decision time* (i.e., the time needed to complete the trial). As in (ibid.), the decision time was computed as the trial total duration minus the time participants spent watching the videos. The whole procedure was automatized via a web application[10].

Finally, participants were asked about the specification models they had considered the easiest and the hardest to understand, followed by a short written justification.

[10] `https://github.com/jeraman/zenstates-paper-vlhcc2018/tree/master/data%20collection`




```
SinOsc sine = new SinOsc(this);

void draw() {
  float time = millis()/1000.0;
  float amplitude = 0.5;
  float frequency = ((float)mouseX/wid
  float value = 0.5;

  value = value + (float) (amplitude *
         Math.sin((2*Math.PI*frequency
  sine.amp(value);
  value = value*255;
  background(value, value, value);
}
```

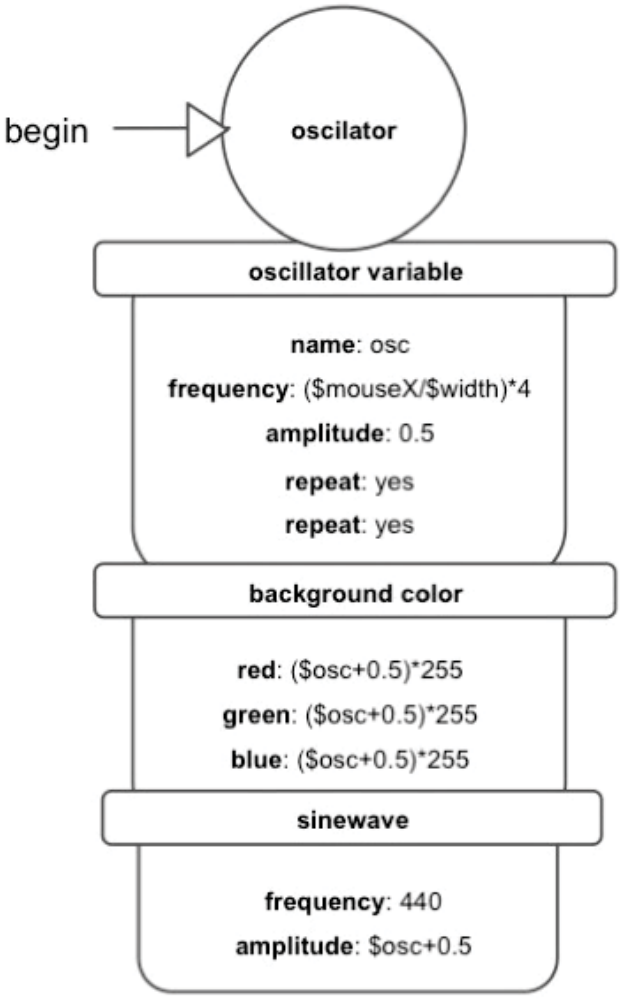


Fig. 5.5 The three specification models presented to the participants: dataflow (top-left), structured (bottom-left), and ZenStates (right). © 2018 IEEE.

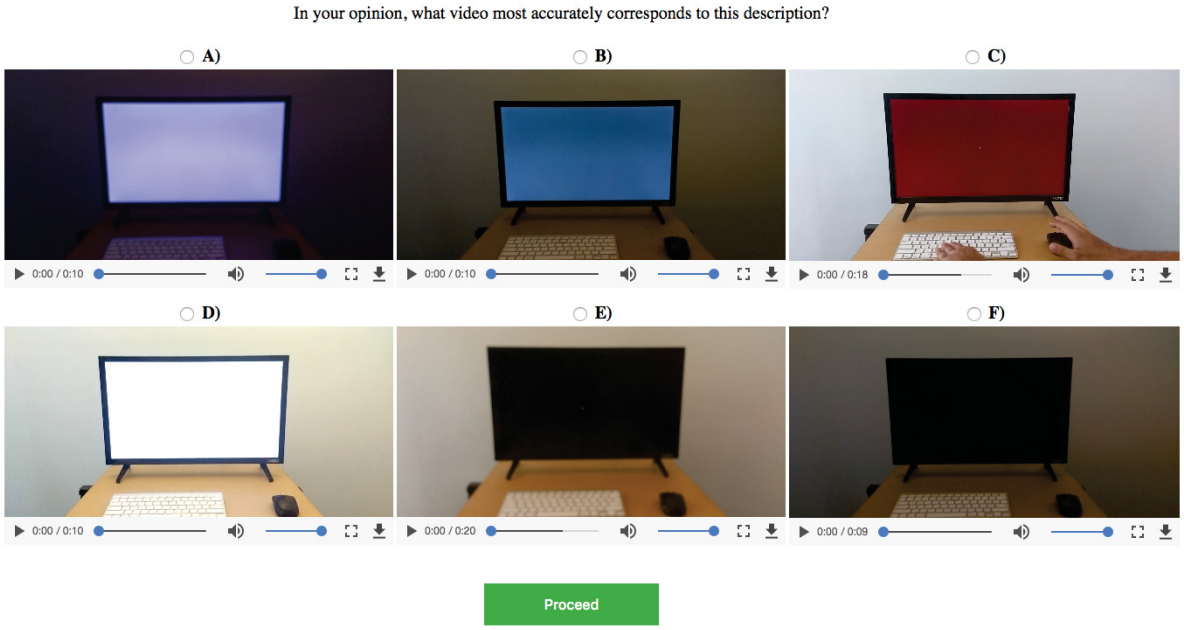


Fig. 5.6 Being introduced to one specification, participants needed to choose which one among six videos they thought that most accurately corresponded to the specification presented. © 2018 IEEE.

Results:

Data from HCI experiments has often been analyzed by applying null hypothesis significance testing (NHST) in the past. This form of analysis of experimental data is increasingly being criticized by statisticians (M. Baker 2016; Cumming 2014) and within the HCI community (Dragicevic 2016; Dragicevic, Chevalier, and Huot 2014). Therefore, we report our analysis using estimation techniques with effect sizes[11] and confidence intervals (i.e., not using p-values), as recommended by the APA (VandenBos 2007).

Regarding *decision time*, there is a strong evidence for ZenStates as the fastest model (mean: 40.57s, 95% CI: [35.07 - 47.27]) compared to the dataflow (mean: 57.26s, 95% CI: [49.2 - 67.5]), and the structured model (mean: 70.52s, 95% CI: [59.16 - 85.29]), as shown in Figure 5.7. We also computed pairwise differences between the models (Fig. 5.8) and their associated confidence

[11] Effect size refers to the measured difference of means–we do not make use of standardized effect sizes which are not generally recommended (Baguley 2009).

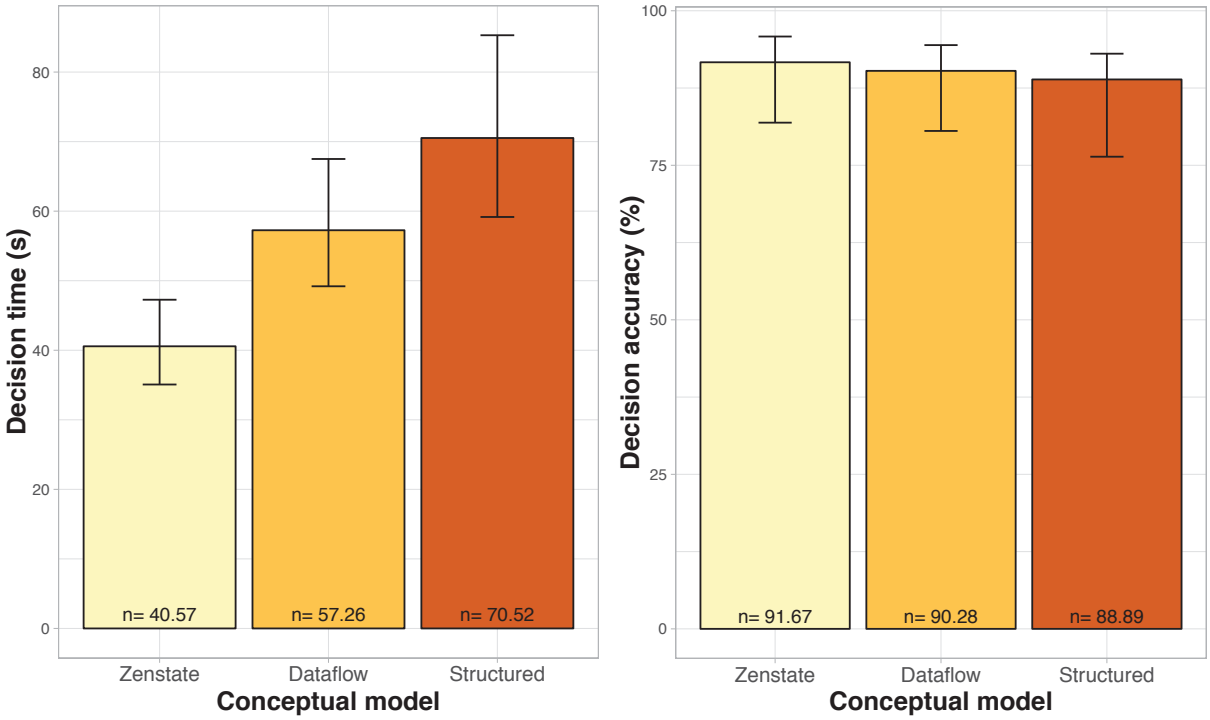


Fig. 5.7 *Decision time* (left) and *Accuracy* (right) per specification model. Error Bars: Bootstrap 95% CIs. © 2018 IEEE.

intervals. Results confirm that participants were making their decision with ZenStates about 1.4 times faster than with the dataflow and 1.8 times faster than with the structured model. Since the confidence interval of the difference between dataflow and structured models overlaps 0, there is no evidence for differences between them.

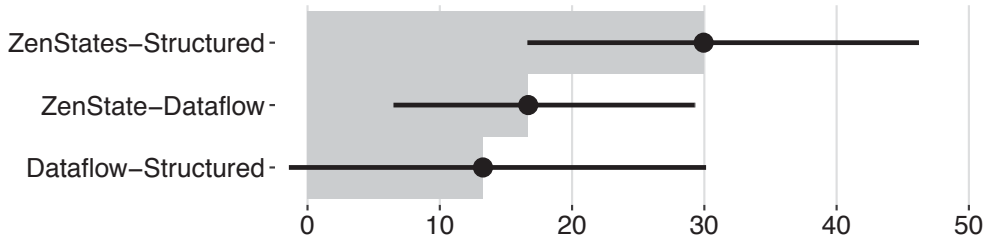


Fig. 5.8 *Decision time* pairwise differences. Error Bars: Bootstrap 95% CIs.

Regarding *decision accuracy*, participants achieved 91.67% (95% CI: [81.89 - 95.83]) accuracy with ZenStates, 90.28% (95% CI: [80.55 - 94.44]) with the Dataflow model, and 88.89% (95% CI: [76.39 - 93.05]) with the structured model (see Fig. 5.7(right)). These results show quite high accuracy with all the specification models overall, although there is no evidence for one of them being more accurate than the other.

Finally, regarding the final questionnaire data, ZenStates was preferred by participants as the *easiest model to understand* (8 people), followed by the structured (3 people), and the dataflow (1 person) models. Participants written justifications reported aspects such as *"graphic display"*, *"code compartmentalization"*, and *"abstraction of low-level details"* as responsible for this preference. Similarly, dataflow (8 people) was considered the *hardest model to understand*, followed by structured (3 people), and ZenStates (1 person)[12].

5.7 Limitations

Our study also revealed limitations in our specification model and its software interface, namely:

- **The blackboard:** Currently represented as a two-columns table on the top right of the

[12]The raw collected data is presented on Appendix B.1 and its analysis script is presented on Appendix B.2.

screen, containing variable name and its value. While this initial approach fulfills its goals (i.e., enhancing communication), it has limitations. A first limitation deals with representing a large amount of sensor data in the screen. For example, if a 3D depth camera is attached to the system (tracking x, y, and z points for all body joints), all detected joints would be added to the blackboard. For users interested in specific information (e.g. hand x position), the amount of visual data can be baffling. Another limitation concerns lack of support to high-level features (e.g. derivatives, and averages), and filtering, which are often as useful as raw sensor data. Further research is needed to improve the blackboard on these directions;

- **Physical environment alignment:** ZenStates assumes that the physical environment (i.e., sensor input, hardware for output media) and the tasks it supports (i.e., sound-related, light-related) is static and consistent, and that it would remain consistent along the execution. This assumption is reasonable when dealing with standard interaction techniques (e.g. WIMP tools for desktop, as in SwingStates (Appert and Beaudouin-Lafon 2008)). However, because we are dealing with potentially more complex setups, it is possible that the assumption is no longer possible in certain cases. For example, in a certain artistic performance, some sensors might be added, or some light strobes removed during the performance. How to maintain this environment-software consistency in these dynamic cases? How should ZenStates react? These are open questions that need to be addressed in future developments;

- **Interface usability and stability:** The evaluation performed so far focuses only on readability of our specification model, not addressing ease of use or usability of the prototype interface that implements the model. We reason that ease of use and usability are not as relevant in such prototype stage as already-known problems would show up, limiting the evaluation of our specification model. At this stage, readability seems more effective as it could make specifications easier to understand, and potentially easier to learn, reuse, and extend. Nevertheless, we highlight that the usability of such interface would play a signif-

icant role in the effectiveness of our model. Examples to be improved include the obtuse expression syntax, using '$' to instantiate variables, and the small font size;

In addition to addressing these problems, we also plan to explore the usage of ZenStates in other creative contexts and to implement principles that could improve general support to creativity inside ZenStates (see (Shneiderman 2007) for examples).

5.8 Conclusion

In this paper, we have analyzed the state of the art of development tools for programming rich interactive behaviors, investigating how these tools could support creative end-user programmers (e.g., media artists, designers, and composers) who typically struggle with technical development. As a solution, we introduced a simple yet powerful specification model called ZenStates. ZenStates combines five key contributions–1) the blackboard, for communication; 2) the tasks, for concrete fine-tunable behaviors; 3) the prioritized guard-condition-based transitions; 4) self-awareness; and 5) live development & reuse–, exploring those in the specific context of interactive environments for music and media arts.

Our evaluation results suggest that ZenStates is expressive and yet easy to understand compared to two commonly used alternatives. We were able to probe ZenStates' expressive power by the means of 90 exploratory scenarios typically found in music/media arts. At the same time, our user study revealed that ZenStates model was on average 1.4 times faster than dataflow model, 1.8 times faster then the structured model in terms of *decision time*, and had the highest *decision accuracy*. Such results were achieved despite the fact participants had no previous knowledge of ZenStates, whereas alternatives were familiar to participants. In the final questionnaire, 8 out of 12 participants judged ZenStates as the easiest alternative to understand.

We hope these contributions can help making the development of rich interactive environments–and of interactive behaviors more generally–more accessible to development-struggling creative communities.

Acknowledgment

The authors would like to thank Sofian Audry and Chris Salter for their support and numerous contributions, and everyone involved in the project *'Qualified Self'* for their time.

This work was partially supported by the NSERC Discovery Grant and the *Fonds de recherche du Québec–Société et culture* (FRQSC) research grant.

Chapter 6

Instrumental Music Skills as Input Primitive for Visual Creative Authoring

Chapter 5 concluded my second case study. This chapter presents my third–and final–case study addressing DM, now for music authoring tools. It is a full manuscript ready to be submitted as:

> `Jeronimo Barbosa`, Marcelo M. Wanderley, Stéphane Huot. *"Instrumental Music Skills as Input Primitive for Visual Creative Authoring"*.

6.1 Abstract

Musicians devote years developing fine instrumental skills to achieve mastery of musical instruments. Yet, domain-specific end-user programming environments remain (i) often centered on mouse and keyboard as primitive input method and (ii) poorly designed for incorporating these expert instrumental skills as input. We investigate the use of instrumental skills as input primitive for enriching creative end-user programming. The result is *StateSynth*: a direct manipulation-based music authoring environment that empowers musicians to build personal interactive musical tools. We evaluate StateSynth's learning thresholds and expressive ceilings in a one week study with two expert musicians. Results suggest that StateSynth accessibly allowed these musicians

to use instrumental skill-driven authoring, enabling at the same time a wide diversity of musical results, highlighting the potential of instrumental skills as input primitive for creative authoring.

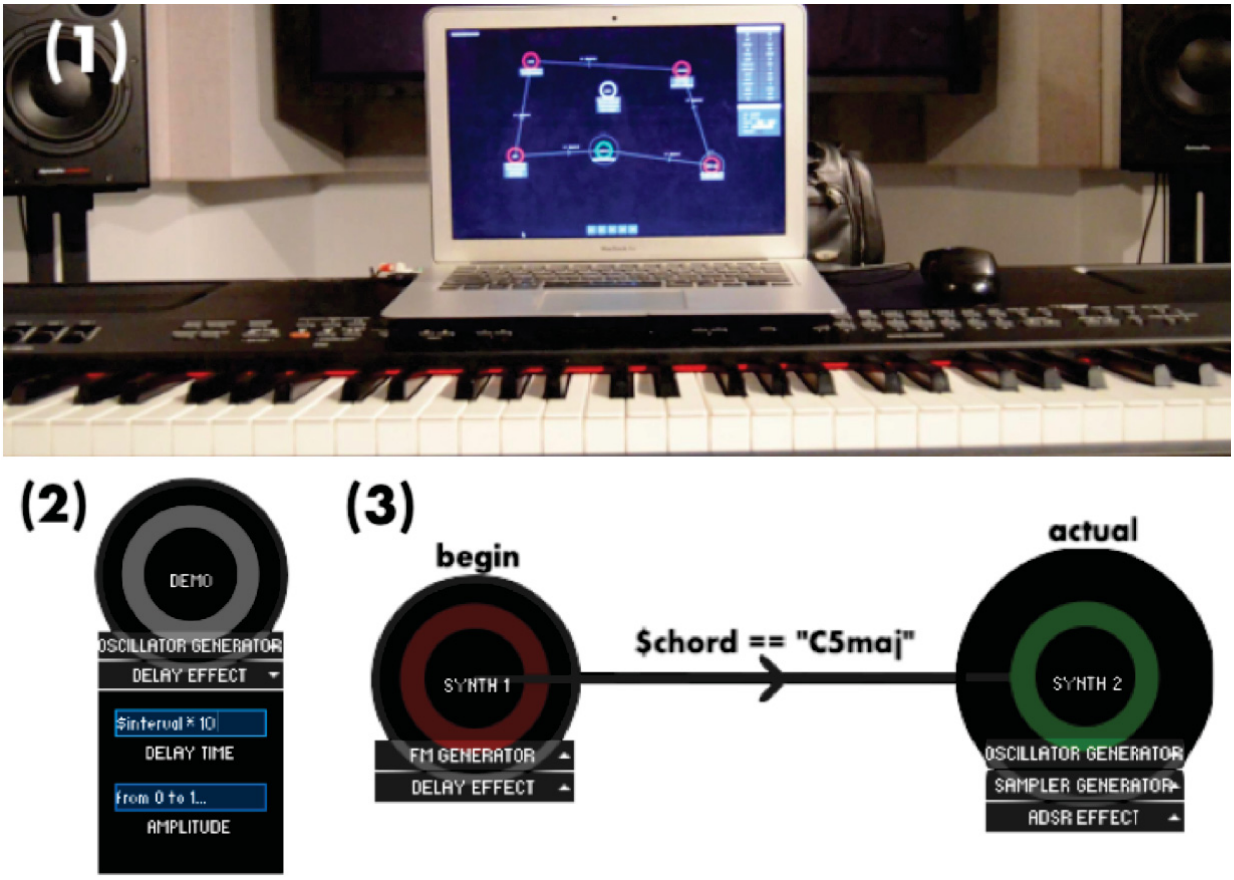


Fig. 6.1 *StateSynth* is connected to a MIDI-enabled piano as presented on image (1). StateSynth captures instrumental piano skills, making them visually available in realtime within the authoring environment via *blackboard variables*. These variables can then be used in a diversity of interactive musical behaviors, such as image (2), where musical intervals dynamically modulate the delay time of a sinusoidal oscillator, and image (3), where playing specific chords yield different synthesizer presets.

6.2 Introduction

Musicians devote years–not rare a lifetime–pursuing mastery in musical instruments. While achieving such mastery is challenging, it is commonly accepted that mastery involves developing *instrumental music skills* (Ericsson 2008; G. E. McPherson 2005)–a complex and multifaceted expertise

composed of fine motor skills and musical knowledge–that enable the embodiment of low level mechanics of the musical instrument. For example, while novice piano players struggle with finding the right notes on a keyboard, or coordinating both hands and feet together, expert players are able to abstract these mechanics–incorporated through years of practice and repetition–and articulate thoughts in terms of these mechanics, via musical elements such as *intervals*, *phrasing*, *rhythm*, and *chords*.

Instrumental music skills have not remained unnoticed in HCI. Some researchers have explored instrumental music skills as input method for user interaction. Ghomi and colleagues showed that rhythmic patterns can be effective and expressive for triggering commands in diverse input modalities (Ghomi, Faure, et al. 2012). Feit and Oulasvirta explored the piano as an input method for text entry, enabling similar speed performance of QWERTY keyboards with only a fraction of the training (Feit and Oulasvirta 2014). The idea of chords as multi-finger gestures have been explored in chord keyboards (Engelbart 1962), and continues to be useful today in contexts such as multitouch input (Ghomi, Huot, et al. 2013; Wagner, Lecolinet, and Selker 2014). These works highlight the potential of adopting instrumental music skills as input method for human-computer systems.

Despite this potential, instrumental music skills are often poorly integrated with musical end-user programming environments (Ko, Myers, Rosson, et al. 2011)–centered on mouse and keyboard input instead. Examples such as Max/MSP (Puckette 2002) and SuperCollider (McCartney 2002) have enabled many musicians to get started on programming, but are distant from traditional instrumental practice, and are considered challenging by non-technical musicians, who urge for more "musician-oriented interfaces" (Bullock, Beattie, and Turner 2011). Others have explored alternative gestural inputs for authoring: The Reactable is a multitouch tabletop display with tangible controls (Jordà, Geiger, et al. 2007); The Mus*ink* integrates interactive paper and pen annotations to OpenMusic authoring software (Tsandilas, Letondal, and Mackay 2009); The Auraglyph is a sketch-based modular programming environment for tablets (Salazar 2017). Still, none of these systems focus on incorporating instrumental music skills as input primitive. Indeed, how can one

leverage instrumental musical skills in end-user programming?

This paper investigates the use of instrumental music skills as primitive input method for creative authoring, focusing on the piano. Unlike previous related works focused on hand drawing (Jacobs, Brandt, et al. 2018; Jacobs, Gogia, et al. 2017; Xia et al. 2018), our approach uses the metaphor of analog synthesizers (Jenkins 2007), where musicians sculpt sound by tweaking diverse audio parameters and playing a MIDI-enabled piano. As musicians play, we dynamically capture musical elements such as intervals, dynamics, chords, motifs, phrasing, and rhythm, making these elements visually available on the interface (Beaudouin-Lafon 2000). These elements can then be used in interactive behavior authoring as presented in Figure 6.1. This approach is implemented on *StateSynth*: a direct manipulation-based music authoring environment that empowers musicians to build personal interactive musical tools.

Our contributions are three-folded. **First**, we survey related works and articulate *instrumental musical skills as primitive input* in terms of three design guidelines: (1) *leveraging existing musical expertise*; (2) *lowering specification model complexity*; and (3) *increasing authoring environment directness*. **Second**, we discuss how StateSynth has been designed after these guidelines, using visual reification (ibid.) and direct manipulation (Shneiderman 1983) of hierarchical finite-states machines, expressions, off-the-shelf components called Tasks, and a global communication system called the Blackboard. **Third**, we provide evidence for the potential of our approach, by reporting a one week study with two expert musicians assessing StateSynth's learning thresholds and expressive ceilings.

6.3 Background

Exploring artistic manual capabilities as input for creative authoring is a exciting research direction. Jacob and colleagues presented the Dynamic Brushes (2018), a visual authoring environment that provides procedural art techniques to traditional manual drawers, centered on drawing input as primitive component for the programming. Previously, Jacobs and colleagues explored how these procedural techniques could be visually represented and direct manipulated by artists

in a widget-based tool called Para (Jacobs, Gogia, et al. 2017). Xia et al. also explore direct manipulation, but combining instead pen and touch inputs to leverage existing drawing skills for personal data visualization (Xia et al. 2018). The result is DataInk, a web system that enables users to directly manipulate their hand-drawn sketches, allowing dynamic data binding, layout automation, and visual customization. In a similar direction, Draco is a sketch-based interface that enables artists to add dynamic animations to static drawings (Kazi et al. 2014). Despite of these advances, all these works focus on drawing as context, failing to address the potential of instrumental music skills as input.

While not targeted at creative authoring, some have addressed instrumental music skills as input. Feit and Oulasvirta (2014) propose four qualities of the piano as an input device–namely *redundancy*, *chords*, *skill transfer*, and *sound feedback*–incorporating these qualities into a piano-based text-entry system called PianoText, which allowed users to achieve performance rates similar to QWERTY keyboards with only a fraction of the training. Ghomi et al. (2012) introduced *rhythmic interaction*, exploring rhythmic patterns as input method. Deepening the role of temporal dimension on user interaction, the authors demonstrate how rhythmic patterns can be as viable as keyboard shortcuts when associated with visual cues, and are suited to trigger commands in diverse input modalities. Wagner, Lecolinet, and Selker (2014) explore the potential of *multifinger chord gestures* as input for tablets, proposing a chord vocabulary that achieved high recognition accuracy rates and high user memorability, while Ghomi et al. introduced Arpège (2013), a technique to support learning and performing chord gestures that allowed users to memorize a high number of chord gestures. The application of these instrumental music-based approaches in creative authoring–specially in music–remains, to our knowledge, to be seen.

6.3.1 Creative authoring for music

There are two main categories of creative authoring tools for music. On the one hand, Digital Audio Workstations (DAWs) offer a wide range of features accessible via widget-based UI, e.g., Ableton Live (Live 2019), Logic (Pro 2019), and Garage Band (Band 2019). Despite the arguable

ease-of-use, these systems provide limited support to interactivity–often reduced to linearly mapping musical controllers to features. On the other hand, with significantly higher expressive power and not as friendly as DAWs, we have domain-specific programming languages for computer music (Dannenberg 2018; Loy and Abbott 1985), including dataflow-based, e.g., Max MSP (Puckette 2002) and Puredata (Puckette 1997), and textual languages, e.g., SuperCollider (McCartney 2002) and ChucK (Wang and Cook 2004). The two categories present contrasting levels of learnability thresholds–describing if novices can easily get started with the tool–and expressive ceilings–describing if the tool can afford expressive results desired by experts (Myers, Hudson, and Pausch 2000; Shneiderman 2007; Wessel and Wright 2002). Finding a balance between them is often challenging: Focusing too much on the former can yield limited ceilings; Focusing too much on the latter can yield high-thresholds (Myers, Hudson, and Pausch 2000).

Aiming at this balance, several solutions have been proposed. Bullock and colleagues user-centered designed Integra Live (Bullock, Beattie, and Turner 2011). Integra offers more high level features to musicians, combined with three different interface perspectives, namely arrange view, live view, and timelines. Preliminary usability testing suggest that Integra Live is easier to use and enabled people to be more expressive than in other tools they had previously tried. The i-Score is a timeline-oriented interface enriched with temporal structured programming primitives such as sequences and conditionals (Celerier, Desainte-Catherine, and Couturier 2016), and the Iannix introduces an idiosyncratic conceptual model based on relational links among different geometrical shapes in 2D and 3D spaces (Jacquemin and Coduys 2014). AuraFX is a high level software tool with highly customizable and parametrizable sound effects (Dannenberg and Kotcher 2010). This high level structure, the authors argue, offer sophisticated interactive sound effects control to non-programmers–despite being less expressive than computer music programming languages.

As a drawback, all the systems presented so far are centered on mouse and keyboard as input method. Within the *NIME* (*New Interfaces for Musical Expression*) and the ICLC (*International Conference on Live Coding*) conferences, some urge instead for embodied input approaches (Salazar and Armitage 2018) more oriented to musical gestures (Jensenius et al. 2010). Examples of

such approaches include: visual tabletop multitouch interfaces with tangible controls for sound synthesis, e.g., Reactable (Jordà, Geiger, et al. 2007) and Audiopad (Patten, Recht, and Ishii 2002); paper-based interfaces where manual annotations are integrated to computer music languages, such as Mus*ink* (Tsandilas, Letondal, and Mackay 2009) and the paper substrates (Garcia et al. 2012); and Auraglyph, a sketch-based modular programming environment for tablets (Salazar 2017). Still, none of these systems engage with instrumental music skills as primitive input within their designs.

Our approach aims at addressing this gap. We believe that instrumental music practice can guide a next generation of creative authoring systems for music. An early insightful vision on this direction is the CodeKlavier, which enables pianists to live code music while/by playing the piano (Veinberg and Noriega 2018). Here, however, as in the PianoText (Feit and Oulasvirta 2014), piano input is limited to text-entry method for a textual live code environment.

6.3.2 Fostering creativity in authoring systems

Creative authoring systems can be framed as a subset of end-user programming (Ko, Myers, Rosson, et al. 2011) known as exploratory programming (Beth Kery and Myers 2017), where exploration of new creative ideas is more important than traditional success criteria such as quality of code. One approach for fostering such creative exploration is making interactive behaviors authoring easier–either by addressing the *specification model* or the *authoring environment*.

Regarding the specification model, Bailey and Kostan (2003) carry out interviews and a survey investigating challenges faced by professional multimedia designers in communicating and exploring rich interactive behaviors. Results suggests that (1) visual "arrow-centered" techniques (e.g. mind map, flowcharts, storyboards) are among the most used tools to communicate ideas; and (2) designers prefer to focus on content rather than "spending time" on programming or learning new tools. Myers and colleagues (2008) investigated interaction design practice and found that prototyping interactive behaviors often involved dealing with state changes, which were considered harder to prototype than appearance. These works provide us with evidence that state machines

metaphor can potentially make interactive behaviors authoring easier. Not surprisingly, several works explored this direction. Harel's StateCharts (1987; 1990), HsmTk (Blanch and Beaudouin-Lafon 2006) and SwingStates (Appert and Beaudouin-Lafon 2008) are some notable examples applied to the context of expert programmers.

Others shed light into how the authoring environment itself might impact authoring. Ko and colleagues (2004) present a study where 40 novices end-user programmers had to learn Visual Basic scripting over five weeks, where learning issues faced by these students were logged. A qualitative analysis of the resulting 130 logs showed six categories of learning barriers: design, selection, coordination, use, understanding, and information. In particular, more than half of the learning barriers were categorized either as use-related (e.g. programming environments not providing enough cues about using their own features, such as in syntax-related issues) or understanding-related (e.g. lack of transparency in how the program executes, how it moves from the input until the output). Considering the challenge of increasing execution transparency and providing cues about the use of different features, Victor provides us with inspiring visual approaches for tackling these challenges, aiming at making code more understandable for non-programmers (Victor 2012).

Both elements–visual state machines along with transparent environments–are combined in recent works. Examples include the ConstraintJS (Oney, Myers, and Brandt 2012) and InterState (Oney, Myers, and Brandt 2014), for the context of web development; ZenStates (Barbosa, Wanderley, and Huot 2018), for immersive multimedia environments; Vizir (Conversy et al. 2018), for operating airport automations; the Dynamic Brushes (Jacobs, Brandt, et al. 2018), for drawing. Yet, none of these engage with expert instrumental music practice for authoring.

6.4 Instrumental Music Skills as Input Primitive

Informed by our literature, we articulate *instrumental music skills as input primitive* in terms of three design guidelines:

6.4.1 DG1: Building upon existing musical expertise

The first guideline concerns incorporating music-specific expertise inside the authoring environment, allowing participants to profit from their existing expertise. This guideline translates to several concrete strategies. One example is incorporating the piano itself as input method, dynamically capturing aspects of musicians' playing, e.g., notes, intervals, chords, dynamics, harmony or rhythm, and making these aspects available within the programming. Another example concerns adopting sound synthesis modules popularly used in music authoring systems. A third example concerns the prevalence of familiar music structure at the core of the environment (e.g., tempo and key), instead of arbitrary general purpose programming concepts.

6.4.2 DG2: Lowering specification model complexity

The second guideline concerns adopting an easy-to-understand specification model, making interactive behaviors authoring with the captured musical expertise faster and easier to learn. By reducing the technical effort required to understand the model, we enable artists to keep their focus on the aesthetic-level instead. One strategy is adopting a state machines-based specification model. Another strategy is encapsulating low-level technicalities into customizable high-level components, with default behaviors that can be easily fine-tuned by users.

6.4.3 DG3: Increasing authoring environment directness

The third guidelines concerns increasing directness for manipulating the captured musical expertise within the authoring environment. The motivation is allowing users to spend less time dealing with actions that do not have a direct impact on the musical outcome (e.g., interface navigation), allowing them to keep their attention on the musical activity, aiming at promoting engagement and flow (Chirico et al. 2015). Another goal is promoting transparency–that is, allowing the user to infer accurately what is happening inside the device. One strategy to achieve this is visually reifying–according to the principle of reification as introduced in (Beaudouin-Lafon 2000)–abstract elements used for programming, transforming these elements into responsive visual objects that

can be directly manipulated by the user (Shneiderman 1983). A second strategy concerns reducing the number of immediately visible input controls, aiming at reducing confusion that a high number of input controls may cause to new users (Barbosa, Wanderley, and Huot 2017).

6.5 Introducing StateSynth

Our guidelines oriented a proof-of-concept music authoring environment called *StateSynth*, implemented in Java, using Processing[1] as library for the graphics, Minim for sound synthesis[2], and JFugue[3] for MIDI and music theory parsing. The features described here are currently implemented and prototype is functional. In this section, we introduce StateSynth in terms of the core elements: State Machine, States, Tasks, Transitions, and the Blackboard. These elements compose an easy-to-understand yet expressive specification model (DG2) recently detailed elsewhere (Barbosa, Wanderley, and Huot 2018).

6.5.1 State machines, states, and tasks

StateSynth is centered on *State machines*. A State Machine is defined as a set of *States*, an abstract entity that represents one piano configuration. States have no concrete behavior by themselves. Concrete behaviors can be added via *Tasks*. Tasks describe individual actions that happen when that particular state is being executed. Following DG1, these tasks represent high-level building blocks of computer music (Pejrolo and Metcalfe 2016)–a list of all available tasks is presented on Table 6.1. Once one state is executed, all tasks associated to this state execute in parallel. As discussed in DG2, tasks are ready to be used off-the-shelf and can be easily fine-tuned in terms of a set of high level parameters. Furthermore, each high level parameter informs what range of input values can be used. For example, the amplitude of a oscillator generator in Figure 6.2 highlights that values between 0 and 1 should be used.

By playing the piano, users can immediately hear sounds shaped by the chosen task (DG1,

[1] `http://processing.org/`
[2] `https://github.com/ddf/Minim/`
[3] `http://www.jfugue.org/`

Generators	Oscillator	*Synthesizes sound with a given waveform*
	FM Synth	*Basic frequency modulated synthesizer*
	Sampler	*Plays sound from file, which is tuned according to the key played*
Augmenters	Note	*Plays an artificial note along with the user*
	Interval	*Plays an artificial musical interval along with the user*
	Chord	*Plays an artificial chord along with the user*
Effects	Delay	*Delays the input audio in time*
	Flanger	*Classic flanger audio effect*
	ADSR	*Envelopes the input audio with attack, decay, sustain, and release*
	BitChrush	*Reduces the bit resolution of the input audio signal*
	Filter	*Filters the input audio according to certain frequency characteristics*
Meta	State Machine	*Creates a nested state machine inside a state*
	JS Script	*Attaches Javascript code to a state*
	OSC Message	*Sends Open Sound Control (OSC) messages*
Blackboard	Random	*Creates a random value in the blackboard*
	Ramp	*Creates a ramping value in the blackboard*
	Low Frequency Oscillator (LFO)	*Creates a sinusoidal-moving value in the blackboard*
	Default	*Create an arbitrary mathematical expression in the blackboard*

Table 6.1 List of all tasks provided by StateSynth. These tasks are separated into five categories: (1) *generators*: tasks that produce sound; (2) *augmenters*: music theory-centered tasks that extends the musical content played by the user; (3) *effects*: tasks that modifies the sound currently produced; (4) *meta*: tasks that extends the states-machine-centred specification model, allowing users to go beyond it; and (5) *blackboard*: tasks that allow users to create new blackboard variables.

DG3). Therefore, while states and tasks are the most elementary components of StateSynth, expert keyboard players can already achieve meaningful sound results by combining keyboard playing with default and non-default values for high-level parameters.

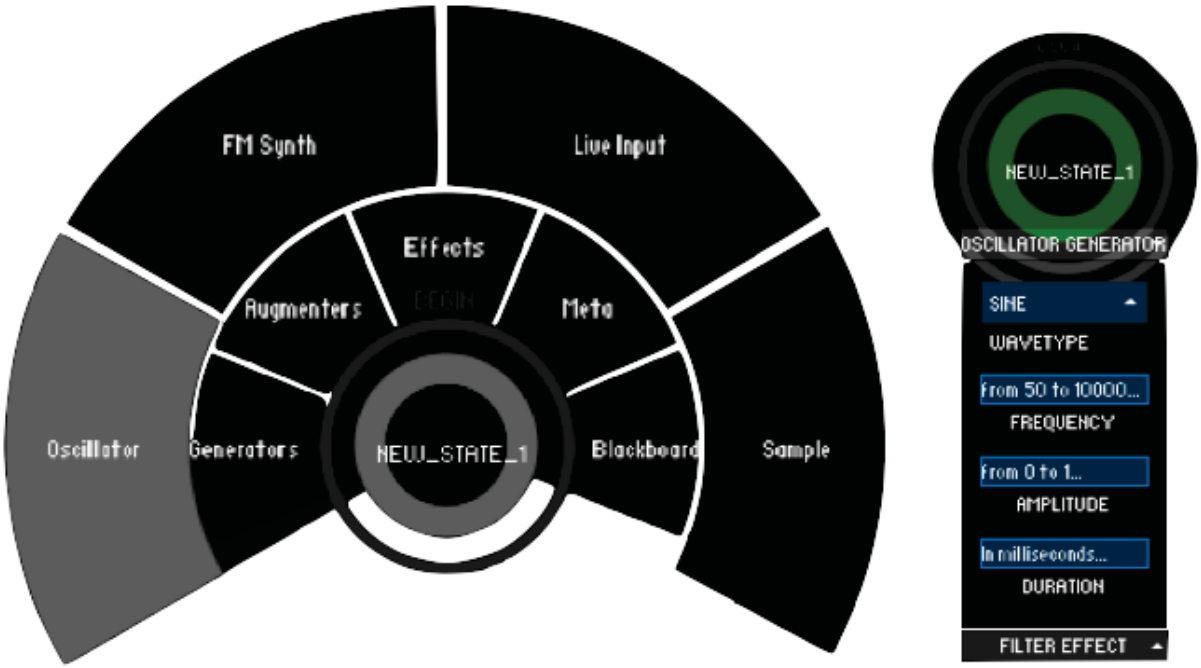


Fig. 6.2 Abstract programming elements such as states and tasks are visually reified in our interface, following DG3. The left image shows how tasks are available to users via contextual pie menu directly attached to states. The right image shows one example of a static musical tool created using only one state (i.e.‘`NEW_STATE_1`’) and two tasks (one oscillator task and a filter task). In this case, every note on the keyboard will result in a low-pass filtered sinusoidal oscillator. Frequency, amplitude, and duration are by default mapped to the notes played on the piano, but can be fine-tuned via high level parameters. Each task owns their own individual set of high-level parameter, each one detailing their possible input ranges. Here, the oscillator parameters are visible while the filter parameters are hidden.

6.5.2 The Blackboard

Examples like the one in Figure 6.2 are static, lacking interactivity, such as those built in DAWs. StateSynth allows users to bring interactivity to these static examples by using the *Blackboard*. The Blackboard is a repository of untyped global variables. These variables can be placed inside high-level parameters, resulting in richer interactive behaviors. One example is presented in Figure 6.3,

where we recreate using mouse X-Y coordinates the core principle of the theremin, an electronic musical instrument invented in early 20 century.

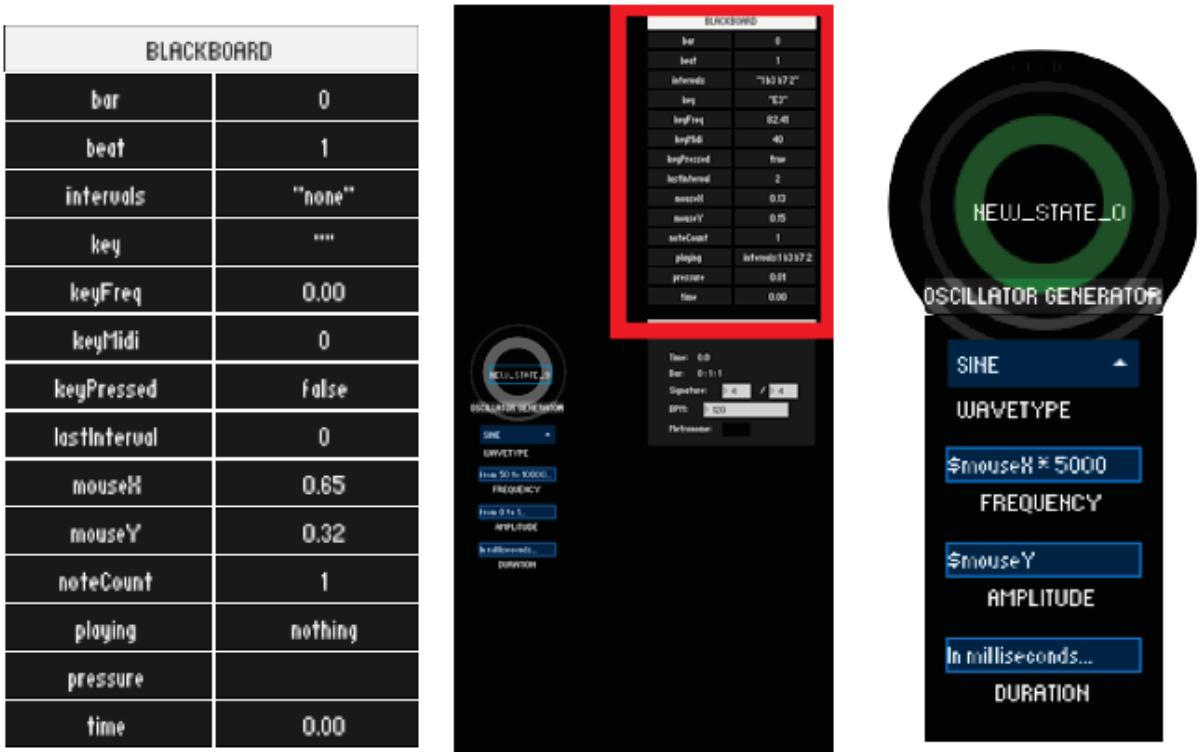


Fig. 6.3 Introduces the blackboard–image (a)–located on the top right of the interface–image (b). Image (c) shows an example of an interactive musical tool built with two blackboard variables–`'mouseX'` and `'mouseY'`–where we recreate the theremin. In the theremin, the player can use one hand to control amplitude of a sinewave, while the other hand control its frequency. Here, we have replaced hand position by mouse coordinates. `'mouseX'` has been multiplied by 5000 in order to fit the input frequency range.

Blackboard variables can be either *visible* or *hidden*. Visible blackboard variables mostly concern musical characteristics extracted at run-time from the keyboard playing (DG1), such as information on notes, chord, intervals, tempo, and pressure on the keys. These variables are reified on the top right of the interface (DG3), as seen on Figure 6.3. On the other hand, hidden variables mostly concern storing native input capabilities of the computer potentially useful for music expression (Fiebrink, Wang, and Cook 2007), such as mouse coordinates, or computer keyboard information. As the name implies, these variables are not directly displayed on the

interface. A full list of built-in Blackboard variables–both visible and hidden–is presented on Table 6.2.

In addition to these built-in blackboard variables, users can create their own blackboard variables via a specific category of tasks: the Blackboard-tasks. Examples include Low-frequency oscillators (LFOs), ramps, and random numbers. These tasks also have their own set of high level parameters–that can be cascaded with other blackboard variables, resulting in even more complex interactive behaviors. One example is presented in Figure 6.4, where the number of notes played is used to modulate the rate of a flanger.

Category	Name	Description
Visible	bar	*Current bar as shown in the tempo window.*
	beat	*Current beat as shown in the tempo window.*
	key	*Last note played on the musical keyboard.*
	keyPressed	*Shows if there is any key down on the musical keyboard.*
	mouseX	*X-axis position of the mouse cursor (from 0 to 1).*
	mouseY	*Y-axis position of the mouse cursor (from 0 to 1).*
	noteCount	*Current note count as shown in the tempo window.*
	playing	*Shows what is being currently played on the keyboard (e.g. a note? a chord? if so, which one?).*
	pressure	*Shows MIDI velocity information for the last key pressed.*
	time	*Shows how much time (in seconds) has passed since play button has been pressed.*
Hidden	cc1 to cc2	*Stores values of MIDI control changes (CC) messages.*
	chord	*Stores the last chord played on the musical keyboard.*
	interval	*Stores the last interval played on the musical keyboard.*
	keyReleased	*Shows if all keys are released on the musical keyboard.*
	minutes	*Shows how many minutes have passed since execution started (from 0 to 60).*
	mousePressed	*Shows if any mouse button is currently down.*
	note	*Stores the last note played on the musical keyboard.*
	numKeyPressed	*Counts how many keys are currently down on the musical keyboard.*
	pcKey	*Last key to be pressed on the computer keyboard (not the musical keyboard).*
	pcKeyPressed	*Shows if any key is down on the computer keyboard (not the musical keyboard).*

Table 6.2 List of all blackboard variables built-in to StateSynth, categorized as *visible* (that is, visually available on the user interface), and *hidden*.

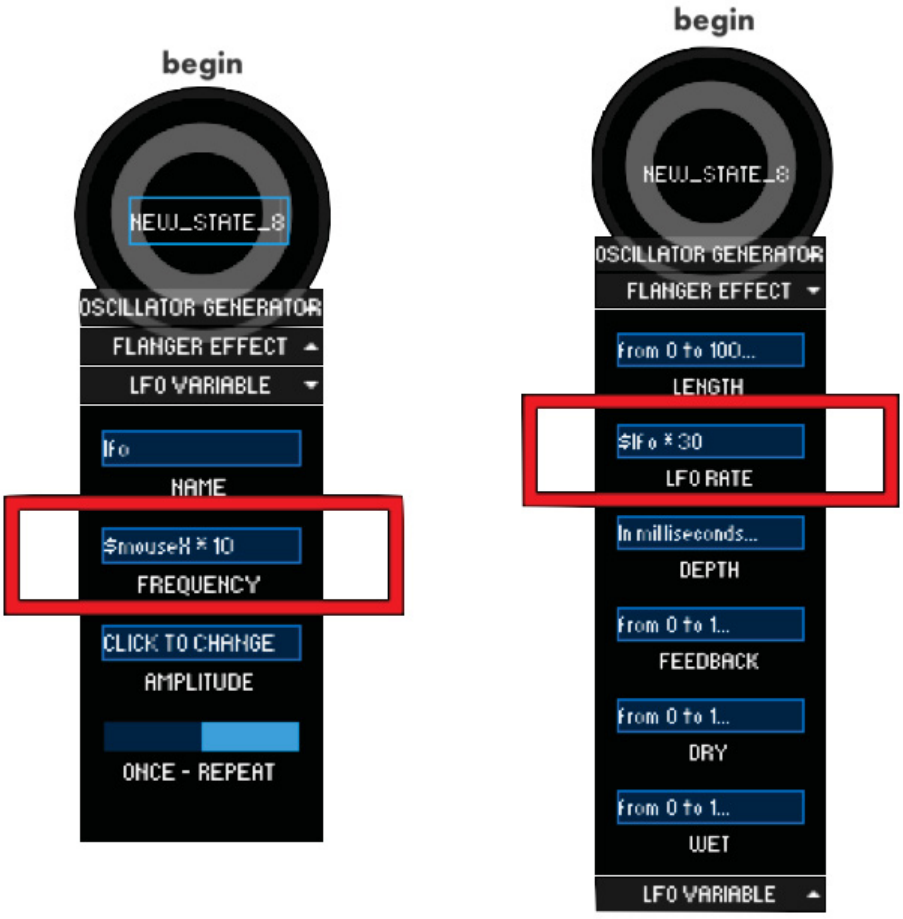


Fig. 6.4 Example of a more complex interactive musical tool built with two cascaded blackboard tasks: one built-in (i.e. `'numKeyPressed'`), and another user-created (i.e. A low-frequency oscillator task named `'lfo'`). In this case, the number of keys pressed feeds the frequency of a sinusoidal LFO (image (a)), which by its turn is used to control the `'LFO Rate'` parameter of a Flanger (image b). Therefore, as the number of keys pressed grows, the frequency of the LFO also grows, resulting in more rapid sinusoidal variation of the flanger.

6.5.3 Transitions

Examples presented so far cover only one state. However, StateSynth can support several states in the same state machine. One state can be connected to other states via *Transitions*.

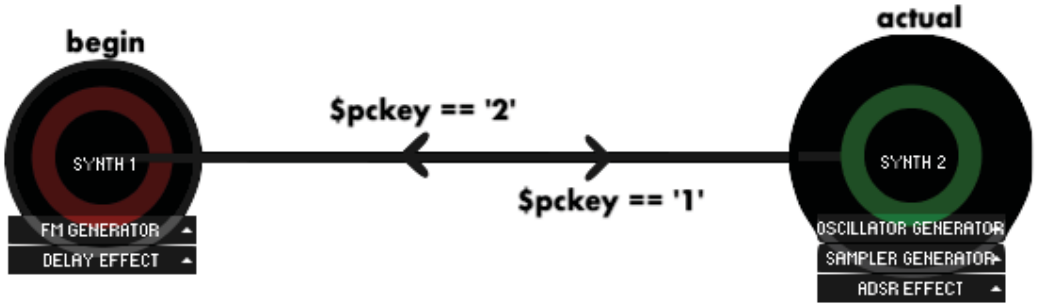


Fig. 6.5 Example of an interactive musical tool where users can switch between two different presets. When the user presses `'1'` on the computers' keyboard, presets described on state `'SYNTH 1'` are loaded. On opposition, presets of `'SYNTH 2'` state are loaded when `'2'` is pressed.

Transitions–also reified following DG3–concern a logical expression that, when met, leads to transition from one state to another. Expressions are written in javascript syntax and users have blackboard variables in order to create meaningful transitions. One simple example based on a hidden blackboard variables (`pcKey`) is presented in Figure 6.5. Furthermore, multiple variables can be used in a single condition, leading to more complex and subtle behaviors typical in expert instrumental practice–following DG1. An example concerns triggering transition by playing an specific chord (e.g. C major chord on the 5 octave) with a certain pressure on the keys, that could be represented by the expression `'chord == ''C5MAJ'' && pressure > 90'`. Another example concerns enabling different synthesizer voices according to the current beat and tempo, as presented in Figure 6.7.

6.5.4 Extensibility

StateSynth provides some features to extend the state-task-transition model. One example concerns the Open Sound Control (OSC) protocol[4], which is widely accepted in the context of mu-

[4] `http://opensoundcontrol.org/`

sic and media arts. StateSynth automatically adds any incoming OSC message directly to the blackboard, making them available to users. This functionality allows users to easily incorporate OSC-compatible external sensors such as the Leap Motion[5]. As an example, one could modify our StateSynth-theremin (described on Figure 6.3) to support actual hand control–as in a real theremin–instead of mouse-coordinates, as shown in Figure 6.6[6]. Other extensibility-related features are grouped in the category of meta-tasks, namely: OSC task, which allows StateSynth to send OSC messages to external media tools; the JavaScript task, that allow custom JavaScript code to be incorporated to a state; and State Machine tasks, that allows nesting–as exemplified in Figure 6.7.

Fig. 6.6 We can quickly modify our theremin example to make it more realistic (image (c)) in two steps. First, we use the Leap Motion sensor and an OSC converter software (image(a)). Second, we modify our previous example from Figure 6.3 to match image (b): that is, we replace `'mouseX'` and `'mouseY'` variables by the incoming-OSC blackboard variables (`'left_hand_z_0'` and `'right_hand_z_0'`) and add a note augmenter to eliminate the need of key pressings.

6.6 Evaluation

Aiming at investigating how StateSynth performs in terms of learnability thresholds and expressive ceilings, we have carried a longitudinal qualitative user study, borrowing the methodology from previous works (Jacobs, Brandt, et al. 2018; Jacobs, Gogia, et al. 2017), addressing:

[5] `https://www.leapmotion.com/`
[6] `https://www.youtube.com/watch?v=xDLBLcMUvmE`

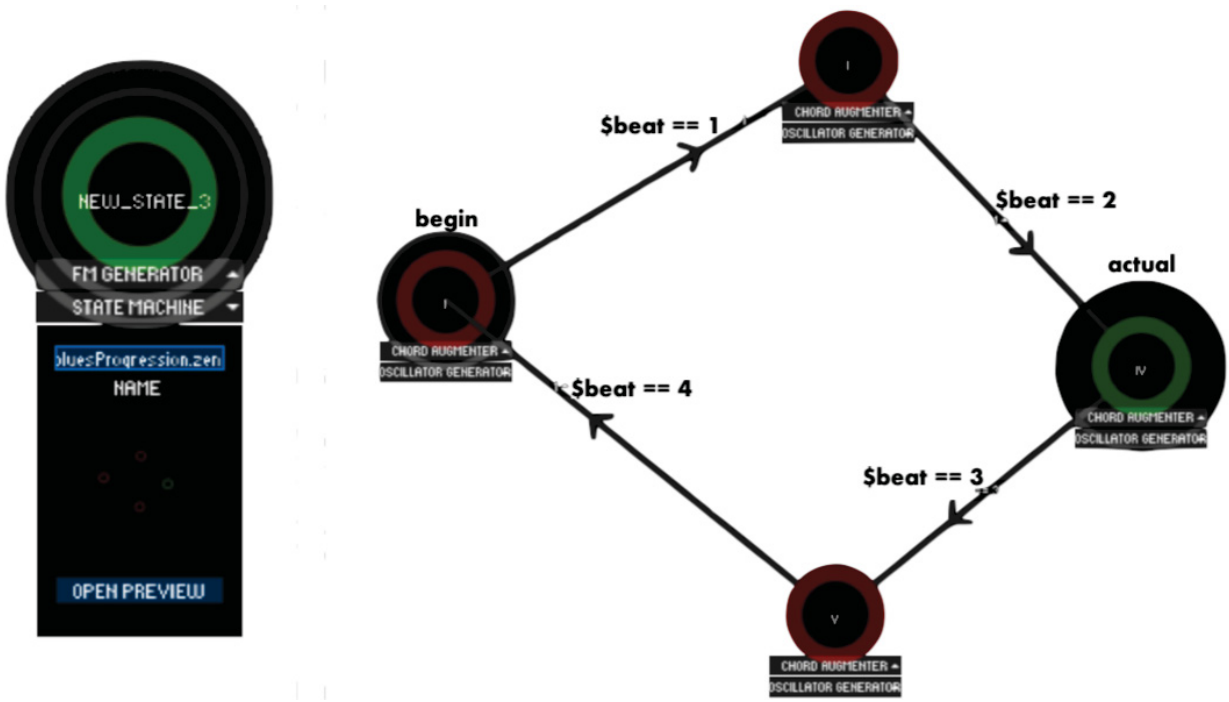


Fig. 6.7 Example of an interactive musical tool using nested state machines. In this case, a nested state-machine rhythmically sequences chords according to the beat (image(a)) while the parent state-machine addresses the melody with a FM synthesizer (image(b)). Users can navigate over the different levels of abstraction (i.e. Parent and nested) by zooming in and out.

Learning threshold: What level of use StateSynth provides to musicians? Did StateSynth allow these musicians to quickly get started on creative exploration without spending much time on training? How does it compare to other existing alternatives (e.g., Max MSP, Puredata)?

Expressive ceilings: What can be created using StateSynth? What is its expressive power? Did it enable different range of creative outcomes? Are musicians able to express their individual aesthetic identity within the system?

6.6.1 Procedure

We commissioned two professional musicians to create musical pieces using StateSynth. These artists were *experienced musicians* with *limited experience in computer programming*. Participants were free to create pieces in any musical style or aesthetics they preferred. Similarly, participants were allowed to use any other tools they prefers along with StateSynth (e.g., sequencers, synthesizers, or DAWs). Participants were given one week to complete this task. They received $75 for their collaboration.

Our procedure comprised (1) *Profiling*, (2) *Introductory tutorial*, (3) *Work sessions*, (4) *Meetings*, and (5) *Retrospective interview*. Each one is further described in the following subsections.

Profiling

Our profiling step comprises pre-screening and an profiling interview. As *pre-screening*, we sent potential participants a short questionnaire assessing their instrumental and programming expertise using a 5-points Likert scale, where 1 is newbie and 5 is expert. To meet our selection criteria, we pre-selected participants who self-rated themselves medium/high (i.e., 3-5) in instrumental expertise and low (i.e. 1-2) in programming skills. Pre-selected participants proceed to a *profiling interview*, further assessing participants' suitability to our study, as well as their past experiences with instrumental practice and programming tools.

The profile of the two selected participants is detailed in Table 6.3. P1 is a young sound artist recently-graduated in digital music, self-rated 3 for instrumental expertise and 2 for programming

expertise. P2 is 31-years old composer, PhD in Music composition, self-rated 5 for instrumental expertise and 2 for programming expertise.

Introductory tutorial

Second, we introduced StateSynth to participants via one 14-minutes video tutorial[7]. This video is the first part of a series of five video tutorials–total approximate duration, 1 hour and 10 minutes. The series details different features of our prototype, with practical examples of what can be built using it. Tutorials are organized incrementally so that the essential features are covered in the first video, and advanced complementary features are seen in the last video. At this moment, only the first video was introduced (the remaining were used later, c.f. section 'Meetings'). Furthermore, we made clear that the researcher is available anytime for technical support in case of questions, bugs, or struggles during development.

Work sessions, Session Logs, and Work diaries

After the tutorial, participants could start using StateSynth. They were asked to carry at least two working sessions of 1h each in different days. During these sessions, the system logged:

- **Usage Time (UT):** How much time has been spent using the system?
- **Playing Time (PlT):** How much time did the user spent playing the MIDI keyboard (i.e., measure the time there was a key down on the musical keyboard)?
- **Programming Time (PrT):** How much time did the user spent coding (i.e., total usage minus playing time)?
- **Number of sessions and crashes:** How many sessions did the user start? How many times has the system crashed?

[7] `https://www.youtube.com/watch?v=KFbYJglFdVA`

	Instrumental Music practice	**Computer programming**
P1	P1 is primarily a guitarist. P1 described his keyboard playing style as slow, mostly 'by ear'. In musical terms, P1 affirmed he often plays ambiance and long pads, often playing rhythmically 'with two fingers'. P1 often uses keyboards and synths to sketch out and develop musical ideas. He also plays these instruments professionally in an electronic music duo. P1 has first learned about synthesizer during his university studies, but he develops his skills on his own. He considers himself a 'geek' of software tools for synthesizing music and tries to keep updated with the newest technology. Software tools he commonly uses include Digital Audio Workstations, such as Ableton, Logic, Reaper, several audio plugins (i.e. VSTs), and iPad music apps.	P1 has took four music-related programming classes at university–namely, HTML, Puredata & Max/MSP, Pyo and C–, but he said he did not pursue programming after these courses (two years ago). Its code experience superficial, limited to small pieces (based on synths and sequencers for drony and ambient music), largely based on examples he found on the internet, with lots of trial and error. He often wouldn't dive into the code. When directly asked about negative programming experiences, P1 remembered problems in debugging sub-patches while keeping the 'big picture' in mind. P1 also mentioned having problems to go beyond the internet examples by himself. On the other hand, P1 mentioned that programming felt more intuitive and easy when using Pyo, as programming was 'linear' and easier to follow.
P2	P2 main instrument is the electric bass. While he can play relatively complex pieces in the keyboards, he is self-taught, has no conservatory technique, and has found his own idiosyncratic way of playing. P2 described his music as "intellectual" ("I always think too much when writing music"). He often composes using 'paper and computers', software synths and–sometimes–hardware interfaces. P2 often uses these software synths as a tool for experimenting with sound, to find the 'right sound'. P2 also plays the keyboard professionally on a pop band, and often uses the instrument to grade assignments in a university-level music course he teaches. In his academic studies, P2 investigated the use of synthesizers in classical concert music, having advanced expertise in these tools.	P2 took two programming courses at university and has read some books about computer programming when a child. However, he quit programming as it distracted him from the music itself, so that he prefers to "keep his intention and energy to music." His code experience is limited to small patches, such as, for example, triggering specific sound effects using a pedal. He said he has experimented with Max/Msp and Reaktor. When directly asked about negative programming experiences, he affirmed programming often involved "too much effort for little result" (i.e. "even simple things in music"–such as timers, LFOs, and arpeggiators–"were hard to create in programming"), and that existing tools often offered what he was looking after. Concerning positive intuitive experience with computer programming, he mentioned the case of Reaktor, where there was an existing "solid structure" that he could customize to fit his needs.

Table 6.3 Comparing profile info on P1 and P2.

- **UT, PLT, and PrT per session:** Average usage time, playing time, and programming time per session.
- **Explorability Index:** What is the percentage of features discovered by users?

JSON-formated logs and automatic screenshots (taken every 20 minutes) were saved locally, and handed during meetings (c.f. subsection 'Meetings'). The collected data is summarized in Table 6.4.

	UT	**PlT**	**PrT**	**NS**	**NC**	**UT/ses.**	**PlT/ses.**	**PrT/ses.**	**EI**
P1	04:36:19	01:15:16	03:21:02	19	7	00:14:32	00:03:57	00:10:34	92.50%
P2	03:26:10	00:48:45	02:37:24	11	6	00:18:44	00:04:25	00:14:18	75.00%

Table 6.4 Summarized logged data for each participant.

In addition, after each work session, participants were asked to subjectively document their experience in a work diary. This work diary comprises three open-ended questions: (1) How was your working session? What were you trying to achieve? (2) What have you tried today in creative terms? (3) Did you face problems today in learning and using the system? If so, which problems?

As workplace (Figure 6.8), participants were provided free access to an adequate room for musical practice–a soundproof room equipped with StateSynth prototype and a professional MIDI-enabled piano, the Yamaha CP300[8]. Participants had the freedom to choose when they wanted to work.

Meetings

After each work session, we scheduled individual 1-hour meetings to further inquire participants on their experience using StateSynth. The meeting started by reviewing work diaries together with the artist. Then, participants shared their progress, including as well technical questions, novel learnability issues and creative possibilities explored in the last days. Whenever possible, we stimulated comparisons to other tools familiar to the artist in order to provide us with a baseline

[8] `https://usa.yamaha.com/products/music_production/synthesizers/cp_series/cp300.html`

Fig. 6.8 Top image: the experimental setup, composed by a computer without prototype, a professional MIDI-enabled piano, and two monitor speakers. Bottom image: P1 and P2 during one of their work sessions.

for the results. Bugs and improvement suggestions were also taken into account. Finally, at each meeting, we provided participants with new video tutorials from our series based on their progress, encouraging participants to explore more advanced features.

We carried out two meetings for each participant–summarized in Table 6.5. These meetings were video recorded for analysis.

Retrospective meeting

After the week, participants finally performed the pieces created in StateSynth. The performances were video recorded. P1 presented a improvisation using one tool created during the second work session. P2 performed short excerpts with all the different tools he had developed throughout both work sessions. Excerpts of their performance can be found online[9]. By the end of this performance, we carried a retrospective semi-structured interview about their experience with our prototype.

6.6.2 Data analysis

We used qualitative content analysis for interviews (i.e. initial interview and retrospective interview), diaries (i.e. support diary and work diaries), and meetings. Quantitative session logs were analyzed using average. Results were triangulated in order to increase trustworthiness. The raw qualitative data is available on Appendix C, and a full report detailing our procedure and results is available online[10].

6.7 Results

In general terms, our study shows that both users (1) engaged in exploring the software (c.f., total usage time and high exploratory index in Table 6.4) (2) were able to create diverse musical results in the end and (3) reported they would like to continuing using the software in the future. In the next subsections, we discuss evidence related to StateSynth's expressive ceilings and learnability

[9] https://www.youtube.com/watch?v=RH5ERSNWKrg

[10] https://drive.google.com/open?id=1OduO2vqNwrBo7ShIHGMyG9-LN8JQrj0z

	First meeting	Second meeting	Final Performance
P1	P1 watched tutorial 1 and spent his work session playing around with the software (6.9). He affirmed he focused on getting to know the different tasks (e.g. Oscillators, FM synthesizer, sound effects), and looking for bugs. By the end of the session, P1 provided us with a list of small bugs he found in the software.	P1 skipped tutorial 2 (experimenter recommendation) and focused on tutorials 3 and 4. Afterwards, P1 spent a second work session playing around with the software, aiming at creating one small piece to present in the next section. P1 spent most of the session on one single file, improvising a lot with it.	P1 improvised a short beat-oriented piece, using a complex set of states and transitions–presented in Figure 6.10. He explored stacking sounds, combining different generators and effects at the same time. Different voices were triggered according to the tempo of the music. For the beats, he used several samples (mostly drum-related) and loops, exploring possibilities for 'gridded and non-gridded' beats. Regarding the latter, he explored augmenters with pitch–shifted samples for unusual out-of-the-grid beats combined with non-defaults tempo. He also explored some ambient samples in the background–enabled and disabled according to timers. Samples were downloaded from the internet during the session.
P2	P2 watched tutorial 1 and spent his work session playing around with the software (6.9). P2 affirmed he explored basic tasks and tried to be playing on the keyboard as much as on the computer.	P2 requested a high level summary all features provided by the system, and the experimenter present a summary of tutorials 3, 4 and 5. Afterwards, P2 went quickly through these all tutorials. P2 was really excited with the potential of the blackboard, in particular with how easy it was to use musical variables (e.g. intervals) inside tasks. P2 spent half an hour presenting creative possibilities as "his mind was flourishing in ideas". Later, on his second work session, P2 explored some of these possibilities, aiming at creating some small instruments he could perform with.	P2 reported the system allowed him to explore musical ideas he always want to test, but was never able to do in other system. In particular, P2 was excited the possibilities about using musical data as input–his favorite feature. As an example (Figure 6.10), P2 used musical intervals to control the FM synthesizer's amplitude and modulation ratio, and delay time–sometimes coupled with additional mouse control. He further affirmed that he would like to have more time to further explore these possibilities.

Table 6.5 Summarizing first meeting, second meetings, and final performance for each participant.

thresholds.

6.7.1 Expressive ceilings

First, participants repeatedly praised StateSynth expressive potential. For example, P1 described StateSynth as a *'experimental sketchpad'*, that allowed one to *'create something from scratch'* despite knowing only *'a basic set of knowledge from the program'*. *"There is already so much you can do with a few hours exploring the software"*, he affirmed. Similarly, P2 said that the system *"opened up a world of possibilities to explore, but that way to experiment with that was quite simple"*. When describing his musical results, P2 affirmed that it was *"pretty cool what he was able to create out of... nothing actually. I could do all that with that little time"*. P2 also affirmed he was able to explore aspects that he couldn't do easily with standard synthesizers (e.g. short duration of augmenters; easy use of mouse pad to control effects)–as shown in Figure 6.10. Furthermore, in their retrospective interviews, both participants reported unexplored creative possibilities they would have liked to try if they had more time (eg. P1 mentioned exploring blackboard variables, subpatches, javascript; P2 mentioned exploring intervals as input data, transitions and additional blackboard variables).

Second, musical results seem diverse and aesthetically aligned with the participants' individual style. This alignment can be noticed by contrasting participants' musical styles (Table 6.3) against their final performance, summarized in Table 6.5. The alignment can also be noticed in other moments during our study. For example, in his first meeting, P1 reported exploring ambiences, droning, short sequences, repeating notes, and using FM synth for rhythms (Figure 6.9)–as self-described in his musical style.

Finally, both participants reported they would be unable or hardly able to implement the same results in other music programming tools (i.e., it would likely require them more effort and time). As justification, P1 mentioned StateSynth visual approach (DG3) and StateSynth tasks-related limitations–a good thing in his perspective–as in other languages he would likely get lost given the numerous possibilities. Also, P2 highlighted how he was able to profit from his musical expertise

Fig. 6.9 Screenshots of the first work sessions of P1 and P2 (respectively).

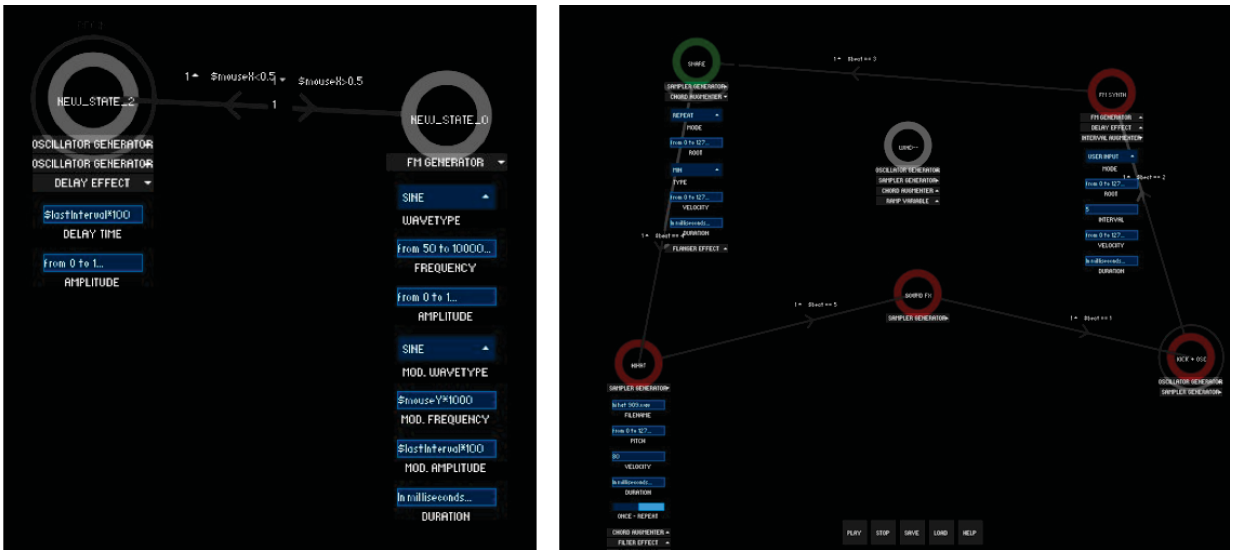


Fig. 6.10 Screenshots of the second work sessions of P2 and P1 (respectively).

inside the system (DG1). He affirmed: *"The keyboard still works, and of what I can do in other synths I can do here... I feel myself at home because these main characteristics are there. It's not a new instrument, it's just a way to complement what I can already do"*. This familiarity empowered P2 to explore the system as much as he was playing on the piano–even though he was in his first session.

6.7.2 Learnability thresholds

Participants repeatedly mentioned how easy it was to learn and use StateSynth. This idea seems aligned with the high exploratory index on Table 6.4. Despite its relative complexity in terms of states and transitions, P1 affirmed that his patch had been *'really easy to build'*. P1 believed he had explored most of the features from StateSynth and that he had learned most of the software, as they were easy to learn and to understand. Similarly, P2 reported multiple times that the system was simple to use and to learn, ready to be used used without the need of setups when compared to alternatives (e.g., no need to *"implement something for two hours before starting doing music"*). As a baseline for comparison, he explained that often he would read manuals to get started in software systems, but that with our system he was able to proceed directly to musical experimentation. He also highlighted multiple times how simple it was to try out new creative possibilities. *"There was no learning curve, I didn't have to program [basic stuff] if you want it to react... It's simple [laughs] I'm repeating this a lot but I think it's worth mentioning"*.

Similar evidence can be found throughout all sessions and work diaries. In his second meeting, P2 has repeatedly highlighted the system as *"really simple and easy to use"*–despites the complexity of the musical ideas he was exploring. *"It's just so easy that I don't understand why it already doesn't exist in other software systems. It just feels so easy to use"*. During his first session, P1 affirmed twice that he started to have fun really fast when using the system. *"It was nice to be able to begin stuff, and just to mess around with one parameter at a time, and there was already nice possibilities for experimental stuff"*. In his retrospective meeting, possibly related to DG3 P2 affirmed *"it's so easy, you do not have to map, for example, an hardware... it's so 'in your face"'*.

Finally, both participants described the experience of using SynthStates as a musical instrument, unlike the alternatives they had tried in the past. P1 described his interaction as *"tweaking a modular synthesizer"*, an improvisational tool rather than a rigid piece defined via code or musical score. P2 also explained: *"I would say it felt more like a musical instrument that these 'softwares'. In a sense that, the idea is to help you with the keyboard to play music"*. We believe that this finding resonates with our three guidelines, that empowered participants to keep their attention to the music, as if the system was a musical instrument.

6.8 Limitations

Our user study revealed limitations in our system. One example concerned *writing expressions*. Both participants reported problems in working with expressions–either inside high level parameters or inside transitions. P1 reported problems in finding the right expression representing his desires, and affirmed this hindered him from achieving more complex results. In the end, P1 avoided writing complex expressions, editing high level parameters in performance time. P2 also reported struggling defining one particular mathematical equation, that yielded system crashes. This issue points towards more user-friendly approaches to create expressions as future work, such as dragging-and-dropping ready-made expression templates (Jacobs, Brandt, et al. 2018).

Another issue concerned the *limited sound diversity and extensibility*. Participants either missed a larger variety of sound units and more high-level parameters for more subtle control. For example: P1 reported missing more parameters to play with inside the sampler; In his first work session, P2 reported that StateSynth's simplicity, while positive, also could translate to *"incomplete"*, referring that the system only covered *"simple sounds"*–our basic musical blocks; Both P1 and P2 missed the reverb as sound effect. This issue relates to the limited expressivity StateSynth has compared to other general purposes interactive music programming environments. Ideally, such authoring system would need to allow users to integrate user-defined tasks, going beyond the ones available inside the system. StateSynth does not support this feature. One potential solution, already used in other systems (Dannenberg and Kotcher 2010), concerns allowing tasks to

be written in general purpose computer music language such as Max, Puredata, or SuperCollider, and then easily incorporated in the system.

A third issue concerned the *interface usability*. Both participants reported interface-level usability problems. P1 reported frustration that sound was sometimes *"stopping and breaking"* during usage, mentioning audio clicking, dropping, and some features not working as he had expected (e.g. sampler). Both also mentioned issues with the reduced font size. Finally, P2 also complained about the fact he couldn't change the pipeline of tasks. It is interesting to note that despite these usability problems, both participants highlighted that the system was easy to use.

Finally, we also inherit limitations from our specification model, such as the blackboard detailed elsewhere (Barbosa, Wanderley, and Huot 2018). These issues need to be considered in future redesigns of StateSynth.

6.9 Summary and Perspectives

We introduced *instrumental music skill as input primitive* for creative authoring, articulated via three design guidelines–(1) *leveraging existing musical expertise*; (2) *lowering specification model complexity*; and (3) *increasing authoring environment directness*. These guidelines were explored on the *StateSynth:* a visual authoring environment that empowers musicians to build personal interactive musical tools, evaluated in a one-week study with two expert musicians.

Our results suggest that, while StateSynth's expressive ceilings are not as high as popular alternatives, e.g., DAWs or music programming languages, they are sophisticated enough to provide a wide diversity of *unique* creative possibilities. At the same time, we found that StateSynth's learnability thresholds seems much lower than these alternatives. Together, these findings indicate that StateSynth may provide a good balance between expressive ceilings and learnability thresholds–despites the significant limitations reported. Qualitative data also provide evidence that our guidelines were critical for succeeding at this balance, highlighting the potential of instrumental skills as input primitive for creative authoring. We hope our work may pave the way for future research on this topic, also contributing to open exciting new musical possibilities.

Chapter 7

Conclusion

This thesis investigated the *exploratory design and evaluation* of *software-based interfaces* based on *dynamic visual feedback* as a strategy to make *music interaction* more *direct and straightforward*.

As a first step (Chapter 2), I investigated the role of evaluation for musical interface design. More specifically, I investigated how the term "*evaluation*" has been employed in the NIME literature by analyzing all papers and posters published in the proceedings of the NIME conference between 2012 to 2014. The results provided us with a better idea of: a) the most common targets and stakeholders considered during the evaluation; b) the most common goals set; c) the most common criteria set; d) the most common techniques/methods used for the evaluation; and e) how long the evaluation lasts.

Following, I hypothesized that one effective strategy to foster directness concerns adopting an interaction model known as *direct manipulation*–characterized by a continuous visual representation of the object of interest, physical actions to manipulate the visual representations, rapid visual feedback, and incremental and easily reversible actions. To investigate this hypothesis, I presented exploratory case studies where direct manipulation was designed and evaluated in three different contexts of music tools.

- The first case study concerned *live looping tools* (Chapter 3), where I introduced a novel direct-manipulation based live looper tool called the **VRM**. Its design and formative eval-

uation uses an open-ended design rationale approach, comprising four steps: 1) Surveying and analyzing 101 existing live looping tools; 2) Building a design space from this analysis; 3) Exploring potential guidelines using our design space as baseline; and 4) Iteratively prototyping several low-fi and functional prototypes exploring our guidelines;

- The second case study regarded *tools for creating interactive artistic installations* (Chapter 4 and Chapter 5), where I introduced the **ZenStates**: a simple yet powerful visual model for interactive installations based on Hierarchical Finite-States Machines–discussed in Chapter 5. ZenStates design (Chapter 4) was iterative and user-centered, based on field studies, interviews, and iterative prototyping. Evaluation of the tool was four-folded: 1) implementing the model in a direct-manipulation based interface; 2) probing what ZenStates can create through 90 exploratory scenarios; and 3) performing a user study to investigate the understandability of ZenStates' model; and 4) a preliminary real-world application with professional artists. Results support ZenStates viability, its expressivity, its potential in a real-world setting, and suggest that ZenStates is easier to understand–in terms of decision time and decision accuracy–compared to two popular alternatives;

- The third and final case study addressed *music authoring tools* (Chapter 6), where I introduced another direct manipulation based music authoring tool called **StateSynth**. In StateSynth, music keyboard expertise becomes a core element for the programming, empowering technically-struggling musicians to build personal interactive music tools, by articulating rich interactive behaviors in terms of their embodied instrumental skills. Designed after ZenStates, I evaluated StateSynth's learning thresholds and expressive ceilings in a qualitative one week study with expert musicians. Results suggest that StateSynth accessibly allowed these musicians to use instrumental skill-driven authoring, enabling at the same time a wide diversity of musical results.

Finally, I conclude this thesis by presenting some considerations on direct manipulation for visual software-based musical interface design. These considerations comprise: 1) a set of interface

design strategies; 2) a set of evaluation approaches; and 3) potential limitations.

7.1 Interface Design Strategies

This thesis has explored eight interface strategies to support the implementation of DM proposed by Shneiderman. I stress that these strategies may be idiosyncratic and specific to the case studies addressed in this thesis. As such, they should not be taken as generalizable rules and should be approached with caution. These strategies are presented in subsections according to principles of direct manipulation.

7.1.1 Continuous representation of the object of interest

I) Conceptual model reification: This strategy concerns making visible abstract elements of the conceptual model used by the system. In the VRM, for example, I reify the basic conceptual model for live looping: audio sample recorded, displayed as a waveform; the head play position, the live microphone input. In ZenStates and StateSynth, all abstract concepts introduced in ZenStates specification model (i.e., states, tasks, transitions, and the blackboard) are equally reified in the interface, enabling direct manipulation and transparency. This strategy has been adopted by Jordà and colleagues in the Reactable (Jordà, Geiger, et al. 2007);

II) Visual feedback equals input control: In the VRM, thanks to the platform chosen (i.e., the iPad touchscreen), both visual feedback and input control take place in the same device, increasing the sense of directness (i.e., the user can finger touch the exact sample position he/she wants to be played). While this strategy has not been explored in the ZenStates or StateSynth, the example of the VRM–along with interfaces such as the Reactable (ibid.) and Ge Wang's visual crafts (2016)–suggest this is an interesting strategy for increasing directness;

7.1.2 Physical actions instead of complex syntax

III) Reducing the complexity of the conceptual model: Artistic responsive environments is a domain much more complex than live looping (e.g., these environments deal with diverse input and output technologies, whereas live looping only deals with audio), requiring much more sophisticated conceptual models. Therefore, it is expected that authoring tools such as Max/MSP cannot be visually represented as easily as the VRM. Despite this fact, the example of the ZenStates shows that it can be possible to reduce complexity in these domains by investigating new conceptual models. This strategy is particularly useful if the new conceptual model directly derives from knowledge inherent to the context (as in the ZenStates in Chapter 4). Reducing the complexity of the conceptual model seems an effective strategy to make the tool potentially easier to use, and arguably more direct. Similar discussions have been carried in terms of interface control metaphors for music (Fels, Gadd, and Mulder 2002; Wessel and Wright 2002);

IV) Centering the interaction on an existing musical instrument: Here, the StateSynths shows an interesting example where a significant part of the interaction happens on the keyboards so that there is no need for setups (like a modular synthesizer, also aligned with Cook's principle "Instant Music, Subtly Later" (Cook 2001));

V) Lowering input capacity: Providing a lower input capacity means reducing the number of standard input controls immediately visible to users for the interaction. The VRM only provided the iPad screen and one button for recording. ZenStates provide four buttons (i.e., play, stop, save, and load), two hotkeys (i.e., '+' and '-'), mouse, and keyboard for typing. The need for the keyboard is a sensitive point for improvement, as it could be reduced by enabling drag-and-drop of blackboard variables. This strategy is aligned with systems such as Spiegel's Music Mouse (1986), Jordà's FMOL (2002), and Levin's AVES (2000), which all opted for the simplicity of the mouse as an input device;

7.1.3 Rapid, incremental, reversible operations

VI) Direct mappings: In the VRM, every single functionality–both basic and advanced–was directly accessible via input controls. That is, each control input is continuously mapped to one sound-related feature and its visual correlated. There is no such input control whose action cannot be immediately perceived in terms of sound and visual feedback. While this has not always been possible on ZenStates/StateSynth, I tried whenever possible to follow this strategy;

VII) Contextual input overload: To compensate for the low-input capacity, I made judicious use of contextual input controls overloads. That is, the same input control could have different direct mappings depending on the context. For example, overloading one main button for play, record, overdub, and stop functionalities is a common practice in live looping. Another example in the VRM concerns the increasing fingers, which can result in either defining the actual position of the 'playhead' or on defining a looping area.

For the ZenStates and StateSynth, functionalities are available by overloading their input controls according to the context. For example, pressing '+' when the mouse hovers an empty canvas area will add a new state (the only action possible). However, pressing '+' when the mouse hovers a state shows a contextual menu for the tasks (to be attached to this state). The hotkey '-' translates to removal of the object hovered by the mouse.

7.1.4 Impact on the object of interest is immediately visible

VIII) Liveness and animations: User manipulations are also visually reified in the interface. These reifications often happen through visual animations. In the VRM, for example, the waveform visually reduces its size as the user reduces volume; Conversely, the waveform dynamically increases its size as the user increases the volume. Another example concerns highlighting the currently played looping area, by graying out the area which is not currently playing.

In the ZenStates, users actions yield an immediate result in physical changes in the responsive environment. Moreover, especially for the cases that do not have physical correspondence in the responsive environment, visual feedback is provided using animations as much as possible. As a result, users and the audience may infer what is going on inside the device (i.e., visual feedback should be transparent).

Compared to ZenStates, StateSynth incrementally improves authoring liveness, as there is no 'start' and 'stop': Like in the Reactable (Jordà, Geiger, et al. 2007), the system is always running, and there is no edit mode.

7.2 Evaluation Approaches

This thesis has explored two approaches to evaluate musical interfaces based on visual software.

In the one hand, I explore the *design process as a formative evaluation to wicked open-ended problems* (Dahl 2016). More specifically, Chapter 3 explores playfulness in a design-rationale based approach and how the VRM results from this process, whereas Chapter 4 presents a formative design study on behavioral show-control and how the ZenStates is built from this process. I hope these examples can contribute to the debate of contextualizing NIME research as research through design, by showing how rich and detailed design processes can potentially work as formative evaluation.

On the other hand, I also explore *empirical studies to evaluate specific easily-verifiable claims about my artifacts* (i.e., summative evaluation). Chapter 5 provided a quantitative study where alternative specification models are compared in terms of their understandability, whereas Chapter 6 provided a longitudinal qualitative study investigating StateSynth's learnability thresholds and expressive ceilings. I hope these examples can contribute to diversifying the range of quantitative and qualitative-based NIME evaluation strategies.

As discussed in Chapter 2, the two approaches described here should not be taken as one-fits-all solutions: No evaluation methodology is perfect, nor complete: each one fits better a certain

range of research problems–especially in a diverse research context such as NIME (Gurevich 2016). For example, our design approach is arguably unsuited to a context where specific easily-verifiable claims are made (e.g., Chapter 5). Similarly, our quantitative approach can be unsuited to some contexts as simple atomic tasks hardly represent the whole musical experience, or because finding a high number of participants with a specific profile may be daunting. Finally, longitudinal studies such as ours may be unsuited to some contexts as they are more expensive to perform, or because they require more robust technical solutions in order to not bias results in the long term (i.e., technical problems can keep users away from exploring the instrument)–which may not be the case of early prototypes.

7.3 Limitations and Criticisms

Finally, I present some limitations of direct manipulation for music interface design. Some general limitations have been presented in previous works (Shneiderman et al. 2010). Here, I focus on those that might relate to music interface design.

One natural criticism concerns visually impaired users. Within music practice, several blind musicians show that the feeling of directness does not require visuals, and is ultimately achieved in music. Still, as demonstrated in this thesis, we point out that visuals may be helpful in several cases.

Another recurrent criticism concerns the display space required by visual representations, that might be unsuited to complex problems better represented via concise textual counterparts.

Abstract visual representations might also keep users ignorant about the code and internal mechanics of computers that might be undesirable in some cases.

Moreover, as Ge Wang argues (2016), sound-to-visuals mappings will always involve idiosyncratic aesthetics decisions. Designing visual aesthetics is hard, an art form on itself, with unique history and traditions. While some literature on visual aesthetics literacy (Sutcliffe 2009) and the guidelines here-presented might be helpful in a techno-functional perspective, these will never be sufficient for an aesthetically well-accomplished visual craft.

I also highlight that directness may be a relative concept, depending on several factors such as task and user expertise, that may change over time. Hutchins and colleagues (1985) make this point clearer by introducing a musical example:

> *"Take the task of producing a middle-C note on two musical instruments, a piano and a violin. For this simple task, the piano provides the more direct interface because all one need do is find the key for middle-C and depress it, whereas on the violin, one must place the bow on the G string, place a choice of fingers in precisely the right location on that string, and draw the bow. A piano's keyboard is more semantically direct than the violin's strings and bow for the simple task of producing notes. The piano has a single well-defined vocabulary item for each of the notes within its range, while the violin has an infinity of vocabulary items, many of which do not produce proper notes at all. However, when the task is playing a musical piece well rather than simply producing notes, the directness of the interfaces can change. In this case, one might complain that a piano has a very indirect interface because it is a machine with which the performer "throws hammers at strings." The performer has no direct contact with the components that actually produce the sound, and so the production of desired nuances in sound is more difficult. Here, as musical virtuosity develops, the task that is to be accomplished also changes from just the production of notes to concern for how to control more subtle characteristics of the sounds like vibrato, the slight changes in pitch used to add expressiveness. For this task the violin provides a semantically more direct interface than the piano".*

A similar argument is made by Pierre Dragicevic, who argues for the DIMP (presented in the Introduction, subsection "Where are we now?") over standard WIMP-based video controls:

> *"On computer screens, where everything is just pixels, you are never physically manipulating objects. Direct manipulation is always an illusion produced by having user's gestures match the resulting motions on the screen as closely as possible. (...) So what's*

> *being manipulated, exactly? Both the video content (i.e., the things you see moving in the video) and the 'tape head'. When using DimP, the user directly manipulates the video content and indirectly manipulates the tape head. When using the seeker bar, the user directly manipulates the tape head and indirectly manipulates the video content".*[1]

All these points need to be considered towards more effective use of direct manipulation on visually-oriented music interface design.

[1] `http://dragice.fr/dimp/#directmanipulation`

Appendix A

Surveying Live Looping Tools

Table A.1: Survey and analysis of the 101 live looping tools discussed in Chapter 3.

Name	Picture	Video	Type	Visual feedback	Input	Interaction	Website
Ableton Live's looper	Link	Youtube	Virtual	Computer Screen	2 knobs; 5 drop-down lists; 11 buttons;	GUI Software	Link
Akai E2 HeadRush	Link	Youtube	Physical	LEDs	2 foot switch; 6 knobs; 2 switches; 1 button;	Foot pedal	Link
Ambiloop	Link	Youtube	Virtual	Computer Screen	–	GUI Software	Link
Angstro Looper	Link	Youtube	Virtual	Computer Screen	–	GUI Software	Link
Augustus Loop	Link	Youtube	Virtual	Computer Screen	–	GUI Software	Link
Boomerang III Phrase Sampler	Link	Youtube	Physical	LEDs	5 foot switchs; 6 knobs;	Foot pedal;	Link
Boss DD-20 Giga Delay	Link	Youtube	Physical	LEDs; LED display;	2 foot switches; 4 buttons; 5 knobs	Foot pedal	Link
Boss DD-500	Link	Youtube	Physical	LEDs; LCD display;	3 foot switches; 4 buttons; 6 knobs	Foot pedal	Link
Boss RC-1	Link	Youtube	Physical	LEDs;	1 foot switch; 1 knob	Foot pedal	Link
Boss RC-20X	Link	Youtube	Physical	LEDs;	2 foot switches; 6 buttons; 5 knobs	Foot pedal	Link
Boss RC-3	Link	Youtube	Physical	LEDs; LED display;	1 foot switch; 1 knobs	Foot pedal	Link
Boss RC-30	Link	Youtube	Physical	LEDs; LED display;	2 foot switchs; 2 knobs; 11 buttons; 2 sliders;	Foot pedal	Link
Boss RC-300	Link	Youtube	Physical	LEDs; LED display;	8 foot switchs; 1 expression pedal; 16 knobs; 13 buttons; 3 sliders	Foot pedal;	Link
Boss RC-505	Link	Youtube	Physical	LEDs; LED display;	32 buttons; 6 knobs; 5 slides;	Tangible	Link

Boss VE-20	Link	Youtube	Physical	LEDs; LCD display;	2 foot switches; 6 buttons; 1 knob	Foot pedal	Link
Boss VE-5	Link	Youtube	Physical	LEDs; LCD display;	11 buttons; 1 knob	Tangible	Link
Cube 120-XL Bass	Link	Youtube	Physical	LED;	4 buttons; 12 knobs	Tangible	Link
Cube 40-XL Amplifier	Link	Youtube	Physical	LED;	4 buttons; 12 knobs	Tangible	Link
Cube 60-XL Bass	Link	Youtube	Physical	LED;	4 buttons; 12 knobs	Tangible	Link
Cube 80 Amplifier	Link	Youtube	Physical	LED;	4 buttons; 12 knobs	Tangible	Link
Cube 80-XL Amplifier	Link	Youtube	Physical	LED;	4 buttons; 12 knobs	Tangible	Link
Digitech DL-8	Link	Youtube	Physical	LED;	1 foot switch; 4 knobs	Foot pedal	Link
Digitech Harman GNX4	Link	Youtube	Physical	LEDs; 3 LED displays	8 foot switches; 42 buttons; 6 knobs	Foot pedal; Tangible	Link
Digitech JamMan Delay	Link	Youtube	Physical	LEDs; LED display;	8 foot switch; 8 buttons; 8 knobs	Foot pedal	Link
Digitech JamMan Express XT	Link	Youtube	Physical	LED;	1 foot switch; 1 knob	Foot pedal	Link
Digitech JamMan Solo XT	Link	Youtube	Physical	LEDs; LED display;	1 foot switch; 5 buttons; 2 knobs	Foot pedal	Link
Digitech JamMan Stereo	Link	Youtube	Physical	LEDs; LED display;	4 foot switchs; 5 knobs; 6 buttons	Foot pedal	Link
Digitech JamMan Vocal XT	Link	Youtube	Physical	LED;	1 foot switch; 1 knob	Foot pedal	Link

Digitech Live FX	Link	Youtube	Physical	LED; LCD display	3 foot switches; 14 buttons; 1 knob	Foot pedal; Tangible	Link
Digitech RP 1000	Link	Youtube	Physical	LEDs; 2 LED displays	14 foot switches; 1 expression pedal; 6 buttons; 6 knobs	Foot pedal	Link
Digitech RP 500	Link	Youtube	Physical	LEDs; 2 LED displays	9 foot switches; 1 expression pedal; 5 buttons; 6 knobs	Foot pedal	Link
Digitech RP 360	Link	Youtube	Physical	LED; LCD displays	3 foot switches; 23 buttons	Foot pedal	Link
Digitech RP 360 XP	Link	Youtube	Physical	LED; LCD displays	3 foot switches; 1 expression pedal; 23 buttons;	Foot pedal	Link
Digitech Trio	Link	Youtube	Physical	LEDs	1 foot switch; 5 knobs;	Foot pedal	Link
DL4 Looper	Link	Youtube	Physical	LEDs	4 foot switchs; 6 knobs	Foot pedal	Link
Drille	Link	Vimeo	Mixed	Video projection	Piivert; VR;	Experimental	Link
Echoloop VST	Link	Youtube	Virtual	Computer Screen	–	GUI Software	Link
Eclipse V4	Link	Youtube	Physical	LEDs; LCD display;	26 buttons; 1 knob	Tangible	Link
EH Stereo Memory Man with Hazarai	Link	Youtube	Phisical	LEDs;	2 foot switches; 5 knobs; 1 button	Foot pedal	Link
EHX 22500	Link	Youtube	Physical	LEDs; LCD display	3 foot switches; 7 buttons; 3 knobs	Foot pedal	Link
EHX 45000	Link	Youtube	Physical	LEDs	9 knobs; 11 buttons; 7 sliders	Tangible	Link
EHX Nano Looper 360	Link	Youtube	Physical	LEDs	1 foot switch; 2 knobs	Foot pedal	Link
Electrix Repeater	Link	Vimeo	Physical	LEDs; LED display	20 buttons; 4 knobs; 4 sliders	Tangible	Link

Elektron Octatrack	Link	Youtube	Physical	LEDs; LCD display	56 buttons; 1 knob; 1 slider	Tangible	Link
Eventide H9	Link	Youtube	Physical	LEDs; LED display	2 foot switches; 4 buttons; 1 knob	Foot pedal; Tangible	Link
EveryDay Looper	Link	Youtube	Virtual	Tablet screen	–	Mobile based	Link
Firehawk FX	Link	Youtube	Mixed	LEDs; LED display; Tablet screen;	12 foot switches; 1 expression pedal; 1 button; 7 knobs	Foot pedal; Mobile	Link
FP-7F Digital Piano	Link	Youtube	Physical	LED; LCD display	keyboard; 26 buttons; 2 knobs	Experimental	Link
Freewheeling	Link	AVI File	Virtual	Computer Screen	–	Experimental	Link
Gibson Echoplex Digital Pro	Link	Youtube	Physical	LEDs; LED display	7 foot switchs; 9 buttons; 4 knobs	Foot pedal; Tangible	Link
Illusio	Link	Vimeo	Mixed	Multitouch screen	3 foot switches; multitouch screen	Experimental	Link
JM4 Looper	Link	Youtube	Physical	LEDs; LED display;	4 foot switchs; 10 knobs; 6 buttons; 1 directional controller (4 buttons)	Foot pedal	Link
Kaoss Pad KP3 Plus	Link	Youtube	Physical	LED Matrix; LEDs; LED display	1 Touch matrix; 20 buttons; 5 knobs; 1 slider; 1 switch	Tangible	Link
Kaoss Pad Quad	Link	Youtube	Physical	LED Matrix; LEDs; LED display	1 Touch matrix; 25 buttons; 5 knobs;	Tangible	Link
Lexicon PSP 42 VST	Link	Youtube	Virtual	Computer Screen	–	GUI Software	Link
Line 6 Spider Jam	Link	Youtube	Physical	LEDs; LED display	11 buttons; 12 knobs	Tangible	Link
Line6 POD HD500	Link	Youtube	Physical	LEDs; LCD display	12 foot switches; 1 expression pedal; 13 knobs	Foot pedal	Link

Line6 POD X3 Live	Link	Youtube	Physical	LEDs; LCD display	12 foot switches; 1 expression pedal	Foot pedal	Link
LiSa	Link	None	Virtual	Computer screen	–	Experimental	Link
Livid Looper	Link	Youtube	Virtual	Computer screen	–	GUI Software	Link
Logelloop	Link	Youtube	Virtual	Computer screen	–	GUI Software	Link
Looperlative mini-looper	Link	Youtube	Physical	LED	4 foot switchs	Foot pedal	Link
Loopr	Link	Youtube	Virtual	Tablet screen	Tablet screen	Mobile	Link
LoopStack	Link	Youtube	Virtual	Tablet screen	Tablet screen	Mobile	Link
Loopy HD	Link	Youtube	Virtual	Tablet screen	Tablet screen	Mobile	Link
Loopy Llama	Link	Youtube	Virtual	Computer screen	–	GUI Software	Link
M13 Stompbox	Link	Youtube	Physical	LEDs; 4 LCD displays;	15 foot switchs; 24 knobs	Foot pedal	Link
M9 Stompbox	Link	Youtube	Physical	LEDs; LCD display	7 foot switches; 6 knobs	Foot pedal	Link
MC-09	Link	Youtube	Physical	LEDs; LED display	51 buttons; 8 knobs; 4 sliders	Tangible	Link
mmTsss	Link	Youtube	Virtual	Computer screen	–	GUI Software	Link
Mobius	Link	Youtube	Virtual	Computer screen	5 menu; 14 buttons; 13 knobs	GUI Software	Link
OCTAPAD SPD-30	Link	Youtube	Physical	LEDs; LCD display	20 buttons; 2 knobs; 8 pads	Experimental	Link
Peavey Sanpera I	Link	Youtube	Physical	LEDs; LED display	5 foot switches	Foot pedal	Link

Peavey Sanpera II	Link	Youtube	Physical	LEDs; LED display	12 foot switches	Foot pedal	Link
Peavey Sanpera Pro	Link	Youtube	Physical	LEDs; LED display	18 foot switches	Foot pedal	Link
Pigtronics Infinity	Link	Youtube	Physical	LEDs; LED display	3 foot switches; 4 knobs; 6 buttons	Foot pedal	Link
Radial	Link	Youtube	Virtual	Computer screen	–	GUI Software	Link
Repetito	Link	Youtube	Virtual	Computer screen	–	GUI Software	Link
RiffBox	Link	Vimeo	Physical	LEDs; LED display	1 foot switch; 2 knobs; 2 discrete sliders	Foot pedal	Link
RiffBox App	Link	None	Virtual	Tablet screen	Tablet screen	Mobile	Link
SooperLooper	Link	Youtube	Virtual	Computer screen	–	GUI Software	Link
TC Ditto Looper	Link	Youtube	Physical	LED	1 foot switch; 1 knob	Foot pedal	Link
TC Ditto Looper Gold	Link	Youtube	Physical	LED	1 foot switch; 1 knob	Foot pedal	Link
TC Ditto Mic Looper	Link	Youtube	Physical	LED	2 foot switches; 1 knob	Foot pedal	Link
TC Ditto Stereo Looper	Link	Youtube	Physical	LED	1 foot switch; 1 knob	Foot pedal	Link
TC Ditto X2 Looper	Link	Youtube	Physical	LED	2 foot switches; 1 knob	Foot pedal	Link
TC Flashback delay	Link	Youtube	Physical	LED	1 foot switch; 4 knobs	Foot pedal	Link
TC Flashback X4 delay	Link	Youtube	Physical	LED	4 foot switch; 5 knobs	Foot pedal	Link
TimeFactor	Link	Youtube	Physical	LEDs; LED display	3 foot switches; 10 knobs; 1 button	Foot pedal	Link

TimeLine	Link	Youtube	Physical	LEDs	2 foot switches; 5 knobs	Foot pedal	Link
VoiceJam	Link	Youtube	Virtual	Tablet screen	Tablet screen	Mobile	Link
VoiceLive Touch	Link	Youtube	Physical	LEDs; LED display	2 knobs; 24 touch buttons	Tangible	Link
VoiceLive Touch 2	Link	Youtube	Physical	LEDs; LCD display	1 knobs; 21 touch buttons; 1 touch slider	Tangible	Link
Vox DelayLab	Link	Youtube	Physical	LEDs; LED display	4 foot switches; 6 knobs; 6 buttons	Foot pedal	Link
Vox Dynamic Looper VDL-1	Link	Youtube	Physical	LEDs; LED display	4 foot switches; 1 expression pedal; 3 knobs; 9 buttons	Foot pedal	Link
Vox Lil'Looper	Link	Youtube	Physical	LEDs	2 foot switches; 5 buttons; 2 knobs	Foot pedal	Link
Zoom G1on	Link	Youtube	Physical	LCD display	2 foot switchs; 8 buttons; 1 knob	Foot pedal	Link
Zoom G1Xon	Link	Youtube	Physical	LCD display	3 foot switchs; 8 buttons; 1 knob; 1 expression pedal	Foot pedal	Link
Zoom G2 1u	Link	Youtube	Physical	LEDs; LED display	3 foot switchs; 6 buttons; 4 knobs; 1 expression pedal	Foot pedal	Link
Zoom G3	Link	Youtube	Physical	LEDs; 3 LCD displays	3 foot switchs; 16 buttons; 9 knobs	Foot pedal	Link
Zoom G3X	Link	Youtube	Physical	LEDs; 3 LCD displays	4 foot switchs; 16 buttons; 9 knobs; 1 expression pedal	Foot pedal	Link
Zoom G5	Link	Youtube	Physical	LEDs; 4 LCD displays	6 foot switchs; 21 buttons; 14 knobs; 1 expression pedal	Foot pedal	Link

Appendix B

ZenStates User Study

B.1 Raw data

Table B.1: Raw collected data from the user study discussed in Chapter 5.

	id	age	exp.	gender	pref.lang.	order	trial	block	answ.	right.answ.	video.time	duration.time	easier	harder
1	p2	22	8	Male	C++	BCA	1	pd	8	8	118.20	346.03	pde	pd
2	p2	22	8	Male	C++	BCA	2	pd	13	13	77.63	157.71	pde	pd
3	p2	22	8	Male	C++	BCA	3	pd	16	5	110.01	224.85	pde	pd
4	p2	22	8	Male	C++	BCA	4	pd	6	6	82.27	186.44	pde	pd
5	p2	22	8	Male	C++	BCA	5	pd	5	16	116.83	300.41	pde	pd
6	p2	22	8	Male	C++	BCA	6	pd	3	3	34.61	143.21	pde	pd
7	p2	22	8	Male	C++	BCA	1	pde	13	13	24.91	40.65	pde	pd
8	p2	22	8	Male	C++	BCA	2	pde	7	7	67.87	202.45	pde	pd
9	p2	22	8	Male	C++	BCA	3	pde	8	8	66.42	196.29	pde	pd
10	p2	22	8	Male	C++	BCA	4	pde	6	6	38.63	167.00	pde	pd
11	p2	22	8	Male	C++	BCA	5	pde	16	16	87.58	174.07	pde	pd
12	p2	22	8	Male	C++	BCA	6	pde	20	20	24.59	108.20	pde	pd
13	p2	22	8	Male	C++	BCA	1	zen	6	6	0.00	15.15	pde	pd
14	p2	22	8	Male	C++	BCA	2	zen	13	7	9.81	42.41	pde	pd
15	p2	22	8	Male	C++	BCA	3	zen	8	5	17.58	74.86	pde	pd
16	p2	22	8	Male	C++	BCA	4	zen	20	20	21.86	118.12	pde	pd
17	p2	22	8	Male	C++	BCA	5	zen	13	13	6.82	35.53	pde	pd
18	p2	22	8	Male	C++	BCA	6	zen	13	3	7.51	16.73	pde	pd
19	p11	44	3	Female	Processing	ABC	1	zen	1	1	52.88	85.93	zen	pd
20	p11	44	3	Female	Processing	ABC	2	zen	12	12	128.69	166.84	zen	pd
21	p11	44	3	Female	Processing	ABC	3	zen	10	10	98.95	154.29	zen	pd
22	p11	44	3	Female	Processing	ABC	4	zen	7	15	93.08	178.58	zen	pd
23	p11	44	3	Female	Processing	ABC	5	zen	22	22	86.15	135.99	zen	pd
24	p11	44	3	Female	Processing	ABC	6	zen	7	7	84.20	137.10	zen	pd
25	p11	44	3	Female	Processing	ABC	1	pd	15	15	152.90	246.79	zen	pd
26	p11	44	3	Female	Processing	ABC	2	pd	3	3	46.30	81.74	zen	pd
27	p11	44	3	Female	Processing	ABC	3	pd	7	10	75.81	116.51	zen	pd

28	p11	44	3	Female	Processing	ABC	4	pd	1	1	36.06	88.38	zen	pd
29	p11	44	3	Female	Processing	ABC	5	pd	7	7	66.44	140.37	zen	pd
30	p11	44	3	Female	Processing	ABC	6	pd	9	12	148.18	198.85	zen	pd
31	p11	44	3	Female	Processing	ABC	1	pde	12	12	61.88	110.40	zen	pd
32	p11	44	3	Female	Processing	ABC	2	pde	3	3	39.18	76.97	zen	pd
33	p11	44	3	Female	Processing	ABC	3	pde	10	10	64.91	175.09	zen	pd
34	p11	44	3	Female	Processing	ABC	4	pde	22	1	58.05	123.41	zen	pd
35	p11	44	3	Female	Processing	ABC	5	pde	7	7	72.66	135.16	zen	pd
36	p11	44	3	Female	Processing	ABC	6	pde	12	15	118.04	273.90	zen	pd
37	p12	46	8	Other	Max/MSP	ACB	1	zen	2	2	29.81	75.01	zen	pd
38	p12	46	8	Other	Max/MSP	ACB	2	zen	14	14	28.39	53.96	zen	pd
39	p12	46	8	Other	Max/MSP	ACB	3	zen	6	6	30.67	61.22	zen	pd
40	p12	46	8	Other	Max/MSP	ACB	4	zen	8	8	183.20	312.45	zen	pd
41	p12	46	8	Other	Max/MSP	ACB	5	zen	3	3	75.99	162.16	zen	pd
42	p12	46	8	Other	Max/MSP	ACB	6	zen	5	5	74.94	169.01	zen	pd
43	p12	46	8	Other	Max/MSP	ACB	1	pde	14	6	43.81	119.71	zen	pd
44	p12	46	8	Other	Max/MSP	ACB	2	pde	8	8	160.52	388.74	zen	pd
45	p12	46	8	Other	Max/MSP	ACB	3	pde	1	1	22.07	162.30	zen	pd
46	p12	46	8	Other	Max/MSP	ACB	4	pde	5	5	45.42	246.84	zen	pd
47	p12	46	8	Other	Max/MSP	ACB	5	pde	2	2	40.70	118.02	zen	pd
48	p12	46	8	Other	Max/MSP	ACB	6	pde	3	3	68.46	145.18	zen	pd
49	p12	46	8	Other	Max/MSP	ACB	1	pd	2	2	14.25	52.61	zen	pd
50	p12	46	8	Other	Max/MSP	ACB	2	pd	10	10	95.90	178.24	zen	pd
51	p12	46	8	Other	Max/MSP	ACB	3	pd	6	6	127.02	207.33	zen	pd
52	p12	46	8	Other	Max/MSP	ACB	4	pd	8	8	85.30	163.25	zen	pd
53	p12	46	8	Other	Max/MSP	ACB	5	pd	5	5	113.06	188.24	zen	pd
54	p12	46	8	Other	Max/MSP	ACB	6	pd	3	3	39.81	149.41	zen	pd
55	p13	32	6	Male	Puredata	BAC	1	pd	23	23	77.66	128.59	zen	pde
56	p13	32	6	Male	Puredata	BAC	2	pd	25	25	26.42	54.24	zen	pde

57	p13	32	6	Male	Puredata	BAC	3	pd	2	2	28.57	73.08	zen	pde
58	p13	32	6	Male	Puredata	BAC	4	pd	11	11	35.20	96.36	zen	pde
59	p13	32	6	Male	Puredata	BAC	5	pd	4	4	97.00	131.94	zen	pde
60	p13	32	6	Male	Puredata	BAC	6	pd	5	5	69.66	124.25	zen	pde
61	p13	32	6	Male	Puredata	BAC	1	zen	25	25	35.20	77.81	zen	pde
62	p13	32	6	Male	Puredata	BAC	2	zen	11	11	37.02	76.99	zen	pde
63	p13	32	6	Male	Puredata	BAC	3	zen	5	5	17.77	71.25	zen	pde
64	p13	32	6	Male	Puredata	BAC	4	zen	4	4	27.87	66.11	zen	pde
65	p13	32	6	Male	Puredata	BAC	5	zen	2	2	24.80	63.74	zen	pde
66	p13	32	6	Male	Puredata	BAC	6	zen	23	23	42.47	97.35	zen	pde
67	p13	32	6	Male	Puredata	BAC	1	pde	4	4	14.96	50.85	zen	pde
68	p13	32	6	Male	Puredata	BAC	2	pde	12	7	51.17	92.42	zen	pde
69	p13	32	6	Male	Puredata	BAC	3	pde	11	11	47.70	101.66	zen	pde
70	p13	32	6	Male	Puredata	BAC	4	pde	12	12	26.14	46.84	zen	pde
71	p13	32	6	Male	Puredata	BAC	5	pde	23	23	16.58	46.08	zen	pde
72	p13	32	6	Male	Puredata	BAC	6	pde	23	2	30.36	52.96	zen	pde
73	p3	22	6	Male	Python	CAB	1	pde	16	16	153.44	334.10	pd	pde
74	p3	22	6	Male	Python	CAB	2	pde	23	23	62.19	183.06	pd	pde
75	p3	22	6	Male	Python	CAB	3	pde	16	17	123.36	281.63	pd	pde
76	p3	22	6	Male	Python	CAB	4	pde	14	14	54.27	116.49	pd	pde
77	p3	22	6	Male	Python	CAB	5	pde	12	12	68.10	120.66	pd	pde
78	p3	22	6	Male	Python	CAB	6	pde	18	18	103.86	292.07	pd	pde
79	p3	22	6	Male	Python	CAB	1	zen	14	14	15.50	50.88	pd	pde
80	p3	22	6	Male	Python	CAB	2	zen	17	17	55.97	167.30	pd	pde
81	p3	22	6	Male	Python	CAB	3	zen	23	23	28.26	79.72	pd	pde
82	p3	22	6	Male	Python	CAB	4	zen	16	16	124.61	227.72	pd	pde
83	p3	22	6	Male	Python	CAB	5	zen	18	18	71.46	158.49	pd	pde
84	p3	22	6	Male	Python	CAB	6	zen	12	12	78.37	116.52	pd	pde
85	p3	22	6	Male	Python	CAB	1	pd	18	18	10.01	48.09	pd	pde

86	p3	22	6	Male	Python	CAB	2	pd	16	16	54.62	193.22	pd	pde
87	p3	22	6	Male	Python	CAB	3	pd	14	14	8.76	73.10	pd	pde
88	p3	22	6	Male	Python	CAB	4	pd	17	17	64.58	174.84	pd	pde
89	p3	22	6	Male	Python	CAB	5	pd	23	23	56.86	131.76	pd	pde
90	p3	22	6	Male	Python	CAB	6	pd	6	6	18.75	75.74	pd	pde
91	P4	23	4	Male	Max/MSP	CBA	1	pde	18	18	49.77	183.35	zen	pd
92	P4	23	4	Male	Max/MSP	CBA	2	pde	23	23	17.24	60.59	zen	pd
93	P4	23	4	Male	Max/MSP	CBA	3	pde	17	17	31.12	123.73	zen	pd
94	P4	23	4	Male	Max/MSP	CBA	4	pde	19	19	15.96	110.65	zen	pd
95	P4	23	4	Male	Max/MSP	CBA	5	pde	25	25	17.17	32.54	zen	pd
96	P4	23	4	Male	Max/MSP	CBA	6	pde	11	11	23.42	40.20	zen	pd
97	P4	23	4	Male	Max/MSP	CBA	1	pd	19	19	25.02	92.70	zen	pd
98	P4	23	4	Male	Max/MSP	CBA	2	pd	6	6	10.29	40.93	zen	pd
99	P4	23	4	Male	Max/MSP	CBA	3	pd	1	1	18.39	36.02	zen	pd
100	P4	23	4	Male	Max/MSP	CBA	4	pd	11	11	7.22	21.03	zen	pd
101	P4	23	4	Male	Max/MSP	CBA	5	pd	17	17	23.59	59.04	zen	pd
102	P4	23	4	Male	Max/MSP	CBA	6	pd	18	18	11.23	36.70	zen	pd
103	P4	23	4	Male	Max/MSP	CBA	1	zen	6	6	9.33	19.74	zen	pd
104	P4	23	4	Male	Max/MSP	CBA	2	zen	19	19	18.53	41.70	zen	pd
105	P4	23	4	Male	Max/MSP	CBA	3	zen	1	1	16.23	31.14	zen	pd
106	P4	23	4	Male	Max/MSP	CBA	4	zen	11	11	17.51	67.66	zen	pd
107	P4	23	4	Male	Max/MSP	CBA	5	zen	23	23	17.49	42.86	zen	pd
108	P4	23	4	Male	Max/MSP	CBA	6	zen	17	17	30.86	52.40	zen	pd
109	p5	27	7	Male	Max/MSP	ABC	1	zen	20	19	41.31	98.60	zen	pde
110	p5	27	7	Male	Max/MSP	ABC	2	zen	10	10	22.07	63.72	zen	pde
111	p5	27	7	Male	Max/MSP	ABC	3	zen	20	20	34.80	90.14	zen	pde
112	p5	27	7	Male	Max/MSP	ABC	4	zen	3	3	16.67	61.07	zen	pde
113	p5	27	7	Male	Max/MSP	ABC	5	zen	23	23	26.66	80.03	zen	pde
114	p5	27	7	Male	Max/MSP	ABC	6	zen	16	16	73.64	122.38	zen	pde

115	p5	27	7	Male	Max/MSP	ABC	1	pd	20	20	19.72	54.19	zen	pde
116	p5	27	7	Male	Max/MSP	ABC	2	pd	10	10	31.67	72.77	zen	pde
117	p5	27	7	Male	Max/MSP	ABC	3	pd	15	15	45.95	87.08	zen	pde
118	p5	27	7	Male	Max/MSP	ABC	4	pd	3	3	29.97	84.99	zen	pde
119	p5	27	7	Male	Max/MSP	ABC	5	pd	7	7	63.28	133.62	zen	pde
120	p5	27	7	Male	Max/MSP	ABC	6	pd	23	23	22.12	48.22	zen	pde
121	p5	27	7	Male	Max/MSP	ABC	1	pde	19	23	23.35	80.34	zen	pde
122	p5	27	7	Male	Max/MSP	ABC	2	pde	15	15	99.39	187.45	zen	pde
123	p5	27	7	Male	Max/MSP	ABC	3	pde	7	7	73.80	145.22	zen	pde
124	p5	27	7	Male	Max/MSP	ABC	4	pde	19	19	36.22	73.27	zen	pde
125	p5	27	7	Male	Max/MSP	ABC	5	pde	20	20	31.10	83.40	zen	pde
126	p5	27	7	Male	Max/MSP	ABC	6	pde	10	10	8.31	42.07	zen	pde
127	p6	24	5	Male	Python	ACB	1	zen	14	14	8.36	20.89	zen	pd
128	p6	24	5	Male	Python	ACB	2	zen	5	5	62.22	91.75	zen	pd
129	p6	24	5	Male	Python	ACB	3	zen	21	21	74.81	126.10	zen	pd
130	p6	24	5	Male	Python	ACB	4	zen	13	13	13.00	28.29	zen	pd
131	p6	24	5	Male	Python	ACB	5	zen	7	7	44.99	87.01	zen	pd
132	p6	24	5	Male	Python	ACB	6	zen	8	8	6.83	18.51	zen	pd
133	p6	24	5	Male	Python	ACB	1	pde	21	21	30.39	81.24	zen	pd
134	p6	24	5	Male	Python	ACB	2	pde	13	13	30.02	50.46	zen	pd
135	p6	24	5	Male	Python	ACB	3	pde	7	7	26.14	51.63	zen	pd
136	p6	24	5	Male	Python	ACB	4	pde	8	8	41.84	85.31	zen	pd
137	p6	24	5	Male	Python	ACB	5	pde	4	14	14.08	31.71	zen	pd
138	p6	24	5	Male	Python	ACB	6	pde	4	4	14.14	36.79	zen	pd
139	p6	24	5	Male	Python	ACB	1	pd	5	5	30.01	70.86	zen	pd
140	p6	24	5	Male	Python	ACB	2	pd	13	13	11.76	34.85	zen	pd
141	p6	24	5	Male	Python	ACB	3	pd	21	21	14.30	52.32	zen	pd
142	p6	24	5	Male	Python	ACB	4	pd	14	14	11.48	32.54	zen	pd
143	p6	24	5	Male	Python	ACB	5	pd	7	7	32.40	66.60	zen	pd

144	p6	24	5	Male	Python	ACB	6	pd	17	17	50.66	72.64	zen	pd
145	p7	25	6	Male	Python	BAC	1	pd	9	16	63.81	118.91	zen	pd
146	p7	25	6	Male	Python	BAC	2	pd	19	19	79.41	176.30	zen	pd
147	p7	25	6	Male	Python	BAC	3	pd	21	21	86.28	195.07	zen	pd
148	p7	25	6	Male	Python	BAC	4	pd	1	1	17.96	40.13	zen	pd
149	p7	25	6	Male	Python	BAC	5	pd	17	17	32.65	67.17	zen	pd
150	p7	25	6	Male	Python	BAC	6	pd	4	4	38.69	65.44	zen	pd
151	p7	25	6	Male	Python	BAC	1	zen	1	1	39.01	55.26	zen	pd
152	p7	25	6	Male	Python	BAC	2	zen	17	17	42.79	67.69	zen	pd
153	p7	25	6	Male	Python	BAC	3	zen	21	21	21.00	38.40	zen	pd
154	p7	25	6	Male	Python	BAC	4	zen	19	19	35.63	58.28	zen	pd
155	p7	25	6	Male	Python	BAC	5	zen	16	9	30.59	51.38	zen	pd
156	p7	25	6	Male	Python	BAC	6	zen	13	13	12.54	25.15	zen	pd
157	p7	25	6	Male	Python	BAC	1	pde	13	13	26.32	45.42	zen	pd
158	p7	25	6	Male	Python	BAC	2	pde	17	17	51.23	73.28	zen	pd
159	p7	25	6	Male	Python	BAC	3	pde	4	4	15.86	27.41	zen	pd
160	p7	25	6	Male	Python	BAC	4	pde	19	19	40.94	61.61	zen	pd
161	p7	25	6	Male	Python	BAC	5	pde	21	21	36.64	53.44	zen	pd
162	p7	25	6	Male	Python	BAC	6	pde	9	9	103.53	151.13	zen	pd
163	p8	23	2	Male	Max/MSP	BCA	1	pd	1	1	44.23	110.20	zen	pd
164	p8	23	2	Male	Max/MSP	BCA	2	pd	16	16	31.20	120.09	zen	pd
165	p8	23	2	Male	Max/MSP	BCA	3	pd	20	20	35.85	103.27	zen	pd
166	p8	23	2	Male	Max/MSP	BCA	4	pd	18	18	35.55	125.95	zen	pd
167	p8	23	2	Male	Max/MSP	BCA	5	pd	1	24	39.16	118.32	zen	pd
168	p8	23	2	Male	Max/MSP	BCA	6	pd	7	7	26.05	63.27	zen	pd
169	p8	23	2	Male	Max/MSP	BCA	1	pde	24	24	21.83	134.12	zen	pd
170	p8	23	2	Male	Max/MSP	BCA	2	pde	7	7	17.71	58.59	zen	pd
171	p8	23	2	Male	Max/MSP	BCA	3	pde	16	16	26.91	75.92	zen	pd
172	p8	23	2	Male	Max/MSP	BCA	4	pde	18	18	19.90	55.38	zen	pd

173	p8	23	2	Male	Max/MSP	BCA	5	pde	23	23	19.12	81.08	zen	pd
174	p8	23	2	Male	Max/MSP	BCA	6	pde	6	6	23.65	82.77	zen	pd
175	p8	23	2	Male	Max/MSP	BCA	1	zen	7	7	5.29	15.67	zen	pd
176	p8	23	2	Male	Max/MSP	BCA	2	zen	1	1	2.58	21.16	zen	pd
177	p8	23	2	Male	Max/MSP	BCA	3	zen	20	20	4.86	24.78	zen	pd
178	p8	23	2	Male	Max/MSP	BCA	4	zen	23	23	8.46	28.14	zen	pd
179	p8	23	2	Male	Max/MSP	BCA	5	zen	24	24	13.24	48.41	zen	pd
180	p8	23	2	Male	Max/MSP	BCA	6	zen	16	16	15.68	48.01	zen	pd
181	p9	33	14	Male	Processing	CAB	1	pde	13	13	79.03	137.56	pde	pd
182	p9	33	14	Male	Processing	CAB	2	pde	20	20	107.15	385.66	pde	pd
183	p9	33	14	Male	Processing	CAB	3	pde	19	19	46.94	100.00	pde	pd
184	p9	33	14	Male	Processing	CAB	4	pde	24	24	44.06	154.06	pde	pd
185	p9	33	14	Male	Processing	CAB	5	pde	16	16	52.39	143.43	pde	pd
186	p9	33	14	Male	Processing	CAB	6	pde	10	10	38.46	142.32	pde	pd
187	p9	33	14	Male	Processing	CAB	1	zen	14	14	31.92	64.14	pde	pd
188	p9	33	14	Male	Processing	CAB	2	zen	13	13	32.41	77.22	pde	pd
189	p9	33	14	Male	Processing	CAB	3	zen	19	19	29.51	91.61	pde	pd
190	p9	33	14	Male	Processing	CAB	4	zen	10	10	47.14	133.02	pde	pd
191	p9	33	14	Male	Processing	CAB	5	zen	24	24	53.07	137.47	pde	pd
192	p9	33	14	Male	Processing	CAB	6	zen	20	20	19.27	58.44	pde	pd
193	p9	33	14	Male	Processing	CAB	1	pd	16	16	26.13	54.24	pde	pd
194	p9	33	14	Male	Processing	CAB	2	pd	19	19	27.29	67.74	pde	pd
195	p9	33	14	Male	Processing	CAB	3	pd	24	24	40.68	105.00	pde	pd
196	p9	33	14	Male	Processing	CAB	4	pd	10	10	30.55	56.96	pde	pd
197	p9	33	14	Male	Processing	CAB	5	pd	20	20	25.43	58.90	pde	pd
198	p9	33	14	Male	Processing	CAB	6	pd	13	13	14.25	45.06	pde	pd
199	p10	32	23	Male	Python	CBA	1	pde	1	1	15.38	36.57	pde	zen
200	p10	32	23	Male	Python	CBA	2	pde	6	6	29.72	65.72	pde	zen
201	p10	32	23	Male	Python	CBA	3	pde	14	14	34.39	50.54	pde	zen

202	p10	32	23	Male	Python	CBA	4	pde	10	10	19.45	46.50	pde	zen
203	p10	32	23	Male	Python	CBA	5	pde	23	23	15.58	40.48	pde	zen
204	p10	32	23	Male	Python	CBA	6	pde	21	21	8.45	27.11	pde	zen
205	p10	32	23	Male	Python	CBA	1	pd	21	21	18.27	41.26	pde	zen
206	p10	32	23	Male	Python	CBA	2	pd	10	10	6.24	40.02	pde	zen
207	p10	32	23	Male	Python	CBA	3	pd	1	1	8.84	17.71	pde	zen
208	p10	32	23	Male	Python	CBA	4	pd	6	14	12.82	23.96	pde	zen
209	p10	32	23	Male	Python	CBA	5	pd	6	6	0.69	14.57	pde	zen
210	p10	32	23	Male	Python	CBA	6	pd	12	12	11.89	21.86	pde	zen
211	p10	32	23	Male	Python	CBA	1	zen	6	6	10.51	16.71	pde	zen
212	p10	32	23	Male	Python	CBA	2	zen	2	2	10.48	19.15	pde	zen
213	p10	32	23	Male	Python	CBA	3	zen	10	10	4.91	13.19	pde	zen
214	p10	32	23	Male	Python	CBA	4	zen	1	1	9.63	17.24	pde	zen
215	p10	32	23	Male	Python	CBA	5	zen	12	12	7.67	13.97	pde	zen
216	p10	32	23	Male	Python	CBA	6	zen	23	23	9.99	26.67	pde	zen

B.2 Data analysis script

Follows the data analysis script used in the user study discussed in Chapter 5. The script is writen in R language.

```
##############################
##############################
# Jeronimo Barbosa ##########
# August 14 2017 ##############
##############################

# import json library
library(jsonlite)
library(plyr)

#setting the workspace
setwd(PATH_TO_FOLDER)

#reading one json file
raw <-loading_one_user_file("./pilot/p-user1.json")
raw <-loading_one_user_file("./pilot/p-user2.json")
raw <-loading_one_user_file("./pilot/p-user3.json")
raw <-loading_one_user_file("./pilot/p-user4.json")
raw <-loading_one_user_file("./pilot/p-user5.json")
raw <-loading_one_user_file("./pilot/p-user6.json")

raw <-loading_one_user_file("./rawdata/user1.json")
raw <-loading_one_user_file("./rawdata/user2.json")
raw <-loading_one_user_file("./rawdata/user3.json")
raw <-loading_one_user_file("./rawdata/user4.json")
```

```
raw <-loading_one_user_file("./rawdata/user5.json")
raw <-loading_one_user_file("./rawdata/user6.json")
raw <-loading_one_user_file("./rawdata/user7.json")

#loading all file names
filenames <- list.files("./rawdata", pattern="*.json", full.names=TRUE)

#loading the first
raw = stream_in(file(filenames[1]))

#computing size
size = length(filenames)
print("loading...")

#loading all files
for (i in 1:size) {
  print(i)
  print(filenames[i])
  temp <- stream_in(file(filenames[i]))
  Sys.sleep(1)
  if (i > 1) {
    raw = rbind(raw,temp)
  }
}

#formating data types
raw = configuring_data_types(raw)

#summarizing data
summary(raw)
```

```
56 View(raw)
57
58 #press space bar + enter on the line you want to execute
59 computing_decision_speed_right_answers_boxplot_per_tool()
60 computing_decision_speed_all_answers_boxplot_per_tool()
61 computing_decision_errors_barplot_per_tool()
62 computing_average_completion_time()
63 detail_age()
64 detail_experience()
65 computing_decision_speed_barplot_with_sem_per_tool_all_answers()
66 detail_questionnaire()
67 decision_speed_anova_and_pairwise_tests()
68 decision_accuracy_anova_and_pairwise_tests()
69 bootstrap_and_estimation_plots()
70
71 # function that computes a boxplot over the different evaluated tools
72 computing_decision_speed_right_answers_boxplot_per_tool <- function () {
73     # selecting a sebset of the original data (only the right ones)
74     only_right_answers = subset(raw, decision_accuracy == TRUE)
75     #ploting only the right ones
76     boxplot(decision_speed ~ block, only_right_answers, col="blue", xlab = "Conceptual␣
          model", ylab= "Decision␣time␣(in␣seconds)", names=c("Dataflow", "Imperative",
          "ZenStates"))
77     #drawing a line containing the average
78     abline(h=median(only_right_answers$decision_speed))
79 }
80
81 computing_decision_speed_all_answers_boxplot_per_tool <- function () {
82   # selecting a sebset of the original data (only the right ones)
83   #only_right_answers = subset(raw, decision_accuraccy == TRUE)
```

```
84    #ploting only the right ones
85    boxplot(decision_speed ~ block, raw, col=c("#fff7bc", "#fec44f", "#d95f0e"),
          ylim=c(0,100), names=c("Dataflow", "Imperative", "ZenStates"), frame.plot=FALSE,
          boxwex=.4)
86    title(xlab = expression(bold("Conceptual␣model")), ylab= expression(bold("Decision␣
          time␣(in␣seconds)")))
87    abline(h=median(raw$decision_speed), lwd=1, lty=2)
88  }
89
90  # function that computes a decision speed barplot per tool with standard error of the
        mean
91  computing_decision_speed_barplot_with_sem_per_tool_all_answers <- function () {
92    #getting raw data
93    data.dist <- split(raw$decision_speed, raw$block)
94
95    #renaming columns
96    names(data.dist)[names(data.dist)=="pd"] <- "Dataflow"
97    names(data.dist)[names(data.dist)=="pde"] <- "Structured"
98    names(data.dist)[names(data.dist)=="zen"] <- "ZenStates"
99
100   #computing mean to order
101   data.dist.mean <- sapply(data.dist, mean)
102   #sorting main array
103   data.dist=data.dist[order(data.dist.mean,decreasing=FALSE)]
104   #computing mean again
105   data.dist.mean <- sapply(data.dist, mean)
106   #computing sd
107   sd <- sapply(data.dist, sd)
108   #computing length
109   length <- sapply(data.dist, length)
```

```
  #computing sem
  data.dist.sem <- sd / sqrt(length)
  #ploting the graph
  g = barplot(data.dist.mean, col=c("#fff7bc", "#fec44f", "#d95f0e"), ylim=c(0,100),
      space=0.5)
  grid(nx=NA, ny=NULL, col = "black")
  #zero line
  abline(h=0, lwd=2, lty=1)
  #ploting the error
  arrows(x0=g, y0=data.dist.mean-data.dist.sem, x1=g, y1=data.dist.mean+data.dist.sem,
      lwd=.8, code=3, angle=90, length=0.15)
  title(xlab = expression(bold("Conceptual model")), ylab= expression(bold("Decision
      time (in seconds)")))
  #formating digits
  value=round(data.dist.mean, digits=2)
  #printing actual values
  text(g, 3, paste("n = ", value), cex=0.7)
}

# function that computes a boxplot over the different evaluated tools
computing_decision_errors_barplot_per_tool <- function () {
    # selecting a sebset of the original data (only the wrong ones)
    only_wrong_answers = subset(raw, decision_accuracy == FALSE)

    length(raw$decision_accuracy)/3

    print("counting errors")
    print(table(only_wrong_answers$block))
    print("accuracy rate")
    n=length(raw$decision_accuracy)/3
```

```
    100-((table(only_wrong_answers$block)/n)*100)
}

#computes average completion time
computing_average_completion_time <- function() {
  return(median(raw$endTimestamp - raw$beginTimestamp))
}

#function that computes average age and standard deviation
detail_age <- function() {
  print('AGE')
  print(paste("mean", mean(raw$age)))
  print(paste("median", median(raw$age)))
  print(paste("sd", sd(raw$age)))
  print(paste("min:", min(raw$age)))
  print(paste("max age:", max(raw$age)))
}

#function that computes average experience and standard deviation
detail_experience <- function() {
  print('EXPERIENCE')
  print(paste("mean", mean(raw$experience)))
  print(paste("median", median(raw$experience)))
  print(paste("sd", sd(raw$experience)))
  print(paste("min:", min(raw$experience)))
  print(paste("max age:", max(raw$experience)))
  #filtering one entry per id
  sub = raw[match(unique(raw$id),raw$id),]
  #counting
  print("language frequency")
```

```
  print(table(sub$language))
}

#details the answer of the questionnaire
detail_questionnaire <- function() {
  #filtering one entry per id
  sub = raw[match(unique(raw$id),raw$id),]
  print("easier to understand")
  print(table(sub$easier))
  print("harder to understand")
  print(table(sub$harder))
}

decision_speed_anova_and_pairwise_tests <- function () {
  print(anova(lm(decision_speed ~ block, data=raw)))
  #none
  print(pairwise.t.test(raw$decision_speed, raw$block, p.adj="none"))
  #bonferroni (used in proton)
  print(pairwise.t.test(raw$decision_speed, raw$block, p.adj="bonferroni"))
  #bonferroni (used in holm)
  print(pairwise.t.test(raw$decision_speed, raw$block, p.adj="holm"))
  #tukey pairwaise comparison
  TukeyHSD(aov(decision_speed ~ block, data=raw))
}

decision_accuracy_anova_and_pairwise_tests <- function () {
  print(anova(lm(decision_accuracy ~ block, data=raw)))
  #none
  print(pairwise.t.test(raw$decision_accuracy, raw$block, p.adj="none"))
  #bonferroni (used in proton)
```

```
  print(pairwise.t.test(raw$decision_accuracy, raw$block, p.adj="bonferroni"))
  #bonferroni (used in holm)
  print(pairwise.t.test(raw$decision_accuracy, raw$block, p.adj="holm"))
  #tukey pairwaise comparison
  TukeyHSD(aov(decision_accuracy ~ block, data=raw))
}

subarray_of_logins_and_anwers <- function () {
    #select unique values in subarray 7:16
    subarray <- unique(raw[, 7:16])
    #return result
    return(subarray)
}

loading_one_user_file <- function(filename) {
  #loading all file names
  raw <- stream_in(file(filename))
  raw = configuring_data_types(raw)
  return(raw)
}

configuring_data_types <-function(data) {
  #converting to the right datatype
  data$selectedanswer = as.character(data$selectedanswer);
  data$rightanswer = as.character(data$rightanswer);
  data$block = as.factor(data$block);
  data$durationtime = as.numeric(data$durationtime);
  data$videotime = as.numeric(data$videotime);
  # login
  data$id = as.character(data$id);
```

```
  data$beginTimestamp = as.POSIXlt(data$beginTimestamp)
  data$endTimestamp = as.POSIXlt(data$endTimestamp)
  #profiling
  data$age = as.numeric(data$age);
  data$gender = as.factor(data$gender);
  data$experience = as.numeric(data$experience);
  data$experience = as.numeric(data$experience);
  data$language = as.factor(data$language);
  #final questionnaire
  data$easier = as.factor(data$easier);
  data$harder = as.factor(data$harder);
  data = computing_decision_speed_and_accuracy(data)
  return(data)
}

computing_decision_speed_and_accuracy <- function(data) {
  #computing the decision time
  data["decision_speed"] = data$durationtime - data$videotime
  #computing if the answers were right
  data["decision_accuracy"] = (data$selectedanswer == data$rightanswer)
  return (data)
}

#from: http://vitalflux.com/data-science-scale-normalize-numeric-data-using-r/
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

### ADDED BY SH...
## Gives count, mean, standard deviation, standard error of the mean, and confidence
```

```
    interval (default 95%).
##   data: a data frame.
##   measurevar: the name of a column that contains the variable to be summariezed
##   groupvars: a vector containing names of columns that contain grouping variables
##   na.rm: a boolean that indicates whether to ignore NA's
##   conf.interval: the percent range of the confidence interval (default is 95%)
sh.summarySE <- function(data=NULL, measurevar, groupvars=NULL, na.rm=FALSE,
                         conf.interval=.95, .drop=TRUE) {
  library(plyr)

  # New version of length which can handle NA's: if na.rm==T, don't count them
  length2 <- function (x, na.rm=FALSE) {
    if (na.rm) sum(!is.na(x))
    else       length(x)
  }

  # This does the summary. For each group's data frame, return a vector with
  # N, mean, and sd
  datac <- ddply(data, groupvars, .drop=.drop,
                 .fun = function(xx, col) {
                   c(N    = length2(xx[[col]], na.rm=na.rm),
                     mean = mean   (xx[[col]], na.rm=na.rm),
                     median = median (xx[[col]], na.rm=na.rm),
                     sd   = sd     (xx[[col]], na.rm=na.rm)
                   )
                 },
                 measurevar
  )

  # Rename the "mean" column
```

```
286   datac <- rename(datac, c("mean" = measurevar))
287
288   # Add a column with 4 digits rounded measure for graphs
289   new_col <- paste0(measurevar,"_rnd")
290   datac[,new_col] <- signif(datac[,measurevar], digits=4)
291   # Calculate standard error of the mean
292   datac$se <- datac$sd / sqrt(datac$N)
293   # Confidence interval multiplier for standard error
294   # Calculate t-statistic for confidence interval:
295   # e.g., if conf.interval is .95, use .975 (above/below), and use df=N-1
296   ciMult <- qt(conf.interval/2 + .5, datac$N-1)
297   datac$ci <- datac$se * ciMult
298   return(datac)
299 }
300
301 sh.boot <- function(data=NULL, measure, fun, iter) {
302   library(boot)
303   result = boot(data[[measure]], fun, iter)
304   return <- result
305 }
306
307 sh.histoboot <- function(data=NULL, measure, boot_results, title, bwidth = 0.5) {
308   library(plyr)
309   library(ggplot2)
310   #print(boot_results)
311   d1 = data.frame(Measure = data[[measure]])
312   d2 = data.frame(Measure = boot_results$t[, 1])
313   d1$Source = "orig"
314   d2$Source = "boot"
315   both = rbind(d1, d2)
```

```
316   bothmeans <- ddply(both, c("Source"), summarise, mmean = mean(Measure))
317   ci = boot.ci(boot_results)
318   histo_boot <- ggplot(both, aes(Measure, fill = Source, colour = Source)) +
319     geom_histogram(aes(y = ..density..), binwidth = bwidth,
320                    alpha = 0.6, position = "identity", lwd = 0.2) +
321     geom_vline(data = bothmeans, aes(xintercept = mmean,
322                                      colour = Source), linetype = "dashed", size = 0.5) +
323     geom_vline(aes(xintercept = ci$bca[, c(4)], colour = "CI")) +
324     geom_vline(aes(xintercept = ci$bca[, c(5)], colour = "CI")) +
325     ggtitle(title) +
326     labs(x=measure)
327     print(histo_boot)
328     rm(d1)
329     rm(d2)
330     rm(both)
331 }
332
333 sh.violin <- function(data=NULL, data_sse, measure, facet=NULL, bwidth = 2.5,
        ylabel=NULL, gtitle="") {
334   library(scales)
335   violin <- ggplot(data = data, aes_q(x = as.name(measure))) +
336     geom_histogram(aes_string(y=paste("..density..*",bwidth,sep=""),
            fill="..density.."),
337                    col="white", binwidth = bwidth,
338                    alpha = .8, position = "identity", lwd = 0.2) +
339     scale_x_log10()
340   if (!is.null(facet))
341     violin <- violin + facet_grid(facet)
342   violin_b <- ggplot_build(violin)
343   ylab <- measure
```

```
344   if (!is.null(ylabel))
345     ylab <- ylabel
346   maxd <- max(violin_b$data[[1]]$density)*bwidth
347   violin <- violin +
348     geom_rect(data = data_sse, aes(xmin=CIl,ymin=0,xmax=CIh,ymax = maxd), linetype =
          "blank", alpha = 0.3) +
349     geom_segment(data = data_sse,
          aes_q(x=as.name(measure),y=0,xend=as.name(measure),yend = maxd), linetype =
          "solid", size = 0.2) +
350     geom_text(data=data_sse, aes(x=data_sse[[measure]],y=2*maxd/3,label =
          signif(data_sse[[measure]], digits=4), vjust = "left"),size=2.5,col="black") +
351     geom_segment(data = data_sse, aes(x=0,y=maxd,xend=+Inf,yend = maxd), linetype =
          "dashed", size = 0.2) +
352     geom_errorbarh(data = data_sse, aes_q(x = as.name(measure), y = maxd,
          xmin=as.name("CIl"),xmax=as.name("CIh"),height=0.005)) +
353     coord_flip() +
354     scale_color_brewer(palette="Accent") +
355     scale_y_continuous(labels = percent_format()) +
356     labs(title = gtitle, x = ylab, y = "density", fill = "density")
357   return(violin)
358 }
359
360 raw.fn_mean = function(x, indices) {
361   return(mean(x[indices]))
362 }
363
364 raw.fn_median = function(x, indices) {
365   return(median(x[indices]))
366 }
367
```

```
# Effect size
# Percent difference beetwen 2 measures m1,m2 are computed using
# p = 100 * (m2 - m1) / (.5 * (m1 + m2))
sh.effect_size <- function(m1,m2) {
  return(100 * (m2 - m1) / (.5 * (m1 + m2)))
}

bootstrap_and_estimation_plots <- function() {
  boot_results.r <- 2000

  #Bootsraping distributions for decision_speed
  #All conditions
  boot_results.time.all = sh.boot(raw, "decision_speed", raw.fn_mean, boot_results.r)
  print(boot_results.time.all)
  #sh.histoboot(raw, "decision_speed", boot_results.time.all, "All data")

  #By block (i.e. technique)
  boot_results.time.by_block <- list()
  for (block in levels(raw$block)) {
    raw.filter <- raw[raw$block == block, ]
    rest <- sh.boot(raw.filter, "decision_speed", raw.fn_mean, boot_results.r)
    #sh.histoboot(raw.filter, "decision_speed", rest, block)
    boot_results.time.by_block[[block]] <- rest
    print(rest)
  }

  #Bootsraping distributions for decision_accuracy
  #All conditions
  boot_results.acc.all = sh.boot(raw, "decision_accuracy", raw.fn_mean, boot_results.r)
  print(boot_results.acc.all)
```

```
#sh.histoboot(raw, "decision_accuracy", boot_results.acc.all, "All data")
summary(raw$decision_accuracy)
summary(raw[raw$block == 'zen', ]$decision_accuracy)
summary(raw[raw$block == 'pd', ]$decision_accuracy)
summary(raw[raw$block == 'pde', ]$decision_accuracy)

#By block (i.e. technique)
boot_results.acc.by_block <- list()
for (block in levels(raw$block)) {
  raw.filter <- raw[raw$block == block, ]
  rest <- sh.boot(raw.filter, "decision_accuracy", raw.fn_mean, boot_results.r)
  #sh.histoboot(raw.filter, "decision_accuracy", rest, block)
  boot_results.acc.by_block[[block]] <- rest
  print(rest)
}

#Computing 95% CIs for decision_speed
#All conditions
raw.sse.time.all <- sh.summarySE(raw, measurevar = "decision_speed")
cis = boot.ci(boot_results.time.all)
raw.sse.time.all$CIl <- cis$bca[, c(4)]
raw.sse.time.all$CIh <- cis$bca[, c(5)]
print(raw.sse.time.all)

#By block (i.e. technique)
raw.sse.time.block <- sh.summarySE(raw, measurevar = "decision_speed",
                                   groupvars = c("block"))
raw.sse.time.block$CIl=raw.sse.time.block$ci
raw.sse.time.block$CIh=raw.sse.time.block$ci

```

```
#compute confidence intervals for each conditions in the summary using bootstrap
    results
for (block in levels(raw.sse.time.block$block)) {
  pouet <- boot_results.time.by_block[[block]]
  cis = boot.ci(pouet)
  raw.sse.time.block[raw.sse.time.block$block == block, ]$CIl <- cis$bca[, c(4)]
  raw.sse.time.block[raw.sse.time.block$block == block, ]$CIh <- cis$bca[, c(5)]
}
print(raw.sse.time.block)

#Computing 95% CIs for decision_accuracy
#All conditions
raw.sse.acc.all <- sh.summarySE(raw, measurevar = "decision_accuracy")
cis = boot.ci(boot_results.acc.all)
raw.sse.acc.all$CIl <- cis$bca[, c(4)]
raw.sse.acc.all$CIh <- cis$bca[, c(5)]
print(raw.sse.acc.all)

#By block (i.e. technique)
raw.sse.acc.block <- sh.summarySE(raw, measurevar = "decision_accuracy",
                                  groupvars = c("block"))
raw.sse.acc.block$CIl=raw.sse.acc.block$ci
raw.sse.acc.block$CIh=raw.sse.acc.block$ci

#compute confidence intervals for each conditions in the summary using bootstrap
    results
for (block in levels(raw.sse.acc.block$block)) {
  pouet <- boot_results.acc.by_block[[block]]
  cis = boot.ci(pouet)
  raw.sse.acc.block[raw.sse.acc.block$block == block, ]$CIl <- cis$bca[, c(4)]
```

```
  raw.sse.acc.block[raw.sse.acc.block$block == block, ]$CIh <- cis$bca[, c(5)]
}
print(raw.sse.acc.block)

# Simple bargraphs with 95%CIs as error bars
dodge <- position_dodge(width = 0.9)

#Decision speed
bars.time.block <- ggplot(raw.sse.time.block, aes(x = reorder(block, decision_speed),
    y = decision_speed, fill = block)) +
  geom_bar(stat = "identity", position = dodge) +
  geom_errorbar(aes(ymin=CIl, ymax=CIh), width=.2, position=position_dodge(.9)) +
  labs(x = expression(bold("Conceptual model")), y = expression(bold("Decision time
      (in seconds)")))
print(bars.time.block)

#Decision accuracy
bars.acc.block <- ggplot(raw.sse.acc.block, aes(x = reorder(block,
    decision_accuracy), y = decision_accuracy, fill = block)) +
  geom_bar(stat = "identity", position = dodge) +
  geom_errorbar(aes(ymin=CIl, ymax=CIh), width=.2, position=position_dodge(.9)) +
  labs(x = expression(bold("Conceptual model")), y = expression(bold("Decision
      accuracy")))
print(bars.acc.block)

#Print distribution graphs with CIs
#All conditions
violin.time.all <- sh.violin(raw, raw.sse.time.all, measure="decision_speed",
    bwidth=0.15, ylabel="Completion Time (s)", gtitle="Completion Time")#bwidth=4
print(violin.time.all)
```

```
481   #By block (i.e. technique)
482   violin.time.block <- sh.violin(raw, raw.sse.time.block, measure="decision_speed", ". ~ block", bwidth=0.15, ylabel="Completion Time (s)", gtitle="Completion Time by Technique")
483   print(violin.time.block)
484
485   #Effect size
486   print("pd vs pde decision_speed effect size")
487   print(sh.effect_size(raw.sse.time.block[raw.sse.time.block$block == "pd",]$decision_speed,
488                        raw.sse.time.block[raw.sse.time.block$block == "pde",]$decision_speed))
489
490   print("zen vs pd decision_speed effect size")
491   print(sh.effect_size(raw.sse.time.block[raw.sse.time.block$block == "zen",]$decision_speed,
492                        raw.sse.time.block[raw.sse.time.block$block == "pd",]$decision_speed))
493
494   print("zen vs pde decision_speed effect size")
495   print(sh.effect_size(raw.sse.time.block[raw.sse.time.block$block == "zen",]$decision_speed,
496                        raw.sse.time.block[raw.sse.time.block$block == "pde",]$decision_speed))
497 }
```

Appendix C

StateSynth User Study–Raw transcriptions

Follows the raw trascribed material used in the user study discussed in Chapter 6.

C.1 Participant 1

C.1.1 Questionnaire

Answers in a Likert-scale from 1 (newbie) to 5 (expert).

How would you rate your keyboard/composition expertise? 3.

How would you rate your computer programming skills? 2.

C.1.2 Profiling interview

INTERVIEWER: There are two main topics for this interview. We're going to start with your musical background and then we move to programming related questions. So to start with the musical ones. First: Could you tell me a little bit more about your keyboard and composition experience? How's that?

P1: I've never had a keyboard lesson in my life. I play the guitar. But I'm familiar with MIDI

keyboards because I have some at home. I mainly use it to sketch out ideas and then use the MIDI notes to program... to perfect my ideas in a DAW like Ableton, or Reaper. I have a piano at home that I mess around sometimes... [laughs]

INTERVIEWER: [laughs]

P1: So that's it. I'm not the keyboard player, and I never played professionally, or in a band or in a song...

INTERVIEWER: So sketching musical ideas is your key usage of [the keyboard]. Okay. So what specific tools do you use in this practice, with the keyboard? You said you have a MIDI keyboard. So what kinds of software or tools you use with the keyboard?

P1: [Software] synthesizers. In DAWs like Ableton, Logic, Reaper, and also lots of different VSTs types, like Arturia keyboards, all the stuff in Ableton, sampler... I use the keyboard with granular synthesizers also. I also had hardware keyboards but I sold them. [laughs] Because I was not able to play the keys, and it took a lot of space in my room so I switch to desktop and after... I'm actually more on the iPad. I'm a big iPad fan. I do a lot of stuff on the iPad, this software synths from the iPad, the apps, the touch stuff.

INTERVIEWER: So often you connect your media keyboard to the iPad?

P1: Yes, with the connectors.

INTERVIEWER: Great, okay. I think you kind of responded this question that I'm going to ask you, but maybe you could detail a little bit more. Could you briefly describe one example of one piece—or one of these sketches that you told me about—where you used the keyboard and the iPad [together]. Could you tell me a little bit more how would you do this? Shortly.

P1: I have an answer to that but not with the iPad. I'm in a synth wave band. We are actually right now producing an album, and then my band mate, we are a duo, he comes up with ideas and we work on them together. So at a time, I might want to do an arpeggiator on a song. I have an Arthuria keyboard and I can try ideas there in the arpeggiator, and we record the MIDI in the song. And after that, we tie everything up. So I have this kind of experience.

Also, to do kind of little basses—mono things are more simple to me. I can do rhythm pretty

good with one or two fingers.

Also, with the iPad, for very ambient and long pads. So I don't switch chords or keys too fast. That's really it. I can play song chords, but really by ear. I know my keys, I know where the C is, so I can set everything out, my ideas, my songs, what I want to do. But not so fast...

INTERVIEWER: Okay, I see. And how have you learned in the case of the keyboards and these software [that you use]. So how do you learn how to use them?

P1: I learned a bit at school. I took a course called "Audio and MIDI musical creation", one session. And after that at my program we used all kind of stuff, our teachers pushed us to explore a lot of different software, but really on our own, and talking to people... I spent a lot of time on websites such as Synthopia... like iPad [inaudible] stuff... I'm a little bit geek about software. [Laughs]

INTERVIEWER: Okay, cool. And how do practice... the ensemble... the keyboards and the synthesizers—if you consider that you practice...

P1: No, I think that I just play. It's really by feeling. I don't do keys on a regular basis, but it's part of the project I'm working right now [the synth wave band]. So I do a little more often right now than before. Also with the iPad, that I can just sit and try some stuff with granular synths... mostly granular synths.

INTERVIEWER: Okay, I understand. That's it for the musical practice. So I'll move to the programming part. More or less the same kind of questions, so let's start. You've filled out two on the questionnaire [out of five in a Likert scale]. I would like to understand more about this programming experience.

P1: I took one... two... three... four programming courses in the university. One was HTML, so it's not really related. There are two mandatory courses as part of my bachelor degree at my school: One is with Pure data [dataflow programming language]; Another with PYO [Python library for music]. So I know a little bit how to program on these platforms, on these languages, but I didn't really continue to use them after the courses. Pure data a little bit more. But PYO, not really. So when I was there, studying and focusing on these courses, I think I was kind of good

because I've had good grades. But after that, I didn't really continue. Right now I even using Max 4 Live [Cycling 64 framework to incorporate Max/MSP inside Ableton Live], but I don't go deep into the patches. That's pretty much it.

The courses were once per week, three hours per class. It's been two years since I've finished the last of these courses. I also took an introductory course to C. That helped with the PYO, because I was used to the text programming. It really helped me a lot on confidence.

INTERVIEWER: Could you exemplify—if possible—one project where you needed to do the programming and the music together. *P1:* We had to do small pieces, about a minute or so. Our last project in the PYO class was like... mini pieces. We had to program MIDI notes and short sequences... a sequencer, like a step-sequencer, and different kind of synths that we programmed ourselves. But everything I did was programmed by examples. You get the example and you start with this, and after you could build up your ideas with trial and error.

INTERVIEWER: Great. How you learned it, you already mentioned it, right? Through the courses, you took at university. Great. This final question has two parts. I will ask you to take your time to think about one experience involving programming and music where you felt of one of the following sensations I'm going to mention to you. The first one is: One occasion where you felt struggling with the programming. Where you found "man, this is hard, I can't do it"... Take your time and then, please, share this moment with me. Summarized.

P1: [Pause] I think... it was at a time, in our last Puredata project. It was one of these mini pieces of music, but it was a little bit more complex because we had to build synths and rhythms. And... It was like copying some stuff around, from other patches, and then applying it on the patch I was working on. Then, something inside the under [sub] patches was not working. It was a little patch we used to do simple and not so simple operations, that we were trying to use in the big picture. I think it was related to the rhythm section, I don't know. This was a little bit more complex to think because the more simple things I could look up on the internet and try to connect these things to what I was working on. I hadn't programmed it myself, I had copied it from some other patch—maybe something found on the internet because it was all open source

stuff. When I tried to use it, it was not working the way I was thinking it would work.

INTERVIEWER: I understand, thank you. The second question is the same thing, but completely opposite: if you can remember one occasion where the programming felt... "oh, this makes sense! this is intuitive! I'm on the control of what is going on". Where you felt you were able to express yourself within the code.

P1: I think with the PYO. I was a little bit easier because I had taken the C course. All the operators, the words to use to code, they were pretty well documented. And for me, it was easier because it was a little bit more linear. Sometime in patch programming, you forget something because the patch is big and hidden. But in PYO, it's linear stuff, that I could just go, it was a little bit easier. I coded lots of synths, dronny stuff, ambient stuff, and that was fun to do with PYO.

INTERVIEWER: Okay, thank you! That's all for the interview.

C.1.3 First meeting

Up to this point, P1 had watched tutorial 1 and had spent 1 work session playing around with the software. By the end, P1 provided us with a list of small bugs he found in the software.

INTERVIEWER: Here's your last diary. I had three questions for you. In the first one, I asked you "How was your working session? What were you trying to achieve?". Your answer was pretty much straightforward. "I was looking at all the effects and augmenters to learn how to use them and to find bugs."

P1: Just really basic... I didn't go too deep in the software in one hour. Maybe I spent too much time looking for bugs... [laughs] Because when I found one, I stuck to that. I thought about it as a list, so I passed all the effects...

INTERVIEWER: Okay. The second question was: "What were you trying to achieve today in creative terms?". You said: "Lots of stuff. Short sequences, droning, ambiances, repeating notes with additional input from the keyboard, messing around with the possibilities of the oscillators, FM synth and effects, pretty much".

P1: Yeah. When I began with the oscillator with the keyboard, I had a lot of fun really fast. Like, I was nice to be able to begin stuff, and just to mess around with one parameter at a time, and there was already nice possibilities for experimental stuff. The sequencers that I mentioned was pretty much done by hand. I didn't sequence with programming or using other parameter tasks. [For example], I used the delay to do repeating notes, and put some chords... maybe we shouldn't call that sequencing but... [laughs]. I wasn't like programmed sequences. It was more by felling, messing with chord progressions on the keyboard. Really with the rhythmic possibilities of the effects and the oscillator and the FM instrument. When I put the frequency carrier at a certain value, it begins to do more rhythmic stuff... it was a lot of fun for a first session.

INTERVIEWER: Nice. Let's move on to the third final question. [I asked] "Did you face problems today in learning and using the system? If so which problems." Then you said: "The only thing is knowing the values of the different parameters. If these were shown on the screen I would know the limitations and then be more creative faster." Do you mean the range of values in each [high level] parameters?

P1: Yeah. For the instruments, and for the effects. Also for the augmenters, maybe including a drop-down menu or something [for the intervals]. You said, in the end, I could use [the blackboard] text to control these parameters, but I hadn't thought of that before. Maybe [in my next session] I'll go a lot deeper in that side of the software.

INTERVIEWER: Great. Anything else about the software?

P1: No...

INTERVIEWER: Okay. I'm now making a few more questions. Some of them may not be relevant to you, but I'll make them in anyways in case something comes to your mind. Let me know. The first questions are about progress towards your goal. Remembering that you have been asked to create short interactive music pieces using the system. Do you think have made any progress so far. Have you started considering anything?

P1: Yes, I have certain questions, but there wasn't a lot of time with the software to be at this point. The learning curve, I think.

INTERVIEWER: Perfect. Another question, you already talked about it, about novel creative explorations. Is there anything in particular, features, that came to your mind as being "oh, I never thought about this, this could be interesting maybe in the future"...

P1: Not so much... Are you talking about maybe features to add in the software? I don't know if you told me that but MIDI out.. do you have MIDI in and out?

INTERVIEWER: Yes, we do. Maybe it's too early [for this question], this is not a good question for now. I'll move on to the third one. Did you like the result so far? Actually, you answered that too right? About your experience with the system and what you have achieved so far. Do you think it's going well for you?

P1: Yes, it is going well. Maybe the only concern would be a bit about the user interface, [the issues] I told you about. I think fonts are a bit small. But as a first step, for the first version of the software, I think it's pretty great. It makes me want to go deeper into learning it. That's sure. Even, like, after 10 minutes, it was certain that I was having fun and that wanted to go deeper to learn it.

INTERVIEWER: Great. Another question: Any feature that you have seen so far that you thought it was hard to understand, challenging, tricky... About what you've seen so far.

P1: Not for me. No, I don't think. Because I have a certain background and that stuff. Maybe that's was I had so much fun. Because I could understand it pretty much easily at first glance. I think this will be a good question to ask after the next session. Because I passed all the basic stuff, and I will start the more advanced stuff, sequencing, and all that stuff, I think it will be a good question to ask.

INTERVIEWER: Perfect. Okay. So there are two other [questions] following up these ones. Any challenges in learning, any struggles in coding...

P1: Yes... In the last section, I was so into finding bugs that pretty much forgot there as a programming part of the software. So when you told me to use the mouse... I haven't done much programming in the last months... years... so that side will have to come back... [laughs] Maybe just I'll take some notes, just some references, maybe if you can give some input, or with

tutorials... just to have some ideas for the programming side, for the creative side, that would be fun. That would be a good input for the next steps, I think.

INTERVIEWER: Perfect. For the last question, about the software bug, I already have your extensive feedback, that is noted. [laughs] Perfect. Thank you, we are done!

C.1.4 Second meeting

Up to this point, P1 had skipped tutorial 2 (experimenter recommendation) and focused on tutorials 3 and 4. After, P1 spent a second work session playing around with the software, aiming at creating one small piece to present in the next section. P1 spent most of the session on one single file and playing a lot with it.

INTERVIEWER: Here's your last diary. In the first question, I asked you "How was your working session? What were you trying to achieve?". You said: "I was trying to build a complex set of states and transitions to build a short beat-oriented piece. I've had a few crashes but overall it was really fun!"

P1: [Laughs]. Yeah, I've imported few samples and I built a patch with five of six states. I [used] the sample player, and I imported the snares, kicks, hi-hats, and stuff like that, to see what was possible in terms of... sticking to a grid... maybe... yes or no... [Laughs]. Maybe not. So... And finally, I did transitions with chords. I used different techniques to loop the samples and to build chords, and I also stacked generators.

[For example, now pointing to the screen] It begins here, where I have a hi-hat. I used the chord augmenter to put it on repeat. So it plays as a loop, and it does other hi-hats [by pitch-shifting the samples with different durations]. I've added a filter as well, a high pass to make the sound less [louder], I thought it was a little bit loud.

Here, it's some kind of effect. I stacked another sample because.. at first I used only the oscillator and after wanted to switch to the sampler, but I didn't take out the oscillator and, in the end, I thought it was nice like that [laughs]. I could layer two sounds and make them obey the same rule of the state.

Here, I did a little FM synth with delay and a chord augmenter to... like, when the [the system] arrives on the state I can play chords here. And there I did the same thing but with the snare.

The transitions are based like in your video [tutorial], it's by beat, but I did it in 5 [as tempo signature] and not 4/4. In the end, I tried to go a little more complex by building this one, which is a sampler with an oscillator, like this. But I tried to use a longer sample to have an ambiance behind the little piece, like a wind, a forest ambiance. And I tried to do a condition where after a [certain] bar it [would] comes to play for that much time and after it [would] go back to the [main] loop.

So with that, it was more an improvisational tool than like a piece. I improvised a lot with the parameters. It was fun, it had a lot of nice options. It's rare that I compose music that way, with a fixed set of tools and just improvising over... within the limits of these loops of samples... but it was fun!

INTERVIEWER: Nice. So let's move on to the second question. I asked you "what have you tried today in creative terms?". I think you already present everything you did, but I'll just say it here out loud what you've written. "I have imported a few samples to try to create a piece with them, I've built a little patch with transitions and looping samples. It became more an improvisational tool, so I messed around with it for a bit amount of time". Pretty much what you've said. I think we've covered this one. So in the last one, I asked you: "Did you face problems today in learning the system? If so which problems."

You said: "Only a few crashes and sound cutting at some point. Also, I tried to get a little more complex with the transitions, I had to ask some help in finding the right expressions to do it right way". Could you talk a little bit more about this?

P1: Yeah. It's really with the programming part. It's really not natural for me. I have to think about this part a little bit more. The patch in itself was fairly easy because it is easy to build. But you want to make it more complex, to evolve, you need the programming part. And that's where I had mind bugs. [laughs]. I used the modulo [mathematical operation %], which I think it's nice inside conditions... it makes it go to one state but it won't come back after. And I

didn't get to use the ramp [blackboard variable]. You helped me with that, that was a good call, but I was no much in the sound design that I forgot about the other possibilities of the software.

INTERVIEWER: One quick question about this. You are saying "the programming part". By "programming part", you mean so far the mathematical expressions?

P1: Yeah.

INTERVIEWER: Is there anything else you would include as "the programming part" you're mentioning?

P1: Hm... No. Pretty much mathematical expressions. That's where it's not so much natural to me.

INTERVIEWER: Okay. I think that's it for the dairy, thank you.

P1: I could add something, I just had an idea. Maybe here [somewhere in the interface] you could have the option to have a set of expression that is pre-built. So people could just take one expression and use it. So that it would help to make this kind of things more quickly. After that, you could just change the number...

INTERVIEWER: You mean dragging and dropping built-in expressions.

P1: Yes. That would be fun. I had that, I think I could make it evolving and more complex really really fast.

INTERVIEWER: Okay, thank you!

C.1.5 Retrospective meeting

Both the second work session and the final retrospective session were carried in the same day. P1 presented an improvisation using state machines created during the second work session. The interview presented below followed this performance.

[The performance ends, and there is a pause. P1 and interviewer laugh together. Interviewer compliments the performance and P1 comments...]

P1: For improvisation, this is really nice. Just because it doesn't work the way it should, it's nice. [For example], to have the sampler time not equals in all the keys [reference for the pitch-

shifting algorithm that changes playing speed], like when I had the oscillators and the kick, so [in one key] it was really fast and here [another key] really slow, it's nice. It makes a nice contrast in the system. While I was playing, I was thinking of lots of other stuff I could do. It's really nice.

INTERVIEWER: Okay, let's start the retrospective interview. [Explains how the interview unfolds]. Before we go to the main questions, just two quick things that came to my mind while you were playing. Have you ever heard of live coding?

P1: Yes.

INTERVIEWER: And have you ever done some live coding?

P1: No, never.

INTERVIEWER: Do you realize that what you were doing is sort of live coding?

P1: Sort of. Maybe more of tweaking the system... I think about it more of a modular synth... Because these are somehow fixed parameters in states, whereas in live coding you can create everything in the performance

INTERVIEWER: I understand. So let's move to the level of use questions first. In terms of technical usage of the system—by technical usage I mean all the features you've seen in the tutorials and that you've explored here—, do you feel there is still much to learn inside the system?

P1: No. No, really not. I think with the right amount of investment for yourself you can learn it pretty easily. I've been able to do that... today. I think that with a clear manual and the videos, it was very easy to understand. For the features, I didn't use the sub-state. [Actually,] this was supposed to be a sub-state, but I just spent too much time tweaking it... [laughs] But this is nice because it is the right path. For me, it's cool to be able to do something easily from scratch. That's what I've done today [in the second work session and the performance] with a basic set of knowledge from the program. I have maybe some ideas for features. Can I tell you right now?

INTERVIEWER: Yes, please. [P1 suggests some features, namely: master control in sub-patches; more tasks and effects; improve the delay]

INTERVIEWER: Thank you for the suggestions, they're now recorded. Moving on, still on the level use. How would—considering this system as a programming tool, a tweaking tool, as

you said—you compare this tool with other tools you tried in the past? In technical [usage], learnability point-of-view. More specifically, you mentioned PYO, Max, and Puredata.

P1: [Pause] I think that having a limited set of states and parameters is really good for creative use. In Puredata I was always scratching my head to know what was the source of my problem, which subpatch... so having the limitations is nice. I think there could be more things inside and it wouldn't take out the limitations inside [your program]. [For example,] more generators, more parameters. I don't think that would take off the limitations. And in terms of ease of use, maybe for the programming part, maybe a drop-down menu for the expression. Just for the real beginners, that would be really great because... when you start, when you open the software, you [could be] thinking the programming part is important for the software, and that it is not just an oscillator, or audio generator tool. So I think that if I had that in the beginning, the experience would have been different. I think the patch would have more transitions, more programming, and stuff...

INTERVIEWER: Okay. Considering still these examples, Max and PYO. If you were to build exactly the same thing you built today. Within these tools. [Silence and long pause]

P1: Was the question like... if I would build... simpler or more difficult...

INTERVIEWER: Yeah. How would you compare, I want to hear from you. We are still talking on technical [usage] terms, and not the creative potential.

P1: I think... I don't know Max really well. I know the basics on paper, but I haven't so much delt into it. But I know it has some sort of really high level, and you can go a little bit lower—if you want really to program and if you know how to. But there are fixed notes, and things you can just route really easily, I think. In Puredata, there is that also, but it's not so much easy to start from scratch. In PYO, for me, it's different because it's text programming, like supercollider. I think it's a bit simpler, because... it looks like C. In terms of technical [usage], [your program] doesn't look like PYO. PYO would be more sequencing stuff. But maybe I could do something like that in PYO. But for me, it would be really different because I have to think about how it works, how the expressions are, whereas here it's a good amount of visuals and programming expressions.

I think there is a nice balance in the software. It could be better, but it's a first step, so... [silence and long pause]. Yeah.

INTERVIEWER: Okay, thank you. So let's move on to the same questions but [applied] in a different topic, the expressivity. About what you were able to achieve today. The first question would be: In terms of creative potential, do you feel there is still much to explore within the prototype?

P1: Too much?

INTERVIEWER: No, if there are things to explore—left to explore. If there was—for example—other sessions, would you feel that you still have more things to explore...

P1: Yeah, yeah...

INTERVIEWER: ...or do you think that what you've done today, that's it [all the software can do].

P1: No, no, no, I think there's a lot more to explore, like the subpatches. Subpatch, subpatch, subpatch... [laughs] If the software is stable enough you can create a lot of subpatches and make a big mess around that. [Also,] I didn't really use the Javascript programming at all. So a few examples for that could have been nice. I could go deeper with that, I think. But it's good to leave that at the end because there is already so much to do with a few hours exploring the software. In this patch, I didn't really use the [blackboard variables], ramp, LFOs, this kind of stuff for automating parameters. It was more a plug and play patch. But this could have been really cool to express further, to make something more complex. In terms of playability, sometimes there were clicks and drops of audio. And there were parameters that would like to teak more. For example, in the sampler—how it's made—duration doesn't work well. [P1 starts detailing bugs and suggesting system improvements on the sampler].

INTERVIEWER: Thank you for these suggestions. So, still on the expressive potential—that is, things you can create out of [the prototype]—, how do you think [the prototype] would compare to other tools. Not only the programming tools but also DAWs, for example, like Ableton.

P1: It's sure that Ableton is really easy to use. I think that from scratch you can do so much

with it. So much more than that [the prototype does]. But, it's not the same paradigm. To me, this could complement [Ableton] Live well, if I could use it in Live. It's like a sketchpad, an experimental sketchpad, with the sampler... If the sample was really well-featured and robust, it would be really nice with Live.

[For example] In the iPad, I use an app called Samplr, it's like.. amazing. It was a little bit like that in the workflow. There are six tracks. You if have modes, there would be six-seven play modes. So you have a sample, and one of the play modes is looping, and you can loop. At the same time, the other sample there is an arpeggiator on it, so you can do rhythmic stuff [imitates the sounds of an arpeggiator with his mouth]. This makes very nice textures and clear rhythmic songs. Sure, with just that app... It's like using another VSTs... it's the same thing. I use it more like an instrument, the iPad. You can do so much with it. There is also a software called Audulus. For the iPad. It's node-based and patch. But I don't use it, and there a looooooooot of stuff you can do. It's more like Max and Pure data.

I think that goes down to the limitations. I think the limitations are a good point. A good thing to have.

INTERVIEWER: Okay, that was it! Oh, maybe one last question. You said that in Samplr you have this feature of six different voices and that when you play the Samplr, you use these different voices, playing a particular voice, with loops. Were you trying to achieve the same results here [, in the prototype]?

P1: Maybe that was what I was after. [laughs]. Well... when I think it, that's how I do sound design, really often. This app and other stuff from the iPad, it's really easy, because it's hands-on, it's touch stuff. For example, the sample's duration [P1 details how to improve the system's sample task]. Doing all these complex stuff without stopping and breaking. That would be really cool. But I wasn't able to do it easily here yet. But I think that with future developments of the software this should be really easy to do.

INTERVIEWER: Thank you! The last one: Would you be interested in keep exploring this software, in the future, if problems are solved the software is no longer a prototype?

P1: Yes, Sure. Keep me updated.

C.2 Participant 2

C.2.1 Questionnaire

Answers in a Likert-scale from 1 (newbie) to 5 (expert).

How would you rate your keyboard/composition expertise? 5.

How would you rate your computer programming skills? 2.

C.2.2 Profiling interview

INTERVIEWER: There are two subjects for this interview. We're going to start with your musical background and then we move to programming related questions. So to start with the musical ones. First: Could you tell me a little bit more about your keyboard and composition experience? How's that?

P2: Okay. Keyboard, not a lot. I mean, I've been playing the keyboard by myself it's been 10 years. [Mostly] tonal music, [and] chaotic stuff [laughs]. But [for] the last five years, I've teaching harmony at UdM so need to correct harmony homeworks. I can play chorals and stuff like that on the keyboard. As a musician, I've been playing music since 16, maybe? [The participant is 31 years old]. First rock music and pop music, and playing the electric bass. After that, I went to university to try classical music. First, for film music, but after I went somewhere else. I've been composing since that moment.

INTERVIEWER: Okay. And today for what [purpose] you use the keyboard, mostly? In your practice.

P2: Actually, I don't compose that much using the keyboard.

INTERVIEWER: What is your instrument? Do you have an instrument?

P2: I play the electric bass, but I compose using paper and computers, mostly in my head [laughs]... that's my instrument. Because, [instruments] just don't work in my language, my

musical language. Not because I do not use harmony and stuff like that, just because I can't play while I write. So sometimes I can try some stuff, but instead, I would use sequencers and stuff like that. I don't know if that helps you or... [laughs]

INTERVIEWER: It does!

P2: But yeah. Not that much, but just sometimes so that I can try some melodies and stuff like that...

INTERVIEWER: Okay. So what specific tools do you use in your practice? [silence and long pause] In your musical practice.

P2: Musical or composition? Because they might be different. Oh, I'm using the keyboards actually. I haven't told you that before. I play in a pop band too. For weddings and stuff like that. There, I use the keyboard for small things because it's pop music. Yes, I play that, and also use machines like Native Instruments [music technology company], hardware sequencers, stuff like that, that I use in conjunction... all together [with the keyboards]. Yeah. Sorry, what's the question again?

INTERVIEWER: About the tools you use. And you're answering it.

P2: Yeah, I also use Logic, a lot. All Native Instruments synths, a bit of Reactor—although it's been a while since I used that one—, and Sibelius, for writing music, and doing the scores.

INTERVIEWER: So these tools are more related to your composition or your musical practice?

P2: Both. I mean, I don't use the [some] computer software for performing. I mean, I use synth and stuff for performance, but I don't use Logic, Live or Sibelius for performance of course. For composition too [unclear]. And I try to use more the Maschine... I don't know if you heard of this hardware called Maschine. It's a sequencer, but I was looking for a tool because I hate composing with the mouse. I find the mouse so slow.... I tend to be more intellectual, that's actually a problem with my composition because I'm always thinking too much. I wanted something that allowed me to do music just with my hands. So those faders, and stuff like that, help me. A lot. [However,] since [the Maschine] was made for hip-hop and beat production, it doesn't translate well for my music. But I try to incorporate it and work with that.

INTERVIEWER: I see. Another question. Could you very briefly describe one piece—that you've performed or composed—where you used the keyboard and that you think that somehow represents your relationship with the keyboard.

P2: [Silence and long pause] That's a good question. [Silence and long pause]

INTERVIEWER: Or you could only describe one of the occasions you mentioned where you use the keyboards. One, in particular.

P2: The thing that comes to my mind is using samplers and synths when you want people for your projects... How can I say? I really wanted to write pieces for synthesizers in concert [classical] music. Maybe that's something that differs my practice from someone else. And I try to find the tools for them to do that. Or I just like to try stuff on the keyboard without much performing, but just using synths and trying to find the sound with my hands. I don't know if that helps you now... [laughs] Yeah, using synths to try textures and stuff.

INTERVIEWER: Let me know if I got you right. You would use these software [systems you mentioned before] to experiment with sound...

P2: Yeah.

INTERVIEWER: ...and the keyboard would only be the tool to activate these sounds.

P2: Not necessarily, because the difference for me when I think of synth sound is the pop way—which is if I play the C, I'll hear a C but with the synth sound. Not just for "pre-rings". I want to use the octaves from the keyboard to create things with that. Like a synth, when I use the clavichord sound, that's what I would hear. Hm... Yeah. Mostly for experimenting, not as a final piece, as a final tool.

INTERVIEWER: Okay. And how have you learned it? You said you are no keyboard player. So how have you learned it?

P2: By myself, I have no technique. Just trying things, what comes to my mind, and just play with it. Since I'm not trying to do it in performance, it's always in an experimental category [laughs], I found my own way of playing [the keyboard]. It suits what I'm doing.

INTERVIEWER: Okay. Perfect. I think this answer also answer the following question. If

you practice keyboards at all.

P2: Not in a conservatory way. I mean, I'm playing.

INTERVIEWER: I understand you're always practicing, in a sense.

P2: Yeah, exactly. But I'm not following something. I don't know any pieces. I don't work on pieces so that I could perform. Not that much. Just, if you want me to play something, I'll try something. But that's it.

INTERVIEWER: Great. So we are done with the musical part. Let's move on to the programming part. Basically, the questions are the same. If I remember well, you fill out two on your programming experience. So, would you tell a little bit more about this previous experience?

P2: Yes. You know Reaktor, by Native Instruments?

INTERVIEWER: Yes.

P2: It's a programming environment. A bit like Max MSP, but more object-oriented. The objects are a bit more complex by themselves and more "made". For example, in Max, you can program your oscillator by yourself. But the objects already exist for sine and triangular waves. You can go deeper, but I don't touch that. So I worked a bit with that, trying to do stuff, and I was enjoying myself doing that. And then in college, I took one section of Max MSP. I started one course, but I dropped it.

So I put two because I can do simple things—although I don't remember them right now. I haven't made big patches. I like it, but at some point, I stopped programming because it was taking me so much time that it was distracting me from the music itself. For example, for one piece I would work a bit on the internet and then I would get tired and then just stop. But with the programming, I get obsessed [laughs] just trying to make it work. And then at some point, I realized that the tools I needed—for the most part—already exist. Why am I doing [these tools] myself, since they already exist? Why programming your own sample in Max MSP when I can use KONTAKT?

INTERVIEWER: Okay. So your experience with programming was mostly at university, you took one course.

P2: Yeah, and by myself. For Reaktor, I've looked in the internet, master classes, stuff like that. So I know a bit about it, but it's really on a ground level, starting things. I've made a couple of patches, but they were short that I don't remember any of it. It is complex, it might work for other composers, but I like to keep my intention and energy on the music itself and maybe ask for someone to do the patch. However, I like to work on that, and I like to have ideas with it, but I don't like recreating what already exists in a cheaper and poorer way [laughs].

INTERVIEWER: So with what languages would you have more experience? Reaktor and Max?

P2: Yes. I also tried a bit of cecilia—don't know if you have heard of this one. I tried this cecilia, it was based on C sound, but I found so not intuitive. Once again, there are so many plugins that already exist.

INTERVIEWER: Great. I think you already answered the next one, but I'll ask you anyways in case you have something else to say. Could you briefly describe one project where you need to do programming in music? You mentioned some small patches, is there one that could work as a good example? If you remember...

P2: Yes. I remember one piece that I don't like that much. I wanted to perform with the electric bass and a Max MSP patch. It was quite complex for me at the time [laughs]. I'll just try to remember what [the patch] was doing exactly... [silence] I think we were just triggering stuff with the pedal, which wasn't easy to do with something else. Maybe with that one click of the pedal would trigger sounds, effects, everything. But I'm pretty sure that if I had to do it today I would try to find another way [that is not programming himself, laughs].

INTERVIEWER: Okay. And this particular skill, programming. How did you learn it? you mentioned two courses. Something else?

P2: Yes. When I was younger I loved programming. I tried a bit of basic, in a moment. And [Microsoft] Excel, using macros. I've learned that, I've read lots of books by myself. I think I was kind of good, but I had difficulties. I remember one professor who said to me around seven years ago: "the problem with instrumental composer using the computer is that they get excited

about the computer and forget about the music." At one point, that hit me, and I thought "that's exactly me" [laughs]. I wanted to understand the grammar and just not try stuff when I needed. No. And as a composer, it's the same thing. I did my master using the synthesizer in concert [classical music], and the problem was always the same: I need to know what I am doing in my head before writing something. For example, when you use FM synthesis it is hard to predict what will happen, it's mostly turning the knobs, trying, and finding something you like. This is something quite hard for me. So it's the same thing with programming. I needed to know where I'm going, and learning by a book was the best way for me.

INTERVIEWER: How do you practice this skill? [Pause] You said that you quit programming a while ago, but back then how would you practice—if that question makes sense to you...

P2: I understand. How do I own my skills, and stuff... Hm... Mostly doing when I needed it. Actually, this is maybe one reason why I don't do it right now [laughs]

INTERVIEWER: Okay. Last two questions, these about your past experiences in programming. I would like you to take your time to think about one experience involving programming and music where you felt of one of the following situations and then share this situation with me. The first one is: You felt like struggling with the programming. Like "man, this doesn't make sense". One moment where you realized you were struggling.

P2: One of the biggest homeworks in one of the classes was trying to build my own synthesizer in Max MSP. I wanted it to be not exactly for a project, but as a general tool that would help me for life. I was really struggling with the tools in Max, trying to build something that sounded a bit cheap. It was hard to program that, you can just put an oscillator, and then put a filter in, and it might sound good. I was trying to create that, for example, getting a choice of different waveforms. I was trying to do that and at some point and I'm pretty sure I had a software instrument already [like that, and I was] trying to mimic it. I was struggling and I realized: Well, this software exists already. Or trying to input the time for running, putting the tempo with a time value [to control] an arpeggiator, or an LFO [Low-frequency oscillator], just trying to put them all together was hard for me at the time.

INTERVIEWER: I understand. What about the very opposite. Was there such a moment where you felt that: "Oh, this is intuitive. This is fluid." You felt you were on control the programming. Do you remember any occasion you felt like that?

P2: In control, maybe not because I haven't pushed that far. But I remember trying scripts in Kontakt [sampler software by Native Instruments]. Because I thought the sampler was really cool, sounded great, but I wanted more control. I remember programming some instruments, you could use a scripting language to create more stuff. It's was quite simple programming because the overall structure was already built, so you need to work on something, the variables are very clear. I don't remember exactly what I did, I think it was some micro-tonal stuff that I wanted to try. Hm... I think it makes sense to have an already-solid structure of the software and then you can customize a couple of things.

INTERVIEWER: Thank you. That's all for the interview part.

C.2.3 First meeting

Up to this point, P2 had watched tutorial 1 and had spent a work session playing around with the software.

INTERVIEWER: Here I have the three questions I asked you. The first one was: "How was your working session? What were you trying to achieve?" You answered: "I had more time to try different functions. I worked great and I found a couple of things to experiment further with. I tried mainly to explore different ideas and see where it would take me. I tried different things that I couldn't do with a standard synth. Now, I have a couple of ideas for the final piece." Could you explain a little bit more concretely what you have tried today, providing examples with existing tools?

P2: I started without a precise idea in my mind of what wanted to achieve because I didn't know exactly what I could do [with the software]. In some ways, I still don't know exactly what the software is capable of, because I think that it's pretty much simple the things I've tried so far because of the time.

So what I have tried is using every task to see what was interesting, what was new with it. The first thing that comes to mind that I want to explore more—I'm pretty sure I can do that with something else but here it feels so easy to experiment with—is the duration in the augmenters. You can add a chord or an interval and put the duration for the first notes quite small, for example. I know you could do that [elsewhere], but with a chord... [laughs] I don't know exactly how you would do that in another synth. So, this is something I've worked with.

Also, the X-Y pad of the mouse. To adjust all parameters to work with that [mouse X-Y positions]. I tried a couple of different things, just playing a note and moving to create sounds. Once again, pretty simple thing with that [prototype], pretty easy to set up.

What else did I play with? I tried a couple of things, the idea of making my own basic functions as in a traditional synth. I'm still [thinking about] that and I tried to do that, but during the session, I was telling myself to go away from that, maybe it's not meant to replace a synth. But I still trying to make a bigger sound out of it.

INTERVIEWER: Sorry, just a small clarification. By that "build your own synth" you mean something like Reaktor [modular synthesizer software by Native Instruments]?

P2: [Yes.] Or any synth possible [Arthuria] Prophet, stuff like that, subtractive synth... The main problem I have with Max MSP is that sounds are either simple or... just don't sound that good for me. Maybe they have worked on it, the last version I tried was 5, maybe they changed it. So I think sound tasks, for now, are pretty simple. I wanted to see how I could augment the sound, make it bigger, instead of making just a sine wave, or a sawtooth... Just work to create great sound. One parameter I miss is the reverb [laughs]. [P2 details a bug he found in the system that sounded like a reverb]. I don't understand this so I would like to explore this more. [laughs]. So I think that a next step would be to know [the prototype] a little bit more and try bigger structure because I think what I tried was pretty simple, for now. I mean, it was one hour. [laughs].

INTERVIEWER: Okay, good. I think that this also answers the second question: "What have you tried today in creative terms?"

P2: Yeah.

INTERVIEWER: Okay. So, just to clarify, for you, this working session was purely... music in the first place, right? As far as I understood. What do I mean? You were not trying to use the software in technical terms, that is, to understand what technically what each one of the features was; You were direct to the music. In the first interview, you said the "music first" was a goal for you; the software can't be in the middle of your musical practice. Do you think that this has happened here?

P2: Yeah. Hm...

INTERVIEWER: Okay, this is a bad question. I'll keep myself to the script.

P2: As you want, because I don't mind answering that question. *INTERVIEWER:* Okay, so go on.

P2: Hm... This is a great question. I'm not sure if I can say that music was first because I don't think that what I did today musically was that good [laughs]. Because I was experimenting. Actually, when I was younger I read many manuals for synths, software for music, and I didn't know how it sounded. I did the contrary here. So every time I tried something, I wanted to hear it. To see how it responded in sounds. I'd say both [understand the system technically and musically]. I didn't have to program anything, I could make sound, but at the same time, I wanted to learn the software. I think I was on the keyboard as much as I was on the computer, trying stuff.

INTERVIEWER: Okay, so let's move on to the third question: "Did you face problems today learning and using the system? If so which problem." You said: "I faced no problems today. The system with the functions was pretty simple", and you highlighted 'pretty simple' here...

P2: Yeah.

INTERVIEWER: Then in another point, you said: "the boxes to write the text are a bit small. It made difficult to see where the cursor was to modify the text." Then: "One 'frustrating' feature missing is the ability of swap tasks without resetting the system". What do you mean?

P2: I meant, for example, suppose I'm using an oscillator and a couple of different tasks, and then I want to have a delay. But I want to put the delay before the other tasks. For now, I need

to delete them.

INTERVIEWER: Oh, [rearranging] the tasks pipeline.

P2: Exactly. Maybe because I'm pretty much used to [this feature] in sequencers, there is a way to change it. I miss the ability not to thought over but to change order. Actually, the signal flow in traditional synths is the same thing: you can't change the order. But since I'm on a computer, this thing can be done.

INTERVIEWER: Okay. Apart from these points, no other problem that you found challenging...

P2: No. We can talk a bit more about that. I mean, it's only one hour and fifty minutes since last time so... I think I need more time to see if there is something harder. But for now it was pretty simple, the tutorials were clear. For now, what I see is quite simple. In a good way, and also in a way that can be incomplete. But I'm not sure of the exact power of the software right now. Because I see we are missing transitions and other things. So I see that for now, it's only a portion of what it can do, and this portion is quite easy for now.

INTERVIEWER: Okay. [Interviewer quickly recapitulates some questions] "Do you have any work in progress (for the pieces you will create)? Any sketches? Novel creative explorations since last time? Did you like the result? How could it be improved? Problems & challenges since our last meeting? Any feature hard to understand? Challenges in learning? Any struggle with coding? Accessing certain functionality? Software bugs?" I think we covered it all.

P2: Yeah.

INTERVIEWER: Okay. Thank you.

C.2.4 Second meeting

Up to this point, P2 asked for a high-level summary all functionalities provided by the system, and the experimenter present a summary of tutorials 3, 4 and 5. After, P2 went quickly through these all tutorials. P2 was really excited with the potential of the blackboard, in particular with how easy it was to use musical variables (eg. intervals) inside tasks. P2 spent half an hour presenting

creative possibilities as "his mind was flourishing in ideas". Later, on his second work session, P2 explored some of these possibilities, aiming at creating some small instruments he could perform with.

INTERVIEWER: The same three questions. The first one: "How was your working section? What were you trying to achieve?" And you said: "Great. I wanted to try more advanced and unusual functions. I tried to use more of the blackboard and transitions to find new ideas. See below." Cristal clear, I think, I have no question about this. Second one: "What have you tried today in creative terms?" You said: "Using musical data as variables to change the sound. I used intervals for controlling the amplitude and/or the ratio of the FM synth, or to control the delay time." Also very clear to me. Do you have anything else to complement?

P2: I think in the performance I will explain a little bit more. But the idea is: Usually, software [synthesizers] use computer data, or timers, stuff like that. But the ability to use musical data, for example, intervals or one key or something [is disregarded]. This is promising to me.

INTERVIEWER: Okay, next question: "Did you face problems today in learning or using the system? if so, which problem." You said: "Even with the more advanced features, I still haven't found any problems to learn or use the system. Everything is really well implemented and so easy to use. Different problems appeared in different functions: sound stopped, some effects behaved bizarrely, and so on. But I have no worries it will be fixed." Also, very clear. Could you just detail a little bit more some of these problems? For example, the sound stopped?

P2: Because [the prototype] bugged at some point and I had to restart the system. The delay function—maybe because I was always using intervals, and depending of the order [the sound would change]... [intervals detection] is really fast and it detects everything—felt a bit weird. I don't know exactly what, but for example, the delays at some point the sound would be distorted and stopped—instead of just going on. Why that? I'm pretty sure this is a small problem not with the system but with the task by themselves.

INTERVIEWER: And the feature, the blackboard, the transitions, the tasks... you have no comments so far?

P2: No. It's just so easy that I don't understand why it already doesn't exist in other software systems. It just feels so easy to use. Maybe with more hours to work, I could find something, but for now, it seems really simple and easy to use.

INTERVIEWER: Thank you. We are done with this diary.

[TIME CUT - Before his performances, P2 explained concretely what he had explored later]

P2: I wanted to do is... I like these ramped effects. For example, at one time you have some parameters, and then jumping to another one. I didn't know exactly how to implement that, but what I did with this FM synth is using the musical interval as I play it to control different parameters. In this case, the amplitude of the FM modulation is altered by the interval. For example, if I play an octave [P2 demonstrates], there is this amount of modulation. And as I play different intervals [P2 continuously demonstrates], the amplitude of the FM modulation changes. And if I keep a low note down, it will also work. [P2 demonstrates] For me this is something I find really interesting. As a composer, if I try to work with different intervals, I can change the sound, the music, with these [intervals].

INTERVIEWER: Just a small interruption, a quick question. And you mapped something to the mouse as well, right?

P2: Yes, the ratio of the modulation [on an FM synth] is [controlled] by the mouse right now. I tried to use the ratio with the intervals, but I still haven't figured out the equations, I just don't remember it, so when I tried I think I divided by zero at some point, and the system just crashed the whole thing. That's why, for example... the ratio is quite small [P2 plays]. This is a way to change the ratio just by playing something else. Maybe it might be interesting at some point is to limit the keyboard. [Limit the] playing range of the keyboard to a couple of notes, and keep one octave of the bottom part to just to change the intervals, so that I can use these instead of a fader or something. [P2 plays]

This is one thing I've worked with. I tried that with the delay effect also... [P2 plays another state and shows the problem with the delay, which sometimes stops working after a while]

That's what I said about stopping. [P2 plays]

But the idea is that the delay changes with the performer depending on the piece by itself. For example, if I play two voices... [P2 plays] I like this kind of... I don't know what... I say 'step'. The idea of at one point the delay changes... [P2 plays] This is something I really like, and I wanted to implement, and here it's so easy to play with that... I just wish now intervals were not constrained to one octave so that I could have thirteenth, eleventh... [P2 plays and the delay stops working] I like this kind of bugs [P2 plays] It's random, but I like this random thing... [P2 plays]

INTERVIEWER: This interval thing... I implemented it yesterday, in a rush so, so there will be several issues. It's the first time we are testing this.

P2: No, this is really cool. I think this is my favorite feature right now. For the delay, I would like to try four delays depending on intervals instead of the mouse, timers, and stuff like that. I can think about instrumental composition changing the parameters. This is really cool. I think, for example, Barta's pieces, for example, the pieces for a major second, and in different intervals, to create a different sound. I think it's pretty cool.

Let me try to perform with them...

C.2.5 Retrospective meeting

Both the second work session and the final retrospective session were carried in the same day. P2 performed short excerpts with all the different tools he had developed throughout both work sessions. The interview presented below followed this performance.

[Interviewer compliments the performance and P2 comments]

P2: I'm happy. About the sound, the expression, that the system can have, is really good. I won't say impressed, but I like what I played with. Just the modules by themselves were simple, but I don't think these are the point of your [software]. It's pretty cool what I could create out of... nothing, actually [laughs].

INTERVIEWER: Okay, let's start the retrospective interview. The first question is if you would like to keep on exploring this software in a future—in case it's developed, bug fixes, improvements

on the sound modules?

P2: Yes. I really like, as I said, I could do with that little time [laughs]. I'm really curious to see where the software goes.

INTERVIEWER: Okay. [Explains how the interview unfolds]. I'll start with the level of use. The first question is: In terms of technical usage, do you feel there is still much to learn in this prototype?

P2: No. Don't know for other users, but for me, who had previous experiences not with programming but [with synthesizers], no. Really simple, the tool. I think that in one hour, or a bit more, you can learn about much about what it has, about what it can do. And this time would be dedicated to experiment with [the prototype], not to learn it. I think I've seen pretty much everything. But I don't think I've done everything. Don't know if you understand... I mean, it's easy to use, but... it opens up a world of possibilities to explore. How to experiment with that, though, is quite simple.

INTERVIEWER: When you say "opens up a world of possibilities", you mean the creative potential?

P2: Yes. For example, if I put two variables together, how would that react? Stuff like that. There are a lot of possibilities. But the way to do that, it's simple. I have a list. I can just try them one variable at a time. It's easy.

INTERVIEWER: Okay. Still on the level of use, in learning the system. How do you think this prototype compares to other tools that you've tried in the past? When I say other tools, I include programming tools, such as Reaktor, and also things like Ableton, with knobs, which are not programming languages.

P2: I would say it felt more like a musical instrument that these software. In a sense that, the idea is to help you with the keyboard to play music. I could be meant to program sequences, I suppose, but I don't feel this is the main way to use the software. Whereas with Ableton, you have a lot more of parameters to check, it takes longer... when you open the session, you try different things... Of course, it's easy... [pause] I can't understand necessarily why, but yeah it just feels

it's like a musical instrument—something I might be looking for as a composer. Just to sketch quick ideas instead of "ok, I need to do that, and then that, and then that... Don't know if I could call 'compositions' what I did here, but [the prototype] opens up a world [of possibilities] and it is easy to do. This is pretty much what I think, in my mind, it is different from the others. There was no learning curve, I didn't have to program that if you want it to react... It's simple. [laughs] I repeat that much but I think it's worth mentioning.

INTERVIEWER: Okay. That's it for the level of use questions. Now the question concerning expressivity—what you can do in creative terms [with the prototype]. You already answered that, but I would ask you to add if you have any other thoughts on it. The question is: in terms of creative potential, do you feel there is still much to explore within this prototype?

P2: As it is or with different functionalities implemented?

INTERVIEWER: As it is.

P2: Yeah. I mean, I think there are a couple of parameters in the blackboard I haven't explored... [pause] Sorry, could you repeat the question again, just to make sure...

INTERVIEWER: I want to know if in terms of creative potential—of things you can explore that are different from what you've tried today—if you feel there is still much to explore. It's the same as the first question, about the [technical usage], but we are talking about creative ideas. For example, the very opposite of this would be "no think that with these sessions, I have explored everything that can be done in creative terms with his tool".

P2: I think I've said before... I don't think that in creative terms I've explored everything because there were lots of parameters that I wouldn't have thought that could be used. For example, with the musical intervals, I say it opens up a world of possibilities, but I'm not sure if what I did was... great by itself. I think it needs more refinements on my part, as a composer. I think I've explored... I think it's a good thing every function of the software are pretty... [hesitates] so short amount of time. I think I've seen everything but, for example, I hadn't tried that much the... transitions. For now, I just think of one sound going to another one, but I'm pretty sure there are so many possibilities with that that I haven't explored. So yeah, I think I know what I

can do, but I haven't tried everything.

INTERVIEWER: Okay, I see, I understand. Still on the creative side, on the creative possibilities side. How would you compare this with other existing tools?

P2: Expressive terms... I don't know if there much of a difference because I don't know that much about the other tools.

INTERVIEWER: The ones that you know.

P2: No, I know a couple. The thing is... since it's so easy, you don't have to map, for example, hardware... it's so in your face that way [the prototype] does. And the way the setup is done... I don't know if there is any difference, maybe yes, maybe not... But for me, if I want to be expressive... if I want to play music... The fact I don't have to... implement for two hours or something, it helps the expressivity. Not the tool itself, but the way to use it, the short amount of time it takes, I think might help to sketch stuff. I say sketch but even compose or play music. And the fact it didn't reinvent everything. The fact that the keyboard still works, and of what I can do in other synths, I can do here... I feel myself at home because the main characteristics are there. It's not a new instrument, it's just a way to complement what I can already do. That is cool.

INTERVIEWER: Okay, thank you. One last question. Do you think that you would be able to do the same tools—the ones you've presented me today, the three ones in particular—with the programming tools you have tried in the past?

P2: Hm...

INTERVIEWER: How would that be? How would that compare?

P2: Actually, I know for example that the use of the mouse X-Y pad, I know it's not the first time I use that. I don't use it in other software maybe because it was not that easy... Actually, no, that's a difference since when I open the software the mouse already inputs data instead of having to program, that might help. But I'm pretty sure the first patch, the one I did on the third session... I could be able to to that, maybe with more time, I feel like I did that so easily here than in other software, maybe for just me.

The two other ones—the one using intervals and stuff like that—quickly, I would say no. I couldn't have done that so easily with other ones. Since it's not something that is... common, I think, with other people—if everyone wanted that, it would have been implemented. But I know I could do that, one key minus... but just the fact it is right there... I think I wouldn't have been able to do that so easily. I mean, I'm fascinated by now. If I could have done that with other software, but it just never came to my mind. That's a plus.

INTERVIEWER: Okay, that was all. Thank you.

Appendix D

Supplementary video

This thesis is accompanied by a supplementary video that presents the three software music tools discussed on my thesis: (1) the Voice Reaping Machine, (2) the ZenStates, and (3) the StateSynth.

File name: Supplementary_Video_Thesis_Jeronimo_Barbosa.

Duration: 5 minutes and 18 seconds.

Format: MPEG-4.

Codecs: H.264, AAC.

Size: 45.1 MB.

Bibliography

Aceituno, Jonathan (2015). "Direct and expressive interaction on the desktop: increasing granularity, extent, and dimensionality". PhD Thesis. Université Lille 1, p. 237.

Appert, Caroline and Michel Beaudouin-Lafon (2008). "SwingStates: adding state machines to Java and the Swing toolkit". In: *Software: Practice and Experience* 38.11, pp. 1149–1182. ISSN: 00380644. DOI: 10.1002/spe.867.

Appert, Caroline, Stéphane Huot, Pierre Dragicevic, and Michel Beaudouin-Lafon (2009). "FlowStates: Prototypage d'applications interactives avec des flots de données et des machines à états". In: *Proceedings of the 21st International Conference on Association Francophone d'Interaction Homme-Machine - IHM '09*. New York, New York, USA: ACM Press, p. 119. ISBN: 9781605584614. DOI: 10.1145/1629826.1629845.

Arfib, Daniel, Jean-Michel Couturier, and Loïc Kessous (2005). "Expressiveness and Digital Musical Instrument Design". In: *Journal of New Music Research* 34.1, pp. 125–136. ISSN: 0929-8215. DOI: 10.1080/09298210500124273.

Audry, Sofian (2016). "Aesthetics of Adaptive Behaviors in Agent-Based Art". In: *Proceedings of the 22nd International Symposium on Electronic Art*. Hong Kong, pp. 2–9.

Baguley, Thom (2009). "Standardized or simple effect size: What should be reported?" In: *British Journal of Psychology* 100.3, pp. 603–617. ISSN: 00071269. DOI: 10.1348/000712608X377117.

Bailey, Brian P. and Joseph A. Konstan (2003). "Are informal tools better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design". In: *Proceedings of the conference on Human factors in computing systems - CHI '03*. 5. New York, New York, USA: ACM Press, p. 313. ISBN: 1581136307. DOI: 10.1145/642611.642666.

Bailey, Brian P., Joseph A. Konstan, and John Carlis (2001). "Supporting Multimedia Designers: Towards more Effective Design Tools". In: *Proceedings of Multimedia Modeling*, pp. 267–286.

Baker, Monya (2016). "Statisticians issue warning over misuse of P values". In: *Nature* 531.7593, pp. 151–151. ISSN: 0028-0836. DOI: 10.1038/nature.2016.19503.

Band, Garage (2019). *Official Website*. URL: https://www.apple.com/ca/mac/garageband/ (visited on 04/04/2019).

Barbosa, Jeronimo, Filipe Calegario, Veronica Teichrieb, Geber Ramalho, and Giordano Cabral (2013). "Illusio: A Drawing-Based Digital Music Instrument". In: *NIME '13 Proceedings of the 2013 Conference on New Interfaces for Musical Expression*. Daejeon, Korea Republic, pp. 499–502.

Barbosa, Jeronimo, Filipe Calegario, Veronica Teichrieb, Geber Ramalho, and Patrick McGlynn (2012). "Considering Audience's View Towards an Evaluation Methodology for Digital Musical

Instruments". In: *NIME '12 Proceedings of the 2012 conference on New interfaces for musical expression*. Ann Arbor, Michigan.

Barbosa, Jeronimo, Marcelo M. Wanderley, and Stéphane Huot (2017). "Exploring Playfulness in NIME Design: The Case of Live Looping Tools". In: *NIME '17 Proceedings of the 2017 Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark, pp. 87–92.

— (2018). "ZenStates: Easy-to-Understand Yet Expressive Specifications for Creative Interactive Environments". In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Lisbon, Portugal: IEEE, pp. 167–175. ISBN: 978-1-5386-4235-1. DOI: 10.1109/VLHCC.2018.8506491.

Barkhuus, Louise and Jennifer A. Rode (2007). "From Mice to Men - 24 Years of Evaluation in CHI". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1–16. ISBN: 978-1-59593-593-9. DOI: 10.1145/1240624.2180963.

Beaudouin-Lafon, Michel (2000). "Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces". In: *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '00*. Vol. 2. 1. New York, New York, USA: ACM Press, pp. 446–453. ISBN: 1581132166. DOI: 10.1145/332040.332473.

Beaudouin-Lafon, Michel and Wendy E. Mackay (2000). "Reification, polymorphism and reuse". In: *Proceedings of the working conference on Advanced visual interfaces - AVI '00*. May. New York, New York, USA: ACM Press, pp. 102–109. ISBN: 1581132522. DOI: 10.1145/345513.345267.

Berthaut, Florent, Myriam Desainte-Catherine, and Martin Hachet (2010). "DRILE: an immersive environment for hierarchical live-looping". In: *NIME '10 Proceedings of the 2010 conference on New interfaces for musical expression*. Sydney, Australia, pp. 192–197.

Berthaut, Florent, Sriram Subramanian, Mark T. Marshall, and Martin Hachet (2013). "Rouages: Revealing the Mechanisms of Digital Musical Instruments to the Audience". In: *NIME '13 Proceedings of the 2013 Conference on New Interfaces for Musical Expression*. Daejeon, Korea Republic, pp. 165–169.

Beth Kery, Mary and Brad A. Myers (2017). "Exploring exploratory programming". In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Vol. 62. 3. IEEE, pp. 25–29. ISBN: 978-1-5386-0443-4. DOI: 10.1109/VLHCC.2017.8103446.

Birnbaum, David, Rebecca Fiebrink, Joseph Malloch, and Marcelo M. Wanderley (2005). "Towards a dimension space for musical devices". In: *NIME '05 Proceedings of the 2005 conference on New interfaces for musical expression*. Vancouver, BC, Canada, pp. 192–195.

Blackwell, Alan (2013). "Visual Representation". In: *The Encyclopedia of Human-Computer Interaction*. 2nd. Interaction Design Foundation. Chap. 5.

Blanch, Renaud and Michel Beaudouin-Lafon (2006). "Programming rich interactions using the hierarchical state machine toolkit". In: *Proceedings of the working conference on Advanced visual interfaces - AVI '06*. New York, New York, USA: ACM Press, p. 51. ISBN: 1595933530. DOI: 10.1145/1133265.1133275.

Boberg, Marion, Evangelos Karapanos, Jussi Holopainen, and Andrés Lucero (2015). "PLEXQ: Towards a Playful Experiences Questionnaire". In: *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play - CHI PLAY '15*. New York, New York, USA: ACM Press, pp. 381–391. ISBN: 9781450334662. DOI: 10.1145/2793107.2793124.

Bullock, Jamie, Daniel Beattie, and Jerome Turner (2011). "Integra Live: a new graphical user interface for live electronic music". In: *NIME '11 Proceedings of the 2011 conference on New interfaces for musical expression*. June, pp. 387–392.

Çamcı, Anıl (2018). "GrainTrain : A Hand-drawn Multi-touch Interface for Granular Synthesis". In: *NIME '18 Proceedings of the 2018 Conference on New Interfaces for Musical Expression*, pp. 156–161.

Campbell, D. Murray (2014). "Evaluating musical instruments". In: *Physics Today* 67.4, pp. 35–40. ISSN: 0031-9228. DOI: 10.1063/PT.3.2347.

Card, Stuart K., Jock D. Mackinlay, and George G. Robertson (1991). "A morphological analysis of the design space of input devices". In: *ACM Transactions on Information Systems* 9.2, pp. 99–122. ISSN: 10468188. DOI: 10.1145/123078.128726.

Carlson, Chris (2012). *Borderland Granular Official Website*. URL: http://www.borderlands-granular.com/app/ (visited on 12/20/2018).

Celerier, Jean-Michael, Myriam Desainte-Catherine, and Jean-Michel Couturier (2016). "Graphical Temporal Structured Programming for Interactive Music". In: *Proceedings of the International Computer Music Conference*. Utrecht, Netherlands, pp. 377–379.

Chang, Kerry Shih-Ping and Brad A. Myers (2014). "Creating interactive web data applications with spreadsheets". In: *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. New York, New York, USA: ACM Press, pp. 87–96. ISBN: 9781450330695. DOI: 10.1145/2642918.2647371.

Chirico, Alice, Silvia Serino, Pietro Cipresso, Andrea Gaggioli, and Giuseppe Riva (2015). "When music "flows". State and trait in musical performance, composition and listening: a systematic review". In: *Frontiers in Psychology* 6.June, pp. 1–14. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2015.00906.

Colledanchise, Michele and Petter Ögren (2017). "How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees". In: *IEEE Transactions on Robotics* 33.2, pp. 372–389. ISSN: 1552-3098. DOI: 10.1109/TRO.2016.2633567.

— (2018). *Behavior Trees in Robotics and AI: An Introduction*. 1 edition. Boca Raton: CRC Press. ISBN: 978-1-138-59373-2.

Conversy, Stéphane, Jeremie Garcia, Guilhem Buisan, Mathieu Cousy, Mathieu Poirier, Nicolas Saporito, Damiano Taurino, Giuseppe Frau, and Johan Debattista (2018). "Vizir: A Domain-Specific Graphical Language for Authoring and Operating Airport Automations". In: *The 31st Annual ACM Symposium on User Interface Software and Technology - UIST '18*. New York, New York, USA: ACM Press, pp. 261–273. ISBN: 9781450359481. DOI: 10.1145/3242587.3242623.

Cook, Perry R. (2001). "Principles for designing computer music controllers". In: *NIME '01 Proceedings of the 2001 conference on New interfaces for musical expression*. Seattle, US.

— (2009). "Re-designing principles for computer music controllers: A case study of SqueezeVox Maggie". In: *NIME '09 Proceedings of the 2009 conference on New interfaces for musical expression*. Pittsburgh, PA, pp. 218–221.

Cook, Perry R. (2017). "Expert Commentary: 2001: Principles for Designing Computer Music Controllers". In: *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 1–13. ISBN: 978-3-319-47213-3. DOI: 10.1007/978-3-319-47214-0_1. arXiv: 24022624.

Côté, Jean, Joseph Baker, and Bruce Abernethy (2012). "Practice and Play in the Development of Sport Expertise". In: *Handbook of Sport Psychology*. Hoboken, NJ, USA: John Wiley & Sons, Inc., pp. 184–202. DOI: 10.1002/9781118270011.ch8.

Couturier, Jean-Michel (2002). "A scanned synthesis virtual instrument". In: *NIME '02 Proceedings of the 2002 conference on New interfaces for musical expression*, pp. 43–45.

— (2004). "Utilisation avancée d'interfaces graphiques dans le contrôle gestuel de processus sonores". PhD Thesis. Université Aix-Marseille II.

Csikszentmihalyi, Mihaly (2000). *Beyond Boredom and Anxiety: Experiencing Flow in Work and Play*. 25th. New York, United States: John Wiley & Sons Inc, p. 272. ISBN: 0787951404.

Cuenca, Fredy, Karin Coninx, Davy Vanacken, and Kris Luyten (2015). "Graphical Toolkits for Rapid Prototyping of Multimodal Systems: A Survey". In: *Interacting with Computers* 27.4, pp. 470–488. ISSN: 0953-5438. DOI: 10.1093/iwc/iwu003.

Cumming, Geoff (2014). "The New Statistics". In: *Psychological Science* 25.1, pp. 7–29. ISSN: 0956-7976. DOI: 10.1177/0956797613504966.

Cutumisu, Maria and Duane Szafron (2009). "An Architecture for Game Behavior AI: Behavior Multi-Queues". In: *{AIIDE}*.

Dahl, Luke (2016). "Designing New Musical Interfaces as Research: What's the Problem?" In: *Leonardo* 49.1, pp. 76–77. ISSN: 0024-094X. DOI: 10.1162/LEON_a_01118.

Dannenberg, Roger B. (2018). "Languages for Computer Music". In: *Frontiers in Digital Humanities* 5.November, pp. 1–13. ISSN: 2297-2668. DOI: 10.3389/fdigh.2018.00026.

Dannenberg, Roger B. and Robert Kotcher (2010). "AuraFX: a Simple and Flexible Approach to Interactive Audio Effect-Based Composition and Performance". In: *Proceedings of the International Computer Music Conference*. August, pp. 147–152. ISBN: 0971319286.

Dobrian, Christopher and Daniel Koppelman (2006). "The 'E' in NIME: musical expression with new computer interfaces". In: *NIME '06 Proceedings of the 2006 conference on New interfaces for musical expression*. Paris, France, pp. 277–282. ISBN: 2844263143.

Downie, Marc (2005). "Choreographing the Extended Agent: Performance Graphics for Dance Theater". PhD Thesis. Cambridge, MA: Massachusetts Institute of Technology.

Dragicevic, Pierre (2016). "Fair Statistical Communication in HCI". In: ed. by Judy Robertson and Maurits Kaptein. Human–Computer Interaction Series. Cham: Springer International Publishing, pp. 291–330. ISBN: 978-3-319-26631-2. DOI: 10.1007/978-3-319-26633-6_13.

Dragicevic, Pierre, Fanny Chevalier, and Stéphane Huot (2014). "Running an HCI experiment in multiple parallel universes". In: *Proceedings of the extended abstracts of the 32nd annual ACM conference on Human factors in computing systems - CHI EA '14*. New York, New York, USA: ACM Press, pp. 607–618. ISBN: 9781450324748. DOI: 10.1145/2559206.2578881.

Dragicevic, Pierre and Jean-Daniel Fekete (2004). "Support for input adaptability in the ICON toolkit". In: *Proceedings of the 6th international conference on Multimodal interfaces - ICMI '04*. New York, New York, USA: ACM Press, p. 212. ISBN: 1581139950. DOI: 10.1145/1027933.1027969.

Dragicevic, Pierre, Gonzalo Ramos, Jacobo Bibliowitcz, Derek Nowrouzezahrai, Ravin Balakrishnan, and Karan Singh (2008). "Video browsing by direct manipulation". In: *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*. New York, New York, USA: ACM Press, p. 237. ISBN: 9781605580111. DOI: 10.1145/1357054.1357096.

Engelbart, Douglas C. (1962). *Augmenting Human Intellect: A Conceptual Framework*. SRI Summary Report AFOSR-3223.

Ericsson, K. Anders (2008). "Deliberate practice and acquisition of expert performance: A general overview". In: *Academic Emergency Medicine* 15.11, pp. 988–994. ISSN: 10696563. DOI: 10.1111/j.1553-2712.2008.00227.x. arXiv: 93/S3.00 [0033-295X].

Essl, Georg and Sile O'Modhrain (2006). "An enactive approach to the design of new tangible musical instruments". In: *Organised Sound* 11.03, p. 285. ISSN: 1355-7718. DOI: 10.1017/S135577180600152X.

Farbood, M.M., E. Pasztor, and Kevin Jennings (2004). "Hyperscore: a graphical sketchpad for novice composers". In: *IEEE Computer Graphics and Applications* 24.1, pp. 50–54. ISSN: 0272-1716. DOI: 10.1109/MCG.2004.1255809.

Feit, Anna Maria and Antti Oulasvirta (2014). "PianoText: Redesigning the Piano Keyboard for Text Entry". In: *Proceedings of the 2014 conference on Designing interactive systems - DIS '14*. New York, New York, USA: ACM Press, pp. 1045–1054. ISBN: 9781450329026. DOI: 10.1145/2598510.2598547.

Fels, Sidney (2004). "Designing for Intimacy: Creating New Interfaces for Musical Expression". In: *Proceedings of the IEEE* 92.4, pp. 672–685. ISSN: 0018-9219. DOI: 10.1109/JPROC.2004.825887.

Fels, Sidney, Ashley Gadd, and Axel Mulder (2002). "Mapping transparency through metaphor: towards more expressive musical instruments". In: *Organised Sound* 7.02, pp. 109–126. ISSN: 1355-7718. DOI: 10.1017/S1355771802002042.

Fels, Sidney and Kenji Mase (1999). "Iamascope: a graphical musical instrument". In: *Computers & Graphics* 23.2, pp. 277–286. ISSN: 00978493. DOI: 10.1016/S0097-8493(99)00037-0.

Fiebrink, Rebecca, Ge Wang, and Perry R. Cook (2007). "Don't forget the laptop: Using Native Input Capabilities for Expressive Musical Control". In: *Proceedings of the 7th international conference on New interfaces for musical expression - NIME '07*. New York, New York, USA: ACM Press, p. 164. ISBN: 1355771803. DOI: 10.1145/1279740.1279771. arXiv: 1308.0339.

Fitzmaurice, G.W., Hiroshi Ishii, and Bill Buxton (1995). "Bricks: laying the foundations for graspable user interfaces". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 442–449. ISSN: 00220221. DOI: 10.1145/223904.223964.

Fyans, A. Cavan, Michael Gurevich, and Paul Stapleton (2010). "Examining the spectator experience". In: *NIME '10 Proceedings of the 2010 conference on New interfaces for musical expression*, pp. 1–4.

Garcia, Jérémie, Theophanis Tsandilas, Carlos Agon, and Wendy E. Mackay (2012). "Interactive paper substrates to support musical creation". In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. New York, New York, USA: ACM Press, p. 1825. ISBN: 9781450310154. DOI: 10.1145/2207676.2208316.

Gaver, William W. (2012). "What should we expect from research through design?" In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. November. New York, New York, USA: ACM Press, p. 937. ISBN: 9781450310154. DOI: 10.1145/2207676.2208538.

Gaver, William W., John Bowers, Andrew Boucher, Hans Gellerson, Sarah Pennington, Albrecht Schmidt, Anthony Steed, Nicholas Villars, and Brendan Walker (2004). "The Drift Table: Designing for Ludic Engagement". In: *Extended abstracts of the 2004 conference on Human factors and computing systems - CHI '04*. New York, New York, USA: ACM Press, pp. 885–990. ISBN: 1581137036. DOI: 10.1145/985921.985947.

Ghamsari, Mahtab, Amandine Pras, and Marcelo M. Wanderley (2013). "Combining musical tasks and improvisation in evaluating novel Digital Musical Instruments". In: *Proceedings of the International Symposium on Computer Music Multidisciplinary Research*. Marseille, France, pp. 506–515.

Ghani, Jawaid A. and Satish P. Deshpande (1994). "Task Characteristics and the Experience of Optimal Flow in Human-Computer Interaction". In: *The Journal of Psychology* 128.4, pp. 381–391. ISSN: 0022-3980. DOI: 10.1080/00223980.1994.9712742.

Ghomi, Emilien, Guillaume Faure, Stéphane Huot, Olivier Chapuis, and Michel Beaudouin-Lafon (2012). "Using rhythmic patterns as an input method". In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. New York, New York, USA: ACM Press, p. 1253. ISBN: 9781450310154. DOI: 10.1145/2207676.2208579.

Ghomi, Emilien, Stéphane Huot, Olivier Bau, Michel Beaudouin-Lafon, and Wendy E. Mackay (2013). "Arpège : Learning Multitouch Chord Gestures Vocabularies". In: *ITS '13 Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces*. St. Andrews, Scotland, United Kingdom, pp. 209–218. DOI: 10.1145/2512349.2512795.

Graham, T. C. Nicholas, Leon a. Watts, Gaëlle Calvary, Joëlle Coutaz, Emmanuel Dubois, and Laurence Nigay (2000). "A dimension space for the design of interactive systems within their physical environments". In: *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '00*. New York, New York, USA: ACM Press, pp. 406–416. ISBN: 1581132190. DOI: 10.1145/347642.347799.

Greenberg, Saul and Bill Buxton (2008). "Usability evaluation considered harmful (some of the time)". In: *CHI '08 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, p. 111. ISBN: 9781605580111. DOI: 10.1145/1357054.1357074.

Gurevich, Michael (2016). "Diversity in NIME Research Practices". In: *Leonardo* 49.1, pp. 80–81. ISSN: 0024-094X. DOI: 10.1162/LEON_a_01120.

Harel, David (1987). "Statecharts: a visual formalism for complex systems". In: *Science of Computer Programming* 8.3, pp. 231–274. ISSN: 01676423. DOI: 10.1016/0167-6423(87)90035-9. arXiv: arXiv:1011.1669v3.

Harel, David, H Lachover, A Naamad, A Pnueli, M Politi, R Sherman, A Shtull-Trauring, and M Trakhtenbrot (1990). "STATEMATE: a working environment for the development of complex reactive systems". In: *IEEE Transactions on Software Engineering* 16.4, pp. 403–414. ISSN: 00985589. DOI: 10.1109/32.54292.

Harkins, Paul (2015). "Following The Instruments, Designers, And Users: The Case Of The Fairlight CMI". In: *Journal on the Art of Record Production* 1, pp. 1–3. ISSN: 1754-9892.

Hattwick, Ian (2017). "The Creation of Hardware Systems for Professional Artistic Productions". PhD Thesis. McGill University.

Hempel, Brian and Ravi Chugh (2016). "Semi-Automated SVG Programming via Direct Manipulation". In: *Proceedings of the 29th Annual Symposium on User Interface Software and Technology - UIST '16*. New York, New York, USA: ACM Press, pp. 379–390. ISBN: 9781450341899. DOI: 10.1145/2984511.2984575.

Holland, Simon, Jérémie Garcia, Andrew Johnston, Andrew P. McPherson, Wendy E. Mackay, Marcelo M. Wanderley, Michael D. Gurevich, Tom W. Mudd, Sile O'Modhrain, Katie L. Wilkie, and Joseph Malloch (2016). "Music and HCI". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '16*, pp. 3339–3346. DOI: 10.1145/2851581.2856479.

Holland, Simon, Tom Mudd, Katie Wilkie-McKenna, Andrew P. McPherson, and Marcelo M. Wanderley, eds. (2019). *New Directions in Music and Human-Computer Interaction*. 1st ed. Springer International Publishing, pp. IX, 306. ISBN: 978-3-319-92068-9. DOI: 10.1007/978-3-319-92069-6.

Holland, Simon, Katie Wilkie-McKenna, Paul Mulholland, and Allan Seago, eds. (2013). *Music and Human-Computer Interaction*. Springer-Verlag London, pp. VIII, 292. ISBN: 978-1-4471-2989-9. DOI: 10.1007/978-1-4471-2990-5.

Hunt, Andy and Ross Kirk (2000). "Mapping strategies for musical performance". In: *Trends in Gestural Control of Music*. Ed. by Marcelo M. Wanderley and Marc Battier. Vol. 21. Paris, France: IRCAM, Centre Pompidou, pp. 231–258.

Hunt, Andy, Marcelo M. Wanderley, and Matthew Paradis (2003). "The Importance of Parameter Mapping in Electronic Instrument Design". In: *Journal of New Music Research* 32.4, pp. 429–440. ISSN: 0929-8215. DOI: 10.1076/jnmr.32.4.429.18853.

Hutchins, Edwin L., James D. Hollan, and Donald A. Norman (1985). "Direct Manipulation Interfaces". In: *Human-Computer Interaction* 1.4, pp. 311–338. ISSN: 0737-0024. DOI: 10.1207/s15327051hci0104_2.

Jacobs, Jennifer, Joel Brandt, Radomír Mech, and Mitchel Resnick (2018). "Extending Manual Drawing Practices with Artist-Centric Programming Tools". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. New York, New York, USA: ACM Press, pp. 1–13. ISBN: 9781450356206. DOI: 10.1145/3173574.3174164.

Jacobs, Jennifer, Sumit Gogia, Radomír Mech, and Joel R. Brandt (2017). "Supporting Expressive Procedural Art Creation through Direct Manipulation". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*. New York, New York, USA: ACM Press, pp. 6330–6341. ISBN: 9781450346559. DOI: 10.1145/3025453.3025927.

Jacquemin, Guillaume and Thierry Coduys (2014). "Partitions retroactives avec Iannix". In: *Actes des Journées d'Informatique Musicale JIM 2014*. Bourges, France.

Jenkins, Mark (2007). *Analog Synthesizers: Understanding, Performing, Buying–From the Legacy of Moog to Software Synthesis*. Focal Press; p. 336. ISBN: 9780240520728.

Jensenius, Alexander Refsum, Marcelo M. Wanderley, Rolf Inge Godøy, and Marc Leman (2010). "Musical gestures: concepts and methods in research". In: *Musical gestures: Sound, Movement, and Meaning*. Ed. by Rolf Inge Godøy and Marc Leman. 1st Editio. New York, NY, USA: Routledge. Chap. 2, pp. 12–35. ISBN: 9781135183639. DOI: 10.4324/9780203863411.

Johnson, R. Burke and Anthony J. Onwuegbuzie (2004). "Mixed Methods Research: A Research Paradigm Whose Time Has Come". In: *Educational Researcher* 33.7, pp. 14–26. ISSN: 0013-189X. DOI: 10.3102/0013189X033007014.

Johnston, Andrew (2011). "Beyond Evaluation : Linking Practice and Theory in New Musical Interface Design". In: *NIME '11 Proceedings of the 2011 conference on New interfaces for musical expression*. Oslo, Norway, pp. 280–283.

Jordà, Sergi (2002). "FMOL: Toward User-Friendly, Sophisticated New Musical Instruments". In: *Computer Music Journal* 26.3, pp. 23–39. ISSN: 0148-9267. DOI: 10.1162/014892602320582954.

— (2003). "Interactive music systems for everyone: exploring visual feedback as a way for creating more intuitive, efficient and learnable instruments". In: *Proceedings of the Stockholm Music Acoustics*.

— (2005). "Digital lutherie: crafting musical computers for new musics performance and improvisation". PhD thesis. Universitat Pompeu Fabra.

— (2017). "Expert Commentary: 2003: Sonigraphical Instruments: From FMOL to the reacTable*". In: *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, pp. 89–106. ISBN: 978-3-319-47213-3. DOI: 10.1007/978-3-319-47214-0_7. arXiv: 24022624.

Jordà, Sergi, G. Geiger, Marcos Alonso, and Martin Kaltenbrunner (2007). "The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces". In: *Proceedings of the 1st international Conference on Tangible and Embedded interaction*. ACM, pp. 139–146.

Jordà, Sergi and Sebastián Mealla (2014). "A Methodological Framework for Teaching, Evaluating and Informing NIME Design with a Focus on Expressiveness and Mapping". In: *NIME '14 Proceedings of the 2014 Conference on New Interfaces for Musical Expression*. London, UK, pp. 233–238.

Kazi, Rubaiat Habib, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice (2014). "Draco: Bringing Life to Illustrations with Kinetic Textures". In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*. New York, New York, USA: ACM Press, pp. 351–360. ISBN: 9781450324731. DOI: 10.1145/2556288.2556987.

Keating, Noah H. (2007). "The Lambent Reactive: Exploring the Audiovisual Kinesthetic Playform". In: *NIME '07 Proceedings of the 7th international conference on New interfaces for musical expression*. New York, United States, pp. 530–536.

Kell, Thor (2014). "Musical Mapping of Two-Dimensional Touch-Based Control Layouts". Masters Thesis. McGill University.

Kin, Kenrick, Björn Hartmann, Tony DeRose, and Maneesh Agrawala (2012). "Proton++: A Customizable Declarative Multitouch Framework". In: *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*. New York, New York, USA: ACM Press, p. 477. ISBN: 9781450315807. DOI: 10.1145/2380116.2380176.

Ko, Andrew J., Brad A. Myers, and Htet Htet Aung (2004). "Six Learning Barriers in End-User Programming Systems". In: *2004 IEEE Symposium on Visual Languages - Human Centric Computing*. IEEE, pp. 199–206. ISBN: 0-7803-8696-5. DOI: 10.1109/VLHCC.2004.47.

Ko, Andrew J., Brad A. Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, Susan Wiedenbeck, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, and Henry Lieberman (2011). "The state of the art in end-

user software engineering". In: *ACM Computing Surveys* 43.3, pp. 1–44. ISSN: 03600300. DOI: 10.1145/1922649.1922658.

Krueger, Myron W. (1977). "Responsive environments". In: *Proceedings of the June 13-16, 1977, national computer conference on - AFIPS '77*. New York, New York, USA: ACM Press, p. 423. ISBN: 0750605669. DOI: 10.1145/1499402.1499476.

Kvifte, Tellef and Alexander Refsum Jensenius (2006). "Towards a Coherent Terminology and Model of Instrument Description and Design". In: *NIME '06 Proceedings of the 2006 conference on New interfaces for musical expression*. Paris, France, pp. 220–225. ISBN: 2844263143.

Lawson, Jean-Yves Lionel, Ahmad-Amr Al-Akkad, Jean Vanderdonckt, and Benoit Macq (2009). "An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components". In: *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems - EICS '09*. New York, New York, USA: ACM Press, p. 245. ISBN: 9781605586007. DOI: 10.1145/1570433.1570480.

Levin, Golan (2000). "Painterly Interfaces for Audiovisual Performance". Master's Thesis. Massachusetts Institute of Technology, pp. 1–151.

Live, Ableton (2019). *Official Website*. URL: https://www.ableton.com/ (visited on 04/04/2019).

Lohner, Henning (1986). "The UPIC System: A User's Report". In: *Computer Music Journal* 10.4, p. 42. ISSN: 01489267. DOI: 10.2307/3680095.

Loy, Gareth and Curtis Abbott (1985). "Programming languages for computer music synthesis, performance, and composition". In: *ACM Computing Surveys* 17.2, pp. 235–265. ISSN: 03600300. DOI: 10.1145/4468.4485.

Lucero, Andrés, Evangelos Karapanos, Juha Arrasvuori, and Hannu Korhonen (2014). "Playful or Gameful?" In: *interactions* 21.3, pp. 34–39. ISSN: 10725520. DOI: 10.1145/2590973. arXiv: arXiv:1011.1669v3.

MacLean, Allan, Richard M. Young, and Thomas P. Moran (1989). "Design rationale: the argument behind the artifact". In: *Proceedings of the SIGCHI conference on Human factors in computing systems Wings for the mind - CHI '89*. Vol. 20. SI. New York, USA: ACM Press, pp. 247–252. ISBN: 0897913019. DOI: 10.1145/67449.67497.

Magnusson, Thor (2006). "Affordances and constraints in screen-based musical instruments". In: *Proceedings of the 4th Nordic conference on Human-computer interaction changing roles - NordiCHI '06*. New York, New York, USA: ACM Press, pp. 441–444. ISBN: 1595933255. DOI: 10.1145/1182475.1182532.

— (2010). "Designing Constraints: Composing and Performing with Digital Musical Systems". In: *Computer Music Journal* 34.4, pp. 62–73. ISSN: 0148-9267. DOI: 10.1162/COMJ_a_00026.

— (2017). "Interfacing Sound: Visual Representation of Sound in Musical Software Instruments". In: *Musical Instruments in the 21st Century*. Singapore: Springer Singapore, pp. 153–166. ISBN: 9789811029516. DOI: 10.1007/978-981-10-2951-6_11.

Malloch, Joseph, David Birnbaum, Elliot Sinyor, and Marcelo M. Wanderley (2006). "Towards a new conceptual framework for digital musical instruments". In: *Proceedings of the 9th International Conference on Digital Audio Effects (DAFx-06)*. Montreal, Canada, pp. 49–52.

McCartney, James (2002). "Rethinking the Computer Music Language: SuperCollider". In: *Computer Music Journal* 26.4, pp. 61–68. ISSN: 0148-9267. DOI: 10.1162/014892602320991383.

McPherson, Andrew P., Alan Chamberlain, Adrian Hazzard, Sean McGrath, and Steve Benford (2016). "Designing for Exploratory Play with a Hackable Digital Musical Instrument". In: *Proceedings of the 2016 ACM Conference on Designing Interactive Systems - DIS '16*. New York, USA: ACM Press, pp. 1233–1245. ISBN: 9781450340311. DOI: 10.1145/2901790.2901831.

McPherson, Gary E. (2005). "From child to musician: skill development during the beginning stages of learning an instrument". In: *Psychology of Music* 33.1, pp. 5–35. ISSN: 0305-7356. DOI: 10.1177/0305735605048012.

Morreale, Fabio, Antonella De Angeli, and Sile O'Modhrain (2014). "Musical Interface Design: An Experience-oriented Framework". In: *NIME '14 Proceedings of the 2014 Conference on New Interfaces for Musical Expression*. London, UK, pp. 467–472.

Myers, Brad A., Scott E. Hudson, and Randy Pausch (2000). "Past, present, and future of user interface software tools". In: *ACM Transactions on Computer-Human Interaction* 7.1, pp. 3–28. ISSN: 10730516. DOI: 10.1145/344949.344959.

Myers, Brad A., Sun Young Park, Yoko Nakano, Greg Mueller, and Andrew Ko (2008). "How designers design and program interactive behaviors". In: *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, pp. 177–184. ISBN: 978-1-4244-2528-0. DOI: 10.1109/VLHCC.2008.4639081.

National Association of Music Merchants (2015). *The 2015 NAMM Global Report*. Tech. rep. URL: https://www.namm.org/membership/global-report.

Nijholt, Anton (2014). "Playful Interfaces: Introduction and History". In: *Playful User Interfaces*. Ed. by Anton Nijholt. Gaming Media and Social Effects. Singapore: Springer Singapore, pp. 1–21. ISBN: 978-981-4560-95-5. DOI: 10.1007/978-981-4560-96-2_1.

Noble, Joshua (2012). *Programming Interactivity: A Designer's Guide to Processing, Arduino, and openFrameworks*. 2nd. O'Reilly Media, p. 728. ISBN: 0596154143 9780596154141.

O'Modhrain, Sile (2011). "A Framework for the Evaluation of Digital Musical Instruments". In: *Computer Music Journal* 35.1, pp. 28–42. ISSN: 0148-9267. DOI: 10.1162/COMJ_a_00038.

Oney, Stephen, Brad A. Myers, and Joel Brandt (2012). "ConstraintJS: Programming Interactive Behaviors for the Web by Integrating Constraints". In: *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*. New York, New York, USA: ACM Press, p. 229. ISBN: 9781450315807. DOI: 10.1145/2380116.2380146.

— (2014). "InterState: A Language and Environment for Expressing Interface Behavior". In: *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. New York, New York, USA: ACM Press, pp. 263–272. ISBN: 9781450330695. DOI: 10.1145/2642918.2647358.

Oore, Sageev (2005). "Learning Advanced Skills on New Instruments (or practising scales and arpeggios on your NIME)". In: *NIME '05 Proceedings of the 2005 conference on New interfaces for musical expression*. Vancouver, BC, Canada, pp. 60–65.

Pane, John F., Chotirat "Ann" Ratanamahatana, and Brad A. Myers (2001). "Studying the language and structure in non-programmers' solutions to programming problems". In: *International Journal of Human-Computer Studies* 54.2, pp. 237–264. ISSN: 10715819. DOI: 10.1006/ijhc.2000.0410.

Paternò, Fabio and Volker Wulf, eds. (2017). *New Perspectives in End-User Development*. Cham: Springer International Publishing. ISBN: 978-3-319-60290-5. DOI: 10.1007/978-3-319-60291-2.

Patten, James, Ben Recht, and Hiroshi Ishii (2002). "Audiopad: A Tag-based Interface for Musical Performance". In: *NIME '02 Proceedings of the 2002 conference on New interfaces for musical expression*. Dublin, Ireland, pp. 1–6. ISBN: 1-87465365-8.

Pejrolo, Andrea and Scott B. Metcalfe (2016). *Creating sounds from scratch : a practical guide to music synthesis for producers and composers*. 1st editio. Oxford University Press, p. 327. ISBN: 9780199921898.

Penny, Simon (2000). "Agents as Artworks and Agent Design as Artistic Practice". In: *Advances in {{Consciousness Research}}*. Ed. by Kerstin Dautenhahn. Vol. 19. Amsterdam: John Benjamins Publishing Company, pp. 395–414. ISBN: 978-90-272-5139-8 978-1-55619-435-1 978-90-272-9994-9.

Pennycook, Bruce W. (1985). "Computer-music interfaces: a survey". In: *ACM Computing Surveys* 17.2, pp. 267–289. ISSN: 03600300. DOI: 10.1145/4468.4470.

Perin, Charles (2014). "Direct Manipulation for Information Visualization". PhD Thesis. Université Paris-Sud.

Pickard, Alison Jane (2013). "Major research paradigms". In: *Research Methods in Information*. 2nd. Facet Publishing. Chap. 1, pp. 5–25. ISBN: 978-1-85604-813-2.

Pošcic, Antonio and Gordan Krekovic (2018). "Ecosystems of Visual Programming Languages for Music Creation: A Quantitative Study". In: *Journal of the Audio Engineering Society* 66.6, pp. 486–494. ISSN: 15494950. DOI: 10.17743/jaes.2018.0028.

Poupyrev, Ivan, Michael J. Lyons, Sidney Fels, and Tina Blaine (Bean) (2001). "New interfaces for musical expression". In: *CHI '01 extended abstracts on Human factors in computing systems - CHI '01*. New York, New York, USA: ACM Press, p. 491. ISBN: 1581133405. DOI: 10.1145/634067.634348.

Pro, Logic (2019). *Official Website*. URL: https://www.apple.com/logic-pro/ (visited on 04/04/2019).

Puckette, Miller (1997). "Pure Data : another integrated computer music environment". In: *International Computer Music Conference*, pp. 224–227.

— (2002). "Max at Seventeen". In: *Computer Music Journal* 26.4, pp. 31–43. ISSN: 0148-9267. DOI: 10.1162/014892602320991356.

Rasmussen, Jens (1983). "Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.3, pp. 257–266. ISSN: 0018-9472. DOI: 10.1109/TSMC.1983.6313160. arXiv: arXiv:1011.1669v3.

Reas, Casey and Ben Fry (2006). "Processing Code: Programming within the Context of Visual Art and Design". In: *Aesthetic Computing*. Ed. by Paul A. Fishwick. MIT Press. Chap. 11, p. 476. ISBN: 9780262251617.

Robson, Dominic (2002). "PLAY!: Sound Toys for Non-Musicians". In: *Computer Music Journal* 26.3, pp. 50–61. ISSN: 0148-9267. DOI: 10.1162/014892602320582972.

Russ, Sandra W. (2003). "Play and Creativity: Developmental issues". In: *Scandinavian Journal of Educational Research* 47.3, pp. 291–303. ISSN: 0031-3831. DOI: 10.1080/00313830308594.

Ryan, Joel (1991). "Some remarks on musical instrument design at STEIM". In: *Contemporary Music Review* 6.1, pp. 3–17. ISSN: 0749-4467. DOI: 10.1080/07494469100640021.

Salazar, Spencer (2017). "Sketching Sound: Gestural Interaction in Expressive Music Programming". PhD Thesis. Stanford University.

Salazar, Spencer and Jack Armitage (2018). "Re-engaging the Body and Gesture in Musical Live Coding". In: *NIME '18 Proceedings of the 2018 Conference on New Interfaces for Musical Expression*. Blacksburg, Virginia, USA, pp. 386–389.

Schloss, W. Andrew (2003). "Using Contemporary Technology in Live Performance: The Dilemma of the Performer". In: *Journal of New Music Research* 32.3, pp. 239–242. ISSN: 0929-8215. DOI: 10.1076/jnmr.32.3.239.16866.

Serafin, Stefania, Cumhur Erkut, Juraj Kojs, Niels C. Nilsson, and Rolf Nordahl (2016). "Virtual Reality Musical Instruments: State of the Art, Design Principles, and Future Directions". In: *Computer Music Journal* 40.3, pp. 22–40. ISSN: 0148-9267. DOI: 10.1162/COMJ_a_00372.

Shneiderman, Ben (1983). "Direct Manipulation: A Step Beyond Programming Languages". In: *Computer* 16.8, pp. 57–69. ISSN: 0018-9162. DOI: 10.1109/MC.1983.1654471.

— (2007). "Creativity support tools: accelerating discovery and innovation". In: *Communications of the ACM* 50.12, pp. 20–32. ISSN: 00010782. DOI: 10.1145/1323688.1323689.

Shneiderman, Ben, Catherine Plaisant, Maxine Cohen, and Steven Jacobs (2010). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 6th. Pearson. ISBN: 9780321537355.

Smith, David Canfield, Charles Irby, Ralph Kimball, and Eric Harslem (1982). "The star user interface". In: *AFIPS '82 Proceedings of the June 7-10, 1982, national computer conference*. New York, New York, USA: ACM Press, p. 515. ISBN: 088283035X. DOI: 10.1145/1500774.1500840.

Spiegel, Laurie (1986). *Music Mouse - An Intelligent Instrument*. Tech. rep. New York, New York, USA, p. 44. URL: http://musicmouse.com/mm%7B%5C_%7Dmanual/mouse%7B%5C_%7Dmanual.html.

Stowell, Dan, A. Robertson, Nick Bryan-Kinns, and Mark D. Plumbley (2009). "Evaluation of live human-computer music-making: Quantitative and qualitative approaches". In: *International Journal of Human Computer Studies* 67.11, pp. 960–975. ISSN: 10715819. DOI: 10.1016/j.ijhcs.2009.05.007.

Sutcliffe, Alistair (2009). "Designing for User Engagement: Aesthetic and Attractive User Interfaces". In: *Synthesis Lectures on Human-Centered Informatics* 2.1, pp. 1–55. ISSN: 1946-7680. DOI: 10.2200/S00210ED1V01Y200910HCI005. arXiv: arXiv:1011.1669v3.

Troyer, Akito Van (2012). "DrumTop : Playing with Everyday Objects". In: *NIME '12 Proceedings of the 2012 conference on New interfaces for musical expression*. Ann Arbor, USA, pp. 244–247.

Tsandilas, Theophanis, Catherine Letondal, and Wendy E. Mackay (2009). "Musink: Composing Music through Augmented Drawing". In: *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. New York, New York, USA: ACM Press, p. 819. ISBN: 9781605582467. DOI: 10.1145/1518701.1518827.

Tyson, Michael (2009). *Loopy Official Website*. URL: https://loopyapp.com/ (visited on 12/20/2018).

Ur, Blase, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman (2014). "Practical trigger-action programming in the smart home". In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*. New York, New York, USA: ACM Press, pp. 803–812. ISBN: 9781450324731. DOI: 10.1145/2556288.2557420.

VandenBos, Gary R. (2007). *APA dictionary of psychology*. Washington, DC: American Psychological Association. ISBN: 9781591473800 1591473802.

Veinberg, Anne and Felipe Ignacio Noriega (2018). "Coding with a Piano: The first phase of the CodeKlavier's development". In: *International Computer Music Conference*. Daegu, Korea, pp. 93–98.

Verplank, Bill, Max Mathews, and Rob Shaw (2001). "Scanned Synthesis". In: *Journal of the Acoustical Society of America* 109.5, pp. 2400–2400. DOI: 10.1121/1.4744477.

Vertegaal, Roel, Barry Eaglestone, and Michael Clarke (1994). "An Evaluation of Input Devices for Use in the ISEE Human-Synthesizer Interface". In: *Proceedings of the International Computer Music Conference (ICMC)*. Aarhus, Denmark, pp. 159–162.

Victor, Bret (2012). *Learnable programming*. URL: http://worrydream.com/LearnableProgramming/ (visited on 04/26/2018).

Wagner, Julie, Eric Lecolinet, and Ted Selker (2014). "Multi-finger chords for hand-held tablets". In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*. New York, New York, USA: ACM Press, pp. 2883–2892. ISBN: 9781450324731. DOI: 10.1145/2556288.2556958.

Wanderley, Marcelo M. and Nicola Orio (2002). "Evaluation of Input Devices for Musical Expression: Borrowing Tools from HCI". In: *Computer Music Journal* 26, pp. 62–76. ISSN: 0148-9267. DOI: 10.1162/014892602320582981.

Wang, Ge (2014). "Ocarina: Designing the iPhone's Magic Flute". In: *Computer Music Journal* 38.2, pp. 8–21. ISSN: 0148-9267. DOI: 10.1162/COMJ_a_00236.

— (2016). "Some Principles of Visual Design for Computer Music". In: *Leonardo Music Journal* 26.26, pp. 14–19. ISSN: 0961-1215. DOI: 10.1162/LMJ_a_00960.

Wang, Ge and Perry R. Cook (2004). "On-the-fly Programming: Using Code as an Expressive Musical Instrument". In: *Proceedings of the 2004 International Conference on New Interfaces for Musical Expression*. ISBN: 1461217342.

Webster, Jane and Joseph J. Martocchio (1992). "Microcomputer Playfulness: Development of a Measure with Workplace Implications". In: *MIS Quarterly* 16.2, p. 201. ISSN: 02767783. DOI: 10.2307/249576.

Webster, Jane, Linda Klebe Trevino, and Lisa Ryan (1993). "The dimensionality and correlates of flow in human-computer interactions". In: *Computers in Human Behavior* 9.4, pp. 411–426. ISSN: 07475632. DOI: 10.1016/0747-5632(93)90032-N.

Wessel, David and Matthew Wright (2002). "Problems and Prospects for Intimate Musical Control of Computers". In: *Computer Music Journal* 26.3, pp. 11–22. ISSN: 0148-9267. DOI: 10.1162/014892602320582945.

Woszczynski, Amy B., Philip L. Roth, and Albert H. Segars (2002). "Exploring the theoretical foundations of playfulness in computer interactions". In: *Computers in Human Behavior* 18.4, pp. 369–388. ISSN: 07475632. DOI: 10.1016/S0747-5632(01)00058-9.

Wright, Matthew (2017). "Expert Commentary: Unsolved Problems and Continuing Prospects for Intimate Musical Control of Computers". In: *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*. Ed. by Alexander Refsum Jensenius and Michael J. Lyons. Chap. 2001: Prob, pp. 15–27. DOI: 10.1007/978-3-319-47214-0_2.

Xia, Haijun, Nathalie Henry Riche, Fanny Chevalier, Bruno De Araujo, and Daniel Wigdor (2018). "DataInk: Direct and Creative Data-Oriented Drawing". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. New York, New York, USA: ACM Press, pp. 1–13. ISBN: 9781450356206. DOI: 10.1145/3173574.3173797.

Zabelina, Darya L. and Michael D. Robinson (2010). "Child's play: Facilitating the originality of creative output by a priming manipulation." In: *Psychology of Aesthetics, Creativity, and the Arts* 4.1, pp. 57–65. ISSN: 1931-390X. DOI: 10.1037/a0015644.

Zadel, Mark (2012). "Graphical Performance Software in Contexts : Explorations with Different Strokes". PhD Thesis. McGill University, pp. 1–171.