A Framework and Tools for Mapping of Digital Musical Instruments

Joseph Malloch

Input Devices and Music Interaction Laboratory
Schulich School of Music
McGill University
Montreal, Canada

December 2013

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

2013/12/04

Abstract

Digital musical instruments (DMIs) are typically composed of an interface using some type of sensor technology, and real-time media synthesis algorithms running on a digital computer. The connections between various input signals from performer interaction and the parameters of synthesis must be artificially associated – this *mapping* of gesture to sound or other media defines the behaviour of the system as a whole. Mapping design is a challenging and sometimes frustrating process.

In this dissertation, the design and implementation of an open-source, cross-platform software library and several related tools for supporting the mapping task are presented. These tools are designed to provide discovery and interconnection between parts of DMIs and other interactive systems, and to achieve compatibility through translation and transformation of data representations rather than imposing representation standards. The control parameters of software and hardware devices compliant with libmapper can be freely interconnected without requiring any intended mutual compatibility.

Among the unique features presented is support for mapping between systems that include entities with multiple *instances* with dynamic lifetimes, systems which would usually require bespoke programming. A formalization of the problem is described, and several examples of real-world applications are outlined.

Finally, two use-cases for the mapping tools are presented in-depth: the development of the T-Stick digital musical instrument, and the design and use of prosthetic musical instruments for interactive dance/music performance.

Sommaire

Les instruments de musiques numériques (IMN) comprennent généralement ces éléments: une interface qui comporte certaines technologies de captation, et des algorithmes de synthèse qu'un ordinateur calcule en temps réel. Les connexions entre les signaux engendrés par l'interprète et les différents paramètres de synthèse doivent être établies artificiellement. Cette correspondance entre le geste et le son, ce *mappage*, définit le comportement du système. La conception de tels mappages est un processus exigeant, voire parfois frustrant.

Cette thèse présente la conception et l'implémentation d'un ensemble d'outils logiciels qui servent à faciliter l'élaboration de mappages. La pièce centrale de cette collection est une bibliothèque multiplateforme à code source libre appelée libmapper. Ces outils permettent la détection et l'interconnexion des différentes parties d'IMN ou d'autres systèmes interactifs. Ils visent à assurer la compatibilité par la traduction et la transformation des représentations de données plutôt que par l'imposition de standards. Les paramètres de contrôle des dispositifs logiciels ou matériels qui se conforment à libmapper peuvent être interconnectés librement sans qu'une compatibilité mutuelle ne soit prévue.

La possibilité d'établir des mappages entre des systèmes qui incluent des entités, qui à leur tour comportent de multiples *instances* à durée de vie dynamique, constitue l'une des fonctions uniques qui sont présentées. Ce problème, qui devrait normalement nécessiter une programmation sur mesure, est décrit, formalisé et illustré par des exemples concrets.

Finalement, deux cas où ces outils de mappage ont été utilisés sont analysés en profondeur: le développement du T-Stick, un instrument de musique numérique, ainsi que la conception et l'utilisation d'instruments de musiques prosthétiques pour la danse interactive.

Acknowledgments

First and foremost, I would like to thank my advisor Marcelo Wanderley, who has been an invaluable source of support, encouragement, and insight. I have also had the privilege of studying with the other professors in the Music Technology Area at McGill: Philippe Depalle, Gary Scavone, Stephen McAdams, and Ichiro Fujinaga.

One of the most rewarding aspects of studying in the Input Devices and Music Interaction Lab (IDMIL) – and the Music Technology Group at McGill in general – has been the opportunity to interact with so many interesting people, and from so many different fields of study. In particular, I would like to thank Mark Zadel, Stephen Sinclair, Avrum Hollinger, and Ian Hattwick for discussions, advice, patience and generosity.

Extra thanks go to Stephen Sinclair as co-creator of *libmapper*, not only for his own brilliance and hard work, but for being such a patient resource. Many others have also contributed to the library and surrounding toolset: Mark Zadel for the SuperCollider language bindings, Vijay Rudraraju, Aaron Krajeski, and Jonathan Wilansky on graphical user interfaces, Jérome Nika and Gautam Bhattacharya on some core functionality, Mahtab Ghamsari-Esfahani, Avrum Hollinger and Vanessa Yaremchuk on machine learning tools. Thanks also to D. Andrew Stewart for tireless beta-testing.

Extra thanks also go to my collaborators in creating some wonderful new musical instruments: D. Andrew Stewart and Ian Hattwick. All of the instruments presented in this dissertation were created in the context of large collaborations, and thanks go out to all of the performers, composers, and others involved.

Financial support from the Social Sciences and Humanities Research Council of the Canadian Government was indispensable, and travel funding from CIRMMT enabled me to visit the Univerisade Federal de Minas Gerais in Brazil where I was generously hosted by professors Fernando Rocha and Sergio Freire.

And finally, love and thanks to Alexis and Georgia for making this all worthwhile.

Contribution of Authors

The document is formatted as a manuscript dissertation and includes the following publications:

- Chapter 2: J. Malloch, S. Sinclair, and M. M. Wanderley, "Distributed tools for interactive design of heterogeneous signal networks, " Submitted.
- Chapter 3: J. Malloch, S. Sinclair, and M. M. Wanderley, "Generalized multi-instance control mapping for interactive media systems, " Manuscript prepared for submission.
- Chapter 4: J. Malloch and M. M. Wanderley, "The T-Stick digital musical instrument after eight years of development, " Manuscript prepared for submission.
- Chapter 5: J. Malloch, I. Hattwick and M. M. Wanderley, "Instrumented bodies: prosthetic instruments for Music and Dance," Manuscript prepared for submission.

Chapters 2 and 3 concern the conception and design of an open-source software library implementing a network-based system for mapping control signals for interactive systems. I was responsible for the bulk of the first two versions of the system; the current implementation is developed publicly with most of the contributions from myself and my collaborator Stephen Sinclair. The extensions discussed in chapter 3 were largely conceived and developed by myself, but in consultation with Stephen Sinclair and the community of developers and users subscribed to our mailing list. Coauthor Stephen Sinclair also contributed parts of the manuscript of chapter 2.

The development discussed in chapter 4 was carried-out entirely by myself, but in the context of close collaborations with composer D. Andrew Stewart and performers Fernando Rocha and Xenia Pestova.

The work presented in chapter 5 concerns the conception, design and development of a family of three new digital instruments. I am responsible for the initial design concepts, choice and design of the sensing solutions, and final design and implementation of the "Spine" instruments. My coauthor Ian Hattwick is responsible for refinement and final design of the rib and visor instruments, and also contributed portions of the manuscript.

Contents

List of Figures

List of Tables

List of Acronyms

ABS	Acrylonitrile Butadiene Styrene
AHRS	Attitude and Heading Reference System
API	Application Programming Interface
CAD	Computer-aided design
CRC	Cyclic Redundancy Check
CIRMMT	Centre for Interdisciplinary Research in Music Media and Technology
CLEF	CIRMMT Live Electronics Framework
CRC	Cyclic Redundancy Check
DC	Direct Current
DHT	Distributed Hash Table
DMI	Digital Musical Instrument
DOT	Digital Orchestra Toolbox
FIR	Finite Impulse Response
FM	Frequency Modulation
FSR	Force-Sensing Resistor
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HID	Human Interface Device
IC	Integrated Circuit
IDMIL	Input Devices and Music Interaction Laboratory
IIR	Infinite Impulse Response
IMN	Instruments de musiques numériques
IMU	Inertial Measurement Unit
INS	Inertial Navigation System

IP	Internet Protocol
ITO	Indium Tin Oxide
LED	Light Emitting Diode
MIDI	Musical Instrument Digital Interface
NIME	New Interface(s) for Musical Expression
NTP	Network Time Protocol
OSC	Open Sound Control
PCB	Printed Circuit Board
Pd	Pure Data (programming language/environment)
PET-G	Polyethylene Terephthalate Glycol-modified
PVC	Polyvinyl Chloride
RTP	Real-time Transport Protocol
SDK	Software Development Kit
SLERP	Spherical Linear Interpolation
TCP	Transmission Control Protocol
TUI	Tangible User Interface
UDP	User Datagram Protocol
USB	Universal Serial Bus
VST	Virtual Studio Technology

Chapter 1

Introduction

Digital musical instruments (DMIs) generally consist of a physical or graphical interface generating control signals, a digital synthesizer accepting real-time control, and a layer of "mapping" connecting the two [7]. The design of this mapping layer profoundly affects the behaviour and perception of the instrument and the experience of both performer and audience – the creation of mappings is an essential part of the design process for a new digital instrument and is often revisited when composing a new piece of music. Crucially, mapping designers need to be able to quickly experiment with different mapping configurations while either playing the interface themselves or working closely with collaborating performers.

There are many interesting instruments, input devices, sensors, synthesizers, and tools for mapping. There is, however, a lack of compatibility between these systems: different datatypes and rates, different units and ranges, and different approaches to representation, including dozens of attempts at standardization. Does this lack of compatibility demand further standardization? Unlike existing mapping systems, we argue that adding another standard is not helpful, since even if it is deemed successful not everyone will accept it and further fragmentation will result. Worse, there is artistic interest and novelty in building systems that cannot be easily represented within existing standards and established schemas.

This dissertation focuses on the conception, design, and use of a new kind of mapping system and the implementation of open-source software tools for supporting their use by programmers, designers, composers and performers. These tools enable easy experi-

mentation with cross-connections between input devices, interactive systems, and media synthesizers; and support collaboration between users of different tools and programming languages. Unlike most of the existing tools described below in section 1.4, our tools do not impose any standards as to how systems should be represented or what can be mapped.

1.1 Conceptualization of Digital Musical Instruments

Musical instruments are traditionally classified in several ways, the most famous is the Hornbostel-Sachs taxonomy which classifies instruments based on the type of structure that is vibrating to produce sound [8]. The system includes *idiophones*, *chordophones*, *membranophones*, and *aerophones*, but all electric and electronic instruments are lumped into the category *electrophones/electronophones*, which is not particularly useful when trying to compare and contrast different DMIs[1]. A similar approach is taken by [9] in classifying instruments based on the state of the matter (i.e. solid, liquid, gas, plasma) that is vibrating to produce sound.

A more useful system – for our purposes – considers the interface used to interact with the sound-producing machinery, classifying instruments by whether they have keys, buttons, mouthpieces, or bellows, for example. However, considering that we have many kinds of sensors, and many phenomena that can be sensed, it seems likely that many of the new DMIs will not fit into interface types defined for classical musical instruments.

So what is a digital musical instrument – and how can we understand them in useful contexts? Our basic definition of a DMI is simply based on the fact that phenomena are sensed and sampled digitally, and digital computers are used to synthesize the resulting sound or other media. The system's internal representation of quantities is abstract – there may be semantics associated with a particular signal, but the system must be designed to "tag" the signal with this metadata, they are not an intrinsic property of the samples themselves. All the physically-connected associations from acoustic systems are abolished, and if a performer action is to produce or affect synthesized sound the system must be programmed to make it so. At the most basic level we can consider the DMI system to be composed of *sensing* + *mapping* + *sound synthesis* (figure 1.1).

This broad description of DMIs applies to a wide variety of systems; in order to under-

[1] Technically, the H.-S. system should be categorizing the type of loudspeaker producing the sound of the instrument rather than the instrument itself, since these are separable for DMIs

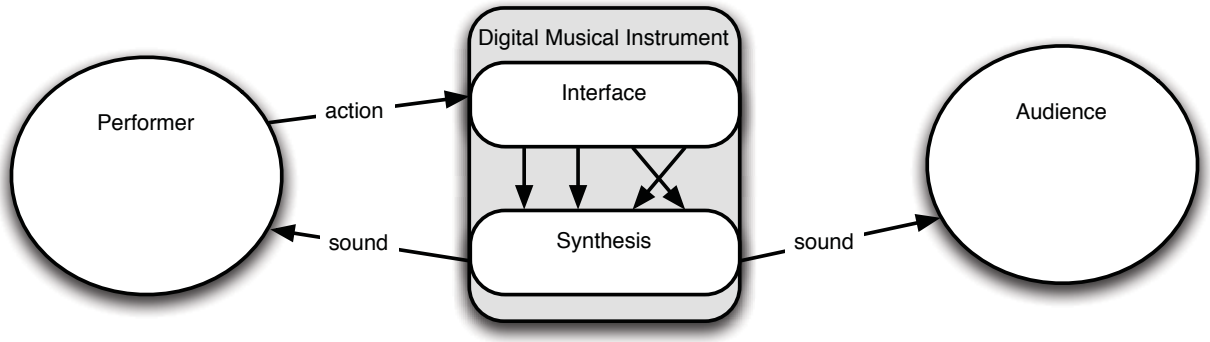

Fig. 1.1 A common (simplistic) representation of the digital musical instrument in context: a performer interacts with the instrument by acting on the interface and receiving the resulting sound, which is also received by an audience. Note that although the DMI is represented as one block in this diagram, it may in fact consist of multiple parts.

stand them more specifically, researchers have proposed a number of relevant metrics and comparisons, which we classify according to their main focus: *object*, *control*, and *agency*. Afterwards, we will briefly touch on the *intentions* of the designer, and communication between performer and audience.

1.1.1 Object

Traditional acoustic instruments are necessarily built from physical materials and cannot possess virtual or modelled parts. The "interface" part of DMIs are also often embodied in physical objects, but we require more flexibility in considering their physical presentation. Several different perspectives of the interface may be useful for comparing DMIs, including:

similarity to traditional instruments – a new DMI might be classified as e.g., an *augmented instrument*, or as *instrument-like*, *instrument-inspired*, or an *alternate gestural controller* [10]

malleability and programmability – in the case of some DMIs, the physical interface itself might not have a fixed shape, and in most cases the response of the instrument is reprogrammable, raising the question "is it still the same instrument when I change the mapping?"

tangibility vs. immersivity – can the interface be touched? Is it embodied in a visible

object - or perhaps in the space around an object? Or is the "instrument" actually an immersive, interactive space? [11] Note that from an HCI or tool-use perspective it may seem inappropriate to call a space an "instrument", but here the word is used as shorthand for "musical instrument" and may carry important cultural associations.

spatial distribution – related to the last point, does the DMI have a location? Perhaps the "instrument" is distributed across multiple locations [12].

object as focus or medium – for certain DMIs, a pre-existing object may be used or repurposed to exploit an audience's nostalgia or cultural contexts, e.g. circuit-bending or "infra-instruments" [13].

1.1.2 Control

The dimensionality of control provided by a DMI is often over-emphasized; it is likely that the relationships between these dimensions [14] and whether they are well-matched to perceived affordances of the system (both interface and sound synthesis) [15] are vastly more important. Considering temporal aspects of control, Schloss considered three levels: *timbral level*, *note level*, and *control over a musical process* [16]; closely-related is Jens Rasmussen's model of human information processing which identifies *signal-*, *rule-*, and *model*-based interaction behaviours [17], which has been applied by ourselves and others to the design and conceptualization of DMIs [18, 19, 20].

1.1.3 Agency

A related concept concerns the balance of *power* between performer and system – unlike acoustic systems, a digital synthesizer can also be programmed to model physical behaviours, make autonomous decisions, or to inject randomness into its internal processes. At one extreme of this continuum we will consider systems that are completely deterministic and possess no autonomy, while at the other end we might find systems with full autonomy. In the latter case, however, it might be more appropriate to refer to the system as a co-performer rather than an instrument.

1.1.4 Designer's Intentions

Almost everything we might say about the instrument design should be prefixed with "intended...", since it is typical that tools for the arts are repurposed and re-imagined by performers, extended in terms of the performance technique to do things that were not initially intended by the designer. Over time, a community's judgement of appropriate or canonical use of the new instrument may change.

User Expertise – Are the instruments intended for use by "expert" performers – implying that is it acceptable to require many hours of practice before achieving any sort of expertise or other reward? Or perhaps for amateur performers, or for casual interaction in the context of a museum installation – in which case it is rather unlikely that the interactor will have the opportunity to practice extensively. The instrument could also be intended for children or for pedagogical use, or perhaps for individuals with physical or mental disabilities who might not be able to play a traditional instrument.

Role of Technology – For many performances involving DMIs, the technology is presented front-and-centre, positioned as an important part of the work rather than the musical result standing on its own. This is not to say that this situation is problematic, but if *novelty* is the most interesting thing about the performance we should acknowledge that there is not likely to be much longevity in the use and appreciation of the instrument.

1.1.5 Expressivity

The term *expressivity* is occasionally used in reference to new instruments (in the NIME conference proceedings, for example) as if it is a property of the interface. We understand this as a basic statement that the designer *intends* for a performer to be able to communicate musically (to express themselves) using the instrument. Discussing the expressivity of an instrument is not useful, since given the proper context a performer could likely express themself using the affordances of *any* interface [21]; meaning is not embedded in the instrument but in the minds of performer and audience. Performer expression is not a raw material that is filtered or mediated by the instrument they are playing, rather the audience observes and perceives the interaction between the performer and their instrument

(see figure 1.2).

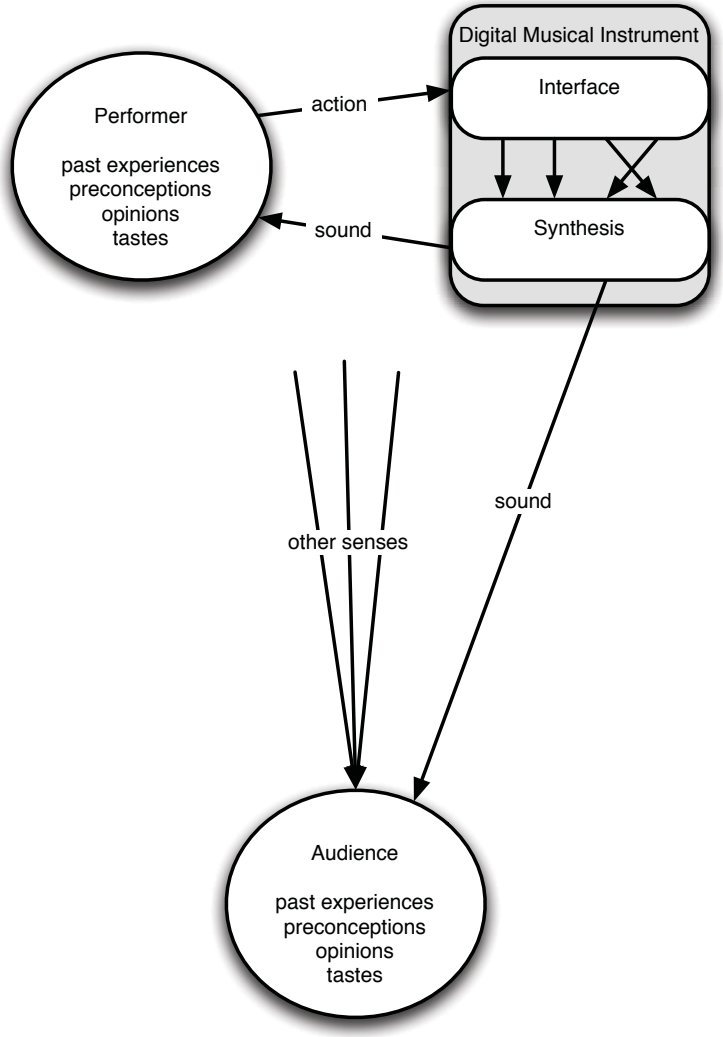

Fig. 1.2 An alternative depiction of the DMI, avoiding possible impressions that the instrument is a conduit from performer to audience.

This implies that any attempt to "reduce barriers" to expression will ultimately fail, since it will make performances less meaningful rather than more so. Since audiences lack contexts in which to understand and evaluate a new instrument and its performance, it is likely that reducing instrument constraints or increasing the degrees of freedom will *slow* the process of understanding for new audiences, since there will be a larger space of performance possibilities.

If an instrument designer wishes to maximize "expression" with their new instrument,

perhaps they should design it to allow communication within existing contexts (e.g., build augmented instruments, instrument-like, and instrument-inspired gestural controllers mentioned above). Alternately, we might consider if there are ways to improve or speed-up the generation of *new* contexts for our new instruments.

1.2 Evaluation of DMIs

The ultimate test for a new DMI must be whether or not it is performed and gains acceptance by some wider community (if this is intended by the designer). Many of the factors for this acceptance involve contexts surrounding the instrument rather than aspects of the design itself, and identification of instrument qualities that might influence success or failure remains largely an open question. Nevertheless, some methods proposed for evaluation of DMIs are presented below.

1.2.1 Quantitative Analysis

Wanderley et al. identify musical tasks appropriate for evaluation tasks, ranging from *isolated tones*, to *scales, arpeggios and complex contours* [22]. They also consider the usability of controllers to be based on four features: *learnability*, *explorability*, *feature controllability*, and *timing controllability*. Many quantitative analysis techniques require such tasks in order to determine a measure of accuracy or error, including all of the following (the first three techniques were suggested by [22] for use in musical tests):

- Fitt's Law: evaluation in terms of time, target width, and linear distance to the target
- Meyer's Law: a refinement of Fitt's Law for movements composed of sub-movements
- Steering Law: used for evaluation of constrained motion along a circular path
- Time on Target: Used for human factors research on anti-aircraft gunnery. Errors which are slightly off-target are penalized as much as dramatic errors [23]
- Cross-correlation of error terms: used for quantifying coordination between dimensions.
- Efficiency: a measure of the length of the path followed divided by the length of the optimal path

- Allocation of Control: This metric combines the spatial Inefficiency metric and the time-based Integrality metric to measure simultaneity of control. [23] uses the *m*-metric for evaluation of *Allocation of Control* in a 6-DOF docking task.

There are also some interesting techniques which do not require a defined task, and could be appropriate for examining unguided musical interaction:

- Integrality: defined as the ratio of the time spent performing integral movements to separable movements [14, 23]. Integral movement exists simultaneously in several dimensions.

- Control Integration: a measure of trajectory slope in degrees, with maximum integration at 45 degrees. Vertegaal used a metric he terms Control Integration to evaluate a mouse, a joystick, and a Nintendo Power Glove for controlling a four DOF timbre space [24, 25].

- Invariance: For applications in which the same gesture is repeated, spatial or temporal invariance can be used for evaluation. This technique does not require a defined task, but works best for movements like walking that are repeated many times [23].

Hunt performed experiments to compare the suitability of three different input devices for controlling musical parameters [26]. The devices consisted of 1) a computer mouse and on-screen sliders, 2) physical sliders, and 3) a "multiparametric" device consisting of the computer mouse and physical sliders used together, but mapped in a complex, integral way. The intent of the experiment was to determine if a *non-analytical* approach to musical control (i.e. with the multiparametric device) would be more successful for certain tasks. Each subject was asked to use each of the devices to replicate 24 sound examples which required either simple discrete changes to one parameter, continuous changes to one parameter at a time, or simultaneous, continuous parameter changes. Automated and expert human analysis of the recorded results indeed found that the subjects performance when reproducing the complex examples was better with the mutiparametric device. Comments in taped interviews also revealed that the subjects found the multiparametric interface more enjoyable and interesting. A longitudinal study was also undertaken, with a reduced number of participants and sound examples. The results of this study indicated strong learning effects for the physical sliders and multiparametric interfaces, but not for the mouse.

1.2.2 Qualitative Analysis

One problem frequently identified with task-oriented evaluation of musical devices or interactions is that people usually do not perform music analytically. Qualitative analysis techniques, such as structured questionnaires and interviews, can be used to extract useful, structured information about experiences, feelings and preferences, and can do so without setting up artificial goals or tasks that may in fact interfere with vital aspects of the interaction (such as enjoyment).

Stowell et al. used a qualitative analytical technique called *Discourse Analysis* for the evaluation of two interfaces for remapping timbre [27, 28]. During testing, the subjects were not given explicit tasks, but instead followed a process of guided exploration followed by a short semi-structured interview to discuss their experiences. Discourse Analysis is a process performed using recordings of the interviews, involving identification of objects, actors, and the relationships between them.

Harrison et al. argue that there is an emerging "third paradigm" in HCI research, distinct from the strong influence from human factors on the one hand, and from cognitive sciences on the other [29]. This paradigm, which they term *situated perspectives*, focuses on investigating and supporting situated action in the real world rather than reducing errors or optimizing efficiency of information transfer. Among other issues, the third paradigm concerns itself with "the domain of non-task-oriented computing, such as ambient interfaces and experience-centred design. These approaches tend to be bad fits to the first and second paradigm, whose methods require problems to be formalized and expressed in terms of tasks, goals and efficiency" [29]. In the third paradigm, the design and/or construction of a musical interface is not a solution to a problem (first paradigm) nor a means to test hypotheses or to generalize knowledge gained (second paradigm). It is instead an element of enquiry, aimed at supporting complex, interesting interaction in the world; in this context ambiguity is celebrated, and enchantment or joy are seen as legitimate criteria of success.

1.3 Mapping

We have established that mapping is a necessary component of a DMI, but how should the creation of the mapping be approached or described? In terms of conception or perception of the mapping design, we must consider its *role* – is the mapping "part of" the instrument,

of a specific composition, or of a performance [30]? Also, is the mapping intended to be *legible* to the audience? To what extent should they understand how parameters are interconnected?

Several less-ambiguous metrics are also useful:

topology – based on the arrangement of connections between parameters, we can consider the mapping to be *one-to-one*, *many-to-one* (or "convergent") in which multiple source parameters are combined somehow to control one destination parameter, *one-to-many* (or "divergent") in which one source parameter affects multiple destination parameters, or *many-to-many*, in which many source and destination parameters are complexly interconnected [31].

static vs. dynamic – does the mapping change over time, and if so over what timescales? Minutes, hours, between performances?

explicit vs. implicit – is the mapping explicitly designed, or "learned" using a machine learning algorithm or other system (e.g., [32]). In the case of implicit mapping, we can also distinguish between *supervised* learning in which associations between input and output are provided to the system (for example, the designer provides both gesture and desired result), and *unsupervised* learning in which the system uses structure found within the performance data alone.

multiple mapping layers – there are several motivations for using multiple layers of mapping between the control interface and synthesis engine, including for parameter abstraction [33], generalizability [34], or for combining multiple mapping approaches.

1.4 Existing Tools for Supporting the Mapping Task

In this section, we will survey existing tools for supporting the mapping process. Since we have described mapping as a hybrid of connectivity and signal processing, this survey is necessarily quite broad, encompassing software platforms, environments, toolboxes, and applications. We will start by examining general tools, and finish with an overview of tools designed for use with specific digital musical instruments. In the interests of brevity, will not consider tools intended for building virtual physical models, which in general are intended for modelling a *fixed* relationship between entities and their virtual environment

– essentially a fixed mapping between input and output – and are thus not relevant to a discussion of tools for designing instrument mappings.

1.4.1 Libraries, Toolboxes and APIs

MnM

MnM (Mapping is not Music) is a toolbox of mapping objects for Max/MSP developed at IRCAM [35]. Based on the vector and matrix processing made possible by FTM, the MnM tools are based on multiple linear regression techniques [36]. Tools for Principle Component Analysis, Gaussian Mixture Models, Hidden Markov Models, and Support Vector Machines are included, along with help patches showing how to easily perform N-to-M dimensional mapping using several techniques. Additionally, tools for matrix multiplication, windowed filters, histograms, and fft and ifft tools are provided. Example patches are also included to demonstrate gesture recognition and gesture-following.

LoM

The *Library of Maps* toolbox (LoM) is a collection of external objects for Max and Jitter implementing multi-dimensional mapping strategies based on geometrical representations of parameter spaces [36]. It includes three main objects, each of which is based on a different geometrical representation:

- `lom.si` implements mapping between an N-dimensional control space and an M-dimensional sound parameter space where N $\leq$ M, based on simplicial complexes created using associated points in both control and sound parameter spaces.
- `lom.multi` implements multilinear interpolation of provided data points. The space created is hyperbolic (curved) and it differentiable across different cells (unlike `lom.si`, in which the surface is non-differentiable across different simplices.
- `lom.rst` implements a global interpolation between stored sound parameter points based on the regularized spline with tension (rst) technique. The number of output parameters is constrained to be N+1 for input space of size N, so this object cannot map from low to high dimension spaces (or vice versa), unlike the other two objects.

All three approaches accept and output parameters in lists, and require sets of control and sound parameter input/output pairs in order to define the dimensionality and the arrangement of the parameter spaces. Unfortunately the LoM toolbox is not currently available, and so couldn't be evaluated further at this time.

hid Toolbox and Mapping Library for Pd

The hid library for Pd is a collection of approximately thirty objects for the Pure Data graphical programming environment, created to facilitate the use of standard Human Interface Devices (HIDs) for controlling software[37]. The library provides an object `hid` for interfacing with generic HIDs, as well as specific objects built on top of it for using joysticks, computer mice, and computer keyboards as input. In addition, Steiner includes objects for processing the data from the HID, including cartesian-polar transformation, filtering, and logarithmic and exponential curves.

One year later, Steiner released and published the Mapping Library for Pd, building on the experience of the hid library [38, 39]. Steiner states that the larger goal of the library (beyond its use to the Pd and DMI communities) is to start a dialogue on the subject of standard *primitives* for mapping, analogous to standard unit generators for audio. In this way the library serves to explore the sort of basic functions that are necessary for mapping. There are approximately 130 objects in the mapping library, coded by Steiner and by Cyrille Henry, including a great number of transfer functions, curves, break-point functions, and control-rate IIR and FIR filters. Interpolation objects and some windowed statistics (mean, median, minima and maxima) are also represented.

Both libraries include objects of the form *one2n* (e.g. `one2two`, `one2three`), modelled on the concept of *divergent mapping* from [40]. Rather than allowing generic divergent mapping, however, these objects simply output multiple, differently-scaled versions of the input and it is unclear what benefit they have over simply using a separate scaling object for each mapping connection.

Since the Mapping Library was developed specifically with the goal of defining and creating a complete set of mapping primitives, it is not surprising that it is so exhaustive. One interesting and perhaps controversial choice made by the authors was to insist on using normalized values (0-1) for nearly all input and output, including for angles and musical pitches. While this can certainly simplify the act of connecting two objects, since there is

no rescaling necessary, this choice can also introduce more confusion than if natural ranges and units are used.

Both the hid and mapping libraries are included in the Pd-extended distribution[2].

Digital Orchestra Toolbox

The Digital Orchestra Toolbox (DOT) is a collection of Max/MSP abstractions (function patches loadable as objects) we developed for mapping digital musical instruments for the McGill Digital Orchestra Project [41]. The toolbox contains more than 100 Max/MSP abstractions, along with help/documentation patches for each one. These tools were built as-needed, rather than to fill a set of mapping primitives like the Mapping Toolbox for Pd, but many of the objects are quite similar. Tools included in the DOT differ from the Mapping Toolbox in their treatment of signal ranges: wherever feasible, real-life units and ranges are used by the tools rather than using normalization.

In the prototype implementations of the mapping tools described in chapter 2, the DOT abstraction `dot.admin` was responsible for all communications between modules, including both session-management and the processing and forwarding of control data. This abstraction has since been removed since it is replaced by new language bindings for the C library *libmapper* encapsulated in external objects build for both Max/MSP and Pure Data.

StreamInput

StreamInput is a cross-platform API for discovery, abstraction, and synchronization of disparate sensor data [42]. The specification for StreamInput is in design phase, coordinated by The Khronos Group and backed by various industry partners. This specification will not be discussed publicly until after it is released, and participation in the standard requires paid membership to the The Khronos Group, however the public website reveals that the system will be graph-based with interconnected nodes for input and filters. All nodes will be timestamped to allow applications to adjust for pipeline delays when combining data from different sensors. Support for motion/position sensors (including "positional sensor fusion"), multi-touch sensors, video cameras, microphone arrays, biometric sensors, and haptic devices is also advertised.

[2] http://puredata.info/

VRPN

The Virtual Reality Peripheral Network (VRPN) is a library and a set of server applications for connecting physical input devices to virtual reality (VR) software applications [43] for real-time interaction and logging. Currently at version 7.3, VRPN supports a wide variety of commercial input devices including motion-capture hardware, Human Interface Devices, and force-feedback peripherals. Generally, a networked PC is tasked with interfacing each input device to the network, and VRPN automatically configures the network and enables connections between the different devices and consuming applications. Messages are timestamped, and can be routed over either UDP or TCP.

Physical interface devices are represented using extensible *device types* matched to a particular functionality and may consist of more than one type. Some common device types are *tracker* (position, orientation, and derivatives), *button*, *dial*, and *ForceDevice* (for interfacing with force-feedback hardware).

Applications designed to consume data from a device type must instantiate a corresponding VRPN application object and provide a callback function – VRPN will call the callback when messages are available from a connected device of the correct type. Connected applications can also send messages to connected devices (e.g. forces, sound) using member functions of the VRPN API[3].

Copperlan

Copperlan is a commercial offering in the space of post-MIDI musical networking[4]. It works over an Ethernet, USB, or Firewire connection rather than requiring dedicated cabling, and provides automatic discovery of connected devices, tunnelling of MIDI data, and global clock synchronization across the network of connected devices. A graphical connection manager application is provided for configuring routing, and software compatibility bridges for connecting existing VST plugins and interfacing with Max/MSP are also available.

Many of the details of CopperLan are not provided to protect intellectual property. Academic and non-commercial users may register for royalty-free licenses, however access to the CopperLan specification and software development kit (SDK) is only possible after signing a restrictive license agreement.

[3] http://www.cs.unc.edu/Research/vrpn/
[4] http://www.copperlan.org/

1.4.2 Platforms and Environments

SenseWorld DataNetwork

The SenseWorld DataNetwork is a client/server framework for sharing data in a collaborative performance or installation environment [44]. The server is implemented in SuperCollider and there are clients available in SuperCollider, Max, PureData and Processing [45]; all communication takes place using Open Sound Control (OSC). Data sources are represented in terms of "nodes," which correspond to wireless motes in the Sense/Stage hardware, and "slots," which correspond to individual data streams within a node (sensor channels on a mote). The source code is released under the open-source GNU/General Public License.

Tapemovie

Tapemovie is a modular patching environment built in Max/MSP, and uses OSC for messaging [46]. It includes various high-level modules, and a "mapper" module which can create and edit mapping connections between parameters. Each mapper object, as with the Jamoma mappers, can only create one connection, and only between two parameters. Data processing capabilities include speedlim, which limits the throughput; linear, logarithmic, and exponential functions; symmetrical functions, ramp time, and range-clipping of both input and output. Smoothing of the data can be performed separately for rising and falling values. The data can be previewed inside the mapper object. The author of Tapemovie is now pursuing the development of a replacement mapping system called SPAN[5].

junXion

junXion v4 is an OSX application from STEIM, built specifically for mapping [47, 48]. It is designed to accept data from Human Interface Devices such as joysticks, MIDI, OSC, WiiRemotes, Arduino, audio or video, and can perform conditional processing and remapping to MIDI or OSC output. In terms of user interface, the user focuses on creating *actions* associated with some input, *tables* that can be applied for look-up processing, and *variables* which can store input values for use in other actions. Actions can result in MIDI or OSC

[5] https://github.com/didascalie-net/span

messages, or they can result in another action. Signal processing available for each action includes smoothing, differentiation, leaky integration, and table look-up. Input and output data streams can be monitored in real-time.

The software is powerful and easy to use judging by the online tutorial videos, it's only weaknesses being a focus on one-to-one mapping (combining inputs is possible but awkward), and fairly weak OSC output in which the messages are constrained to the form `/jXswitch/x/<data>` or `jXcontrol/x/<data>`, depending on whether or not the data are boolean values. Strong semantics in OSC namespaces are to be preferred, especially when dealing with very large numbers of parameters.

Integra

Integra is an open-source software project that aims to both resurrect technologically-obsolete electronic music compositions, and to prevent their future obsolescence (and likewise protect new compositions)[49, 50]. Part of the Integra development effort has been centred on the creation of a new software composition tool (which they envision will not become obsolete). Although this tool is intended for composition, consideration of real-time control has been included from the beginning, since many electronic compositions include interactive elements. Inter-module communication is via OSC, and conceptually the graphical user interface (GUI) is separated from the "engine" where actual sound and data processing takes place. In terms of mapping, the focus has been on displaying and editing inter-module connections in the GUI; like junXion, the assumption is that most users will implement a series of dynamic scenes incorporating a set signal processing and control mapping.

Jamoma

Jamoma is an open-source modular patching framework for Max/MSP with a very active developer community and fast development cycle [51]. Although the original version of Jamoma used normalized values throughout, the influence of the Gesture Description Interchange Format (GDIF) proposal [34] can now be felt in the development, with the release of a *UnitLib* specifically for translating between native units and ranges. *RampLib* and *FunctionLib*, now core components of the Jamoma framework, provide the ability to apply tuneable Cosine, Linear, Lowpass, Power, Tanh, Exponential, or Logarithmic functions to

data streams and to ramps between values.

Jamoma also includes three "mapper" modules: `jmod.mapper`, which allows the user to create multiple connections between module parameters, but fairly simple processing of the mapped data, and the newer `jmod.mapperDiscrete` and `jmod.mapperContinuous`, which only perform one mapping connection per mapper module but offer more sophisticated scaling and warping of the input. A nice user-interface feature that has been recently implemented is to automatically change the background colour of modules that are connected via the mapper so that they match each other, and stand out from the other modules.

MetaMallette

The Meta-Mallette is an environment implemented in Max/MSP for interconnecting modules over OSC [52] . Extensions by Ghomi [53] added separation of control, mapping, and synthesis; and implemented intermediate mapping layers using simulated physics to add dynamic behaviours to the mapping.

Device Server for the Allosphere

The *Device Server* is an application providing a central hub for collecting and distributing real-time data from various input devices [54]. It can connect to devices using MIDI, HID, or VRPN, and was designed for use with the AlloSphere environment[6]. Device Server includes a graphical user interface for interacting with the network, and expressions can be defined using Lua scripting for signal processing for each mapping connection.

Interactive Spaces

Interactive Spaces is a collection of libraries and runtime environment for designing interactive installations [55], implemented by Google's Experience Engineering Team in collaboration with The Rockwell Group's LAB. Implemented in Java with scripting interfaces for javascript and Python, it uses the communications functionality of the Robot Operating System (ROS) for interprocess communication; ROS provides both synchronous and asynchronous data streaming using a publish/subscribe model [56], and is designed for use with a large variety of sensors. The software is a recent project started in July 2012, and is in active development.

[6] http://www.allosphere.ucsb.edu/

OpenInterface

OpenInterface comprises a runtime platform (OpenInterface Kernel) and graphical front-end (SKEMMI) for supporting rapid development of interactive systems [57]. Software modules implementing device drivers, algorithms, etc. can be written in a number of languages, and are made compatible through encapsulation in automatically-generated C++ code. The design aims to provide flexible reconfiguration and iteration during the prototyping of new systems, without enforcing the use of specific devices, interaction techniques, or programming models.

1.4.3 Implicit Mapping Tools

The use of artificial neural networks for mapping musical controls is not new [58, 59, 60] but there has not been a surfeit of tools that make it *easy*. Fiebrink, Cook and Trueman's play-along mapping software [61] does exactly that, allowing the user to simply and quickly train the system by playing along with a recording, which supplies the software with associated inputs and outputs. The software can be set to automatically hand over control to the user once an accuracy threshold has been reached. The software is based on the Wekinator by the same authors, which implements several machine learning techniques, including neural networks, Adaboost, support vector machines, and k-nearest neighbour classification.

1.4.4 Instrument-Specific Mapping Systems

Finally, several commercial DMIs are distributed with dedicated software mapping tools:

Continuum Fingerboard – the *Continuum Editor* is a graphical application for configuring the modular sound synthesizer embedded in the Continuum Fingerboard DMI. It is presented as a 2D patchbay-style matrix, in which the effect of a connection point is determined by a user-defined expression. A graphical function-design interface is provided; expressions can include variables representing the real-time control signals generated by the interface: touch, pressure, and 2D touch position [62].

The Méta-Instrument – *2PIM* is a software application implemented in Max/MSP for calibrating, visualizing and mapping the sensor signals from a variety of interfaces including the MI3 version of the Méta-Instrument [63], Nintendo Wii Remotes, and

graphics tablets. "Profiles" for different synthesizers or signal processing can be implemented as Max/MSP patches and loaded into the application at runtime[7].

Karlax – the *Karlax Bridge* software is also a standalone application implemented in Max-MSP, providing visualization of the signals generated by the instrument and providing drop-down menus for mapping each signal to a specific MIDI device and message. In "note" mode, the control mapping can be configured to alternate between MIDI note-on and note-off messages, or to automatically terminate triggered notes after a given duration. In "control" mode, the mapping can be configured for continuous control over a specified MIDI control-change, or process the data to toggle or ramp between user-defined ranges[8].

Eigenharp – the driver software *EigenD* enables configurable mapping of the sensor signals generated by the instrument to MIDI note and control-change messages[9]. To allow for independent control of multiple notes using control-change messages, "poly" mode automatically distributes new notes over 16 different MIDI channels. *Workbench* is a graphical application enabling configuration of mapping connections using a "patching" metaphor.

1.5 Dissertation Structure

This dissertation concerns the conception and design of a different approach to connectivity of interactive systems – with a specific focus on digital musical instruments intended for expert performance – which prioritizes flexibility of representation rather than compatibility-through-standardization. This system must avoid the constraints of MIDI and post-MIDI representation standards, and allow intercommunication and collaboration between existing tools and languages.

Chapter 2 introduces the central concepts for development of such a system, and describes their implementation as an open-source, cross-platform software library (libmapper) for providing a simple route towards compatibility between the components of digital musical instruments/interactive media systems.

[7] http://www.pucemuse.com/
[8] http://www.dafact.com/
[9] http://www.eigenlabs.com/

This research also aims to support mapping scenarios with multiple ephemeral entities which might dynamically appear and disappear. An obvious musical example is the concept of "notes" – each has a beginning and an end, more than one may sound simultaneously, and there may be long periods in which no notes are sounding at all. There are however many other examples, including the outputs of multitouch interfaces or computer vision systems that track objects of interest (faces, people, "blobs", etc.). Chapter 3 describes our specification for supporting mapping between multi-instance signals, and the implemented extensions to libmapper functionality and programming interfaces. Several examples demonstrating the new functionality are also presented.

In part II, Chapters 4 and 5 present two use cases for the mapping tools presented earlier, in the form of new digital musical instruments. First, the conception and development of the T-Stick DMI over a period of eight years is presented in Chapter 4; to date the T-Stick has been publicly performed at dozens of concerts in seven different countries, with all of the mapping design developed using the tools presented in this dissertation. Next, the more recent design and development of a small family of *prosthetic musical instruments* for musicians and dancers is presented in Chapter 5. While only one public performance tour has utilized the instruments thus far, important segments of their development took place during intensive group workshops for which the use of flexible mapping tools was invaluable.

Finally, general conclusions and future directions for this research are presented in Chapter 6.

Part I

Development of Software Tools for Mapping

Chapter 2

Distributed Tools for Interactive Design of Heterogeneous Signal Networks

The following chapter was submitted as:

J. Malloch, S. Sinclair, and M. M. Wanderley, "Distributed tools for interactive design of heterogeneous signal networks, " Submitted.

2.1 Abstract

We introduce libmapper, an open source, cross-platform software library for flexibly connecting disparate interactive media control systems at run-time. This library implements a minimal, openly-documented protocol meant to replace and improve on existing schemes for connecting digital musical instruments and other interactive systems, bringing clarified, strong semantics to system messaging and description. We use automated discovery and message translation instead of imposed system-representation standards to approach "plug-and-play" usability without sacrificing design flexibility. System modularity is encouraged, and data are transported between peers without centralized servers.

2.2 Introduction

One focus of research in our lab is the development and evaluation of novel "Digital Musical Instruments" (DMI) – using a variety of sensing technologies to provide real-time control over sound synthesis for the purposes of live music performance. In this case, a device may have hundreds of different parameters available for input or output, and the difficult questions arise: which of these parameters should control which? How should they be connected? How can technology and aesthetics help us decide? Especially in collaborative undertakings, who makes these decisions?

These questions are just as applicable to the design of any interactive system for which the precise use-cases are not well defined (here, "... for making music" is not considered a complete specification!). In addition to professional artists and researchers exploring this space, there is also a large and growing community of individuals creating interactive systems with readily-available, low-cost sensors and microcontroller platforms such as Arduino[1]. Mobile telephones now commonly contain a substantial collection of sensors for which real-time data are available: accelerometers, gyroscopes, magnetometers, satellite navigation system receivers, microphones, cameras, and ambient light sensors. The "App" ecosystems surrounding various mobile phone platforms (e.g., iOS, Android) also provide exciting opportunities for developers to experiment with creative, real-time control of media synthesis.

In the field of interactive music, a connection or collection of connections between real-time sensor or gesture data and the inputs of media control is commonly referred to by the noun *mapping* [64]. We also use the term as a verb to refer to the activity or process of designing these relationships, as in "... a tool for mapping of digital musical instruments."

A mapping may be as simple as a single connection, or it may consist of an arbitrary number of complex connections limited only by the constraints of computation, communications bandwidth, or the designer's imagination; it may be explicitly designed or implicitly learned by a machine learning algorithm, which might be guided (supervised) by the preferences of a human designer or might represent structure found in the data alone. In typologies of mapping we also commonly distinguish between *convergent* mapping, in which multiple source parameters are combined to control a single destination parameter, and *divergent* mapping, in which a single source is mapped to control multiple destination

[1] http://www.arduino.cc/

parameters. The system designer may wish for an instrument mapping to be different for different pieces of music, different performances, or indeed within a single piece. Studies have provided evidence that complex mappings may be preferred by performers [65] – this seems to indicate that simple one-to-one mappings, such as the assignment of a single knob to control a particular sound parameter, can be perceived as less interesting to play as compared to mappings which include mixing of controls signals. It follows that a certain amount of iterative experimentation during interaction design is necessary to achieve a balance that is sensible but does not quickly become boring to play.

2.2.1 The Problem

The design of the mapping layer profoundly affects the behaviour and perception of the instrument and the experience of both performer and audience; the creation of mappings is an essential part of the design process for a new digital instrument, and is often revisited when composing a new piece of music for a DMI. Crucially, a mapping designer needs to be able to quickly experiment with different mapping configurations while either playing the interface themselves or working closely with collaborating performers.

There is, however, an a priori lack of compatibility between systems: there is no "natural" mapping between a sensor voltage level and a sound parameter. After digitization, one must deal with different data types and rates, different units and ranges, and different approaches to system representation. In practice this usually means that some fairly extensive programming is necessary in order to make different systems and environments intercommunicate. The mapping designer must consider the control space of both source and destination devices, and explicitly devise a scheme for bridging the two. This takes valuable time that could be spent more creatively on the mapping design itself, and restricts the activity of mapping to the relatively smaller group of artist/programmers.

There are a number of established systems and approaches for representing interactive systems, and for communicating control data; some examples are described below in section 2.2.3. All of these approaches are valid for some systems or scenarios, useful for some users, but does their incompatibility demand further standardization? We argue instead that adding another standard is not helpful, since even if it is deemed successful, not *everyone* will accept it – thus, further fragmentation will result. Worse, there is artistic interest and novelty in building systems that cannot be easily represented within existing standards

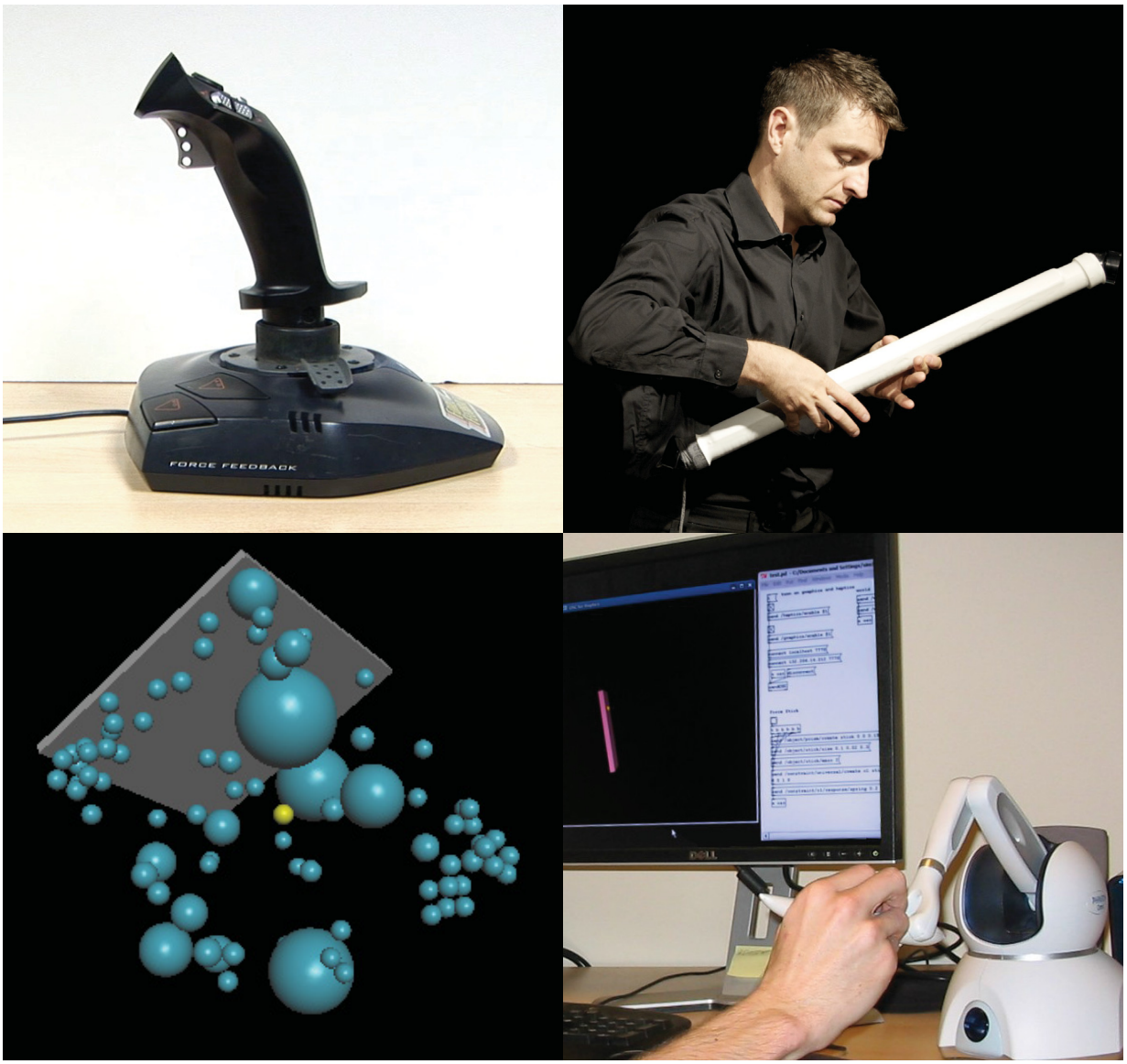

Fig. 2.1 Examples of some of the types of devices we wish to flexibly connect to media synthesizers. Clockwise from top–left: commercial computer input devices such as joysticks, "novel" musical instruments such as the T-Stick [1] (photo: Vanessa Yaremchuk, used with permission), force-feedback/haptic devices, and systems for modelling virtual physical dynamics such as DIMPLE [2].

and established schemas. Since our goal is to support intercommunication of *experimental* interactive systems, imposing further standardization will likely not be helpful.

We argue that what is needed in the research and artistic communities are tools that,

1. allow free reconfiguration and experimentation with the mapping layer during development;

2. provide compatibility between the differing standards (and systems that eschew any standard);

3. and allow the free use of interesting mapping layers between controller and synthesizer (e.g., machine learning, high-dimensional transformations, implicit mapping, dynamic systems, etc.).

Mapping interactive systems is often a time-consuming and difficult part of the design process, and it is appropriate to demand tools and approaches that focus squarely on the mapping task.

2.2.2 Terminology

Here we define some terminology that will aid in describing existing solutions as well as discussing our proposed solution.

Signal

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

Data representation

How a signal's discretized samples are serialized for the purpose of storage or transmission. This could include the data *type* (e.g., floating point, integer), its *bit depth*, *endianess*, and *vector length* if applicable. It might also include an *identifier* or a *name* used to refer to the data, whether it is included with the stored/transmitted data or defined elsewhere in a specification document.

System representation

Further information needed to interpret the data, such as its *range*, *unit*, *coordinate*

system (e.g., Cartesian, polar, spherical, etc.), and any compression applied to the data values (e.g., logarithmic scaling). At a still higher level, the system representation also includes any assumptions or abstractions applied to the control space for the given system. This might include the level of control a parameter addresses, e.g. directly controlling output media vs. controlling a property of a higher-level model, or even whether a given parameter is exposed at all.

As an example, the MIDI standard includes specification of low-level data transport (7– and 14–bit little-endian integers), and a mid-level stream interpretation (e.g., pitch values represent tempered tuning semitone increments with 69 = A4 = 440Hz). It also includes a high-level control abstraction using the concept of a "note" to represent sounds as temporal objects, each with an explicitly defined pitch, beginning and end – something that is often more ambiguous in acoustic systems.

Another familiar example in audio synthesis is the "ADSR" (attack–decay–sustain–release) control model for the evolution of temporal "envelopes" on analog synthesizers. This is not the only way to control envelopes, or even the "correct" way, but it has proven useful enough that it is still used on many hardware and software synths. The very use of envelopes is an abstraction of low-level control to a higher-level (i.e. lower-dimensioned or temporally-compressed) control space, and an important detail to consider when describing the system.

2.2.3 Existing Solutions

Since the early 1980s, the Musical Instrument Digital Interface (MIDI) protocol [66] has been the de facto standard for connecting commercial digital musical instruments. Although it is extremely widespread, it has been argued that MIDI has many failings from the perspective of designers of "alternate" music controllers. To cite some examples, MIDI's bandwidth and data resolution are insufficient [67], messaging semantics are confused [68], its bias to percussive instruments is constraining [69], and it is unsuited for representation of complex control spaces. Although the bandwidth concerns are not strictly tied to the protocol, and extensions such as SKINI have dealt with the limited data types and resolution [70], the protocol's lack of metadata, reliance on normalization, and inability to well-represent control systems substantially different from piano keyboards make it a bad fit for more generalized mapping needs.

In the academic community, Open Sound Control (OSC) [71] has largely supplanted MIDI as the protocol of choice. Unlike MIDI, however, the OSC specification describes only how to properly format and package bytes to form a named OSC message; although this formatting is transport-agnostic, OSC is almost exclusively transported over packet-switching networks as UDP datagrams. OSC is vastly more flexible than MIDI: it includes support for single- and double-precision floating-point numbers, integers, and 64-bit NTP-formatted absolute timestamps. Most importantly, OSC messages are tagged with a user-specifiable, human-readable string instead of a predetermined controller ID number; thus, simultaneously an advantage and disadvantage, OSC messages are not forced into any higher-level hierarchy or schema.

This lack of standardization for specific OSC message names, the so-called OSC *namespace*, means that while OSC-capable hardware or software can decipher the contents of a message originating elsewhere, there is no guarantee that it will be able to make use of it on a semantic level. The result is that most OSC-capable devices use their own custom protocol running on top of OSC designed by individual device developers. Intercommunication between two OSC-capable devices must typically be provided by consulting the documentation for the receiving device, and specifying the IP address, receiving port, OSC path string, and argument format to the sending device.

Although this can be seen as a disadvantage for reasons of inter-device compatibility, it has nonetheless become evident in our experience, (and to the credit of the designers of the OSC protocol) that device-custom naming schemas allow a level of expression that is far easier to understand for humans. To resolve compatibility problems, two approaches are possible: firstly, *normalization* of namespaces and control-space representation to allow automatic interpretation of a set of "known messages"; or secondly, *translation* of messages from one representation to another.

We argue in this work that the latter is a better choice, and translation of representations is the approach adopted by libmapper. Numerous systems for OSC namespace standardization have been proposed [71, 51, 34, 72, 73, 44, 3], but none has yet been widely adopted. We believe that this is symptomatic of the idea that a one-size-fits-all semantic layer is not the right solution. Not only do representation standards risk leaving semantic gaps that force designers to "shoehorn" their data, as is common with MIDI, but we also argue that normalization, while convenient, discards valuable information about the signal being represented. In fact, we believe that this lack of imposed representation standards is

the greatest strength of Open Sound Control over other solutions, and that translation is ultimately a better path to improving compatibility.

2.2.4 Solutions from Other Domains

The (slow) transition from MIDI to OSC mentioned above can be seen as part of a trend in many domains to move legacy dedicated-wire protocols to IP-based systems. In non-musical media control, the legacy DMX512 system for lighting control [74] is gradually being replaced by systems such as the Architecture for Control Networks (ACN) [75]. Meta-data standards for describing the capabilities of devices are often paired with the communication protocol specifications: *Device Description Language* for ACN; *Transducer Electronic Datasheet* for IEEE 1451 [76] and *Transducer Markup Language* [77] for sensor and actuator description. The *Virtual Reality Peripheral Network* (VRPN) [78] functions to allow network-transparent access to control data from various input devices, and also requires a standardized representation of devices.

2.3 Translating representations with libmapper

Our approach to solving the problems of standardization and intercommunication is different than previously proposed standardization/normalization-based approaches. Rather than enforcing conventions in the representation of signals (names, ranges, vector lengths, etc.) we simply provide a minimal layer to help devices describe themselves and their capabilities. One reason we prefer description over standardized representation is that our goal is not precisely *automatic* connectivity but rather *flexible* connectivity. Although libmapper can certainly be used to load previously-designed connections in a production-oriented scenario, we wish to emphasize that libmapper is designed for use in the space of mapping design and exploration, not simply connectivity.

While OSC is currently used by libmapper for transporting data, automatic translation is provided to the namespace expected by the destination. Crucially, this means that the sending application need not know the destination's expected OSC namespace – the destination is responsible for announcing this information, and libmapper takes care of translating the sender's messages into a form expected at the destination. This includes

both translation of the OSC *path*[2] as well as transformation of the data according to some mathematical expression; typically, this is simply a linear scaling, but can be much more complex if desired.

We do not enforce a particular range convention, and in fact explicitly encourage users to avoid arbitrary normalization. This is further encouraged by providing automatic scaling and type coercion between sources and destinations of data. Thus, the endpoints can be represented in the most logical or intuitive way without worrying about compatibility when it comes time to make mapping connections.

Most importantly, we do not presume to tell the user what this "most logical or intuitive way" might be; instead, we simply try to make it easy for the user to represent their systems modularly and with strong semantics. Further, our position is that redundant representations of the signals from different perspectives are often useful (though perhaps for different people or at different times) [34]; by managing connections, libmapper makes it easy to expose large numbers of parameters for mapping without flooding the network with unused data.

Our approach aims to make the mapping task work transparently across programming languages, operating systems, and computers – the user can choose the language or programming environment best suited for the task at hand. From the instrument designer or programmer's point of view, libmapper provides the following services:

- Decentralized resource allocation and discovery
- Description of entities, including extensible metadata
- Flexible, run-time connectivity between nodes
- Interaction with a semantic abstraction of the network (e.g. connecting devices by name rather than setting network parameters)

The first iterations of these tools were developed to meet the needs of a collaborative instrument development project [41]; since then, we have refined the concepts and functionality, reimplemented parts of the system that were written in other languages in transportable C, added bindings for other popular languages, and added utilities for session management, data visualization, recording and playback. We have offered demonstrations

[2]The OSC path refers to the text string identifying the semantics of an OSC message.

and workshops to test our documentation and solicit feedback (e.g. [79]), and used the system for further projects with other universities and industry.

2.4 libmapper concepts

In this section we describe the main features of libmapper, and give our reasoning behind choices made during its conception. The libmapper library itself is used by disparate programs running on a local network, but the collection of these devices can be characterized as a distributed, peer-to-peer communications system based on a representation oriented around named signal streams. It is effectively a metadata, routing and translation protocol built on top of OSC that includes extensive means for describing signals and specifying connections between them over a network.

While the distributed aspect adds complexity as compared to centralized models, this is well-mitigated by a common communications bus and extensive description of signal properties. These make it possible for libmapper programs to find each other automatically, resolve naming conflicts, and make intelligent default decisions during mapping.

2.4.1 Peer to peer communication

As mentioned, for libmapper we have chosen a distributed approach instead of a more centralized network topology. This is not the only choice, and therefore it is necessary to explain our decision.

One possibility for distributing sensor data on a network is the "blackboard" approach: a central server receives data from client nodes, and re-publishes it to receiving clients that request particular signals. This approach is often taken by online media services such as multiplayer online video games, in which a large number of clients must stay synchronized to a common world state. A multi-layer database back-end used to track and synchronize real-time state updates with an "eventual consistency" approach is often employed today for such applications, with a mirrored-server architecture to distribute the load [80]. This has proven to work well, but is a complex solution made necessary by a strong synchronization requirement. Often, ensuring consistency relies on optimistic updates with roll-back to provide perceived fast response times.

Such a centralized method lends itself well to shared environments, however a real-time musical network more typically requires fast, unsynchronized flow of independent signals

from sensors to musical parameters, for which several instances may be present at once – multiple controllers, multiple synthesizers, and multiple performers may share a common studio environment with a single network. MIDI handles this scenario by supporting multiple channels multiplexed over a single serial bus, and, more and more, by providing several MIDI buses and devices in a studio setting, unaware of each other so that they do not interfere. Today, MIDI devices which connect over USB appear to the operating system as their own MIDI device, or even as multiple devices, rather than taking advantage of MIDI's daisy-chaining ability. This means that the user must keep track of which combination of device and channel number represents which physical controller, relying on device drivers to provide meaningful identifiers which are often composed of a concatenation of the product vendor and number. If data from a MIDI device connected to one computer is needed on another, some 3rd-party transport is required, usually specific to the software in use, if such a service is provided at all.

However, an IP-based network actually lends itself well to a peer-to-peer approach, which we have extensively leveraged in the design of libmapper. That is to say, each node on the network can be instructed to send messages to any other node, and any node can issue such an instruction. Additionally, establishment of connections is performed in a stateless manner, and when data transformation is needed, the system is agnostic to where the computation actually takes place.

In our system, data transformation is currently handled by the source node, but since the details of the connection are worked out between the nodes the potential for an alternative agreement is left as a possibility. For example, the receiver could perform all or part of the calculations, or a third-party node could be instructed to process the signal. Computational cost may be considered against hardware capabilities in order to make this decision. This also lends the possibility of agreeing on alternative data transports, such as TCP/IP, or shared memory if the source and destination are on the same host; this latter scenario would be useful for controllers that embed their own audio synthesizer.

2.4.2 A communication bus for "administrative" messages

To enable discovery, it is of course necessary to have a means of communication between all nodes. This is accomplished by having the peer-to-peer data communications run in parallel with a separate bus-oriented architecture for the low-traffic control protocol. Since

our target scenario is several computers cooperating on a single subnet, we have found that multicast UDP/IP technology is ideally suited for this purpose. Multicast is used for example by Apple's Bonjour protocol for tasks such as locating printers on the network [81].

It works by network nodes registering themselves as listeners of a special IP address called the *multicast group*. IP packets sent to this group address are reflected by the Rendezvous Point via the Designated Routers to all interested parties [82]. A small multicast network therefore takes the form of a star configuration, however multicast also allows for more complex multi-hop delivery by means of a Time-To-Live (TTL) value attached to each packet. In the case of libmapper, all nodes listen on a standard multicast group and port, and the TTL is set to 1 by default in order to keep traffic local.

We refer to this multicast port as the "admin bus", since it is used as a channel for "administrative communication": publishing metadata, and sending and acknowledging connection requests. The admin bus is used for resolving name collisions, discovering devices and their namespaces, specifying the initial properties of connections, and making modifications to connection properties. It is important to note that no signal data is communicated on the admin bus; signals consist of OSC messages transmitted (most commonly) by unicast UDP/IP directly between device hosts.

Comparison between multicast and other options

The choice of multicast UDP/IP was not the only possible carrier for this information. Other possibilities include:

1. broadcast UDP messages;
2. a centralized message rebroadcaster;
3. or rebroadcasting of instructions within a mesh network, where each node communicates with a subset of other nodes, and messages are propagated throughout the mesh after several hops.

We chose multicast since, as mentioned, we targeted the use case of a LAN subnet where support for multicast could be guaranteed. Broadcast would also have worked, but we consider multicast to be more "friendly" on a large network, since only interested nodes will receive administrative packets.

The idea of using a mesh is tempting for networks where multicast or broadcast is not available, but the complexity of such an approach was prohibitive and not needed for our applications. However, it is reserved as a future possibility if the need arises. Mesh networks would require the address of a pre-existing node at initialization time, whereas standardizing a multicast port makes it "just work" from the user's point of view. One advantage of mesh networking is to allow the ad-hoc creation of disjoint networks without requiring any special tracking of which multicast ports are available, but use cases for this scenario are, we believe, quite rare.

Another possibility lies between these two extremes: using multicast or broadcast solely for discovery, as in the case of OSCBonjour [83], while sending commands and metadata through a mesh network of reliable TCP connections. Indeed, recent work inclines us towards this solution since we have observed problems of dropped packets when large numbers of signals and devices are present.

The use of multicast has brought to light a distinct lack of support for this useful protocol in OSC-compatible tools. Before developing libmapper as a C library, we added multicast ability to PureData's OSC objects, found bugs in Max/MSP's `net.multi.send` object, and also added multicast to the liblo OSC library, which is used internally by libmapper. With libmapper these improvements were not needed, since libmapper uses liblo directly, and as we provide bindings to libmapper, multicast OSC is handled by the library for all supported bindings. It was nonetheless a useful exercise to provide multicast support in these various environments, and we would like to encourage developers of future audio applications to include the possibility of using it.

Finally, since the information carried by the admin bus is essentially a distributed database, we considered the use of decentralized database technologies such as a distributed hash table (DHT) as used by the BitTorrent protocol [84]. This is certainly an interesting option, but a DHT is best suited for information that is evenly and redundantly distributed throughout a set of nodes. In the case of libmapper, each node maintains its own information, and ostensibly this information is useless if the device is unavailable. Conversely, deleting expired data from a DHT, e.g. when a device goes offline, is unusual, whereas libmapper needs to support the possibility of devices appearing and disappearing from the network. Therefore the idea of devices maintaining their own profile and publishing their own availability seems to be an acceptable approach. On the other hand the idea of one device responding to metadata requests on behalf of another is certainly within libmapper's

possibilities due to the use of multicast, if for example such a strategy is shown to reduce load on less-capable peers.

2.4.3 Comparison with centralized topology

Of course, a centralized organization facilitates or enables certain actions. For example, recording several data sources to a centralized database requires access to all signals at a single point. Likewise, drawing realtime correlations or other statistical analyses on the set or a subset of signals, useful for gesture recognition or trend identification, similarly requires a global view on the data and therefore a centralized approach is best-suited.[3] In our experience, most systems designed for mapping use this client-server approach (e.g., [44]).

However, the use of a central hub is a subset of the possible connection topologies using libmapper's peer-to-peer approach – it is trivial to model such a topology by simply routing all signals through a central libmapper device. This can be done automatically by a program that monitors the network for new connections and re-routes them through itself. It can then record, analyze, or modify any signals before passing them on to the intended destination. As a proof of concept, we make available a small C program called *greedyMapper* that "steals" existing and future network connections and routes them through itself. When the program exits, it returns the mapping network to its previous peer-to-peer topology. This program is not intended for real use, since we find it preferable to simply create a duplicate of each interesting connection, so that the source device sends the signal once to its mapped destination and once to the recording or data-processing device (Figure 2.2).

Extending in this manner by doubling the connections, instead of modifying the network to provide a hub, is a less intrusive means to the same end, and results in the same amount of overall network traffic. In particular, when the central device disconnects, it is not necessary to re-route signals back to their original configuration, creating a potential drop-out period, but is enough to simply disconnect the recording pathways. For the same reason, the sudden disappearance of the central device has far less drastic consequences, in the case of a computer crash or power disruption.

This connection-doubling technique is used by *mapperRec*, a program that automatically

[3]Note that there do exist many decentralized statistical analysis approaches, such as graph-based techniques that can distribute successive reduction steps throughout several computational nodes [85]. Application of such techniques to libmapper may be possible, and is the subject of future work.

duplicates any connections from a device matching a given specification. Ignoring the connection properties, it maps the data untransformed, and records it directly either to a text file, binary file, or to a PostgreSQL database via *OSCStreamDB* [86]. This can be used for later playback, or to convert to formats appropriate to analysis tools.

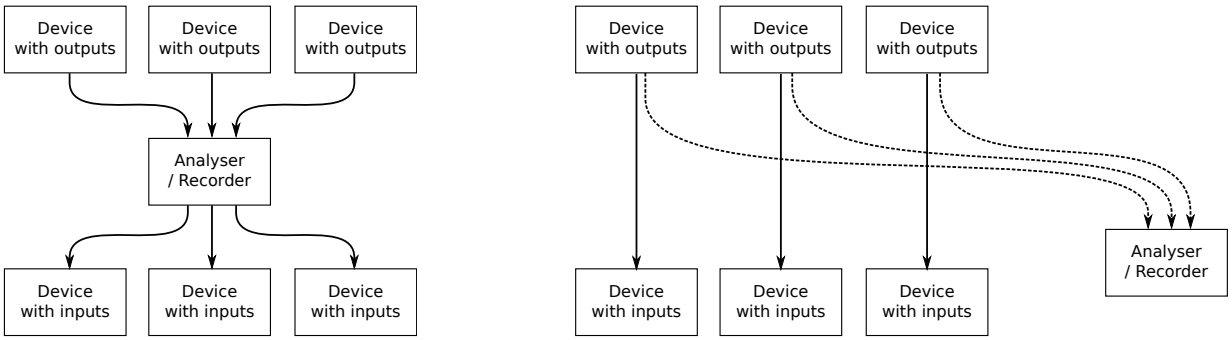

Fig. 2.2 A comparison of network topologies: on the left, data recording or analysis is performed at a central location; on the right, most such scenarios can be handled with lower latency by duplicating the existing connections instead.

2.5 Implementation

Conceptually, libmapper is organized into several components and subcomponents that function together to represent network nodes and their functionality. Most of these components are exposed as user-facing interfaces, however not all components are required by all libmapper programs.

Devices, signals, routers

Firstly, a network node that can send or receive data is called a *device*. This terminology stems from the original usage scenario where a hardware device is controlled by a single libmapper application, however a *device* may just as easily be a software synthesizer, a data transformation service, a bridge to another bus system, or anything else that may send or receive realtime data. Every device has a name as well as an ordinal which is automatically determined to uniquely identify it if other devices with the same name are found on the network.

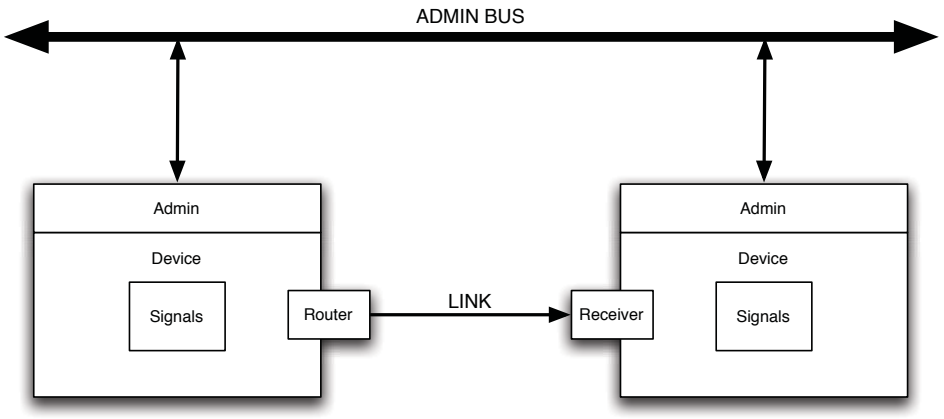

Fig. 2.3 libmapper system components: *devices*, *signals*, *routers*, and *links*. *Connections* are contained within links

A libmapper device must declare its *signals*, which are named value streams available for connecting. Signals have a distinct data direction, i.e. they may either be an input or output, but not both. It is possible in a given device to have one input and one output signal with the same name, however, so this is not a limiting factor. Properties of signals include its name, data type, vector length (see section 2.5.1), optional data range, and optional units.

We also conceive of an entity known as a *router*. Conceptually, a router is an object to which the device sends all of its signal messages, and it handles transforming these messages into the form specified by existing connections, and sends them as required to their destinations. This decouples the device from connection management, with the router maintaining a list of destination addresses and transformation rules for each signal. The reason for this distinction is to allow data processing and translation to potentially take place on other network nodes, either alone or shared with the sending node. A device contains a list of routers, one for each destination device it is linked to, and this is purely an internal concept – application code never needs to interact with a router directly.

Links and Connections

In practise, a router corresponds to a *link* – a network connection created between two devices. Each router contains the address information (IP address and port) required to send data to its destination device, as negotiated by the link-creation protocol.

The router also keeps a list of *connections* associated with each of the device's output

signals mapped to the link's destination device. There is no limit to the number of times a signal may be connected. Connections are specified with information about the source and destination signal names, and any desired data transformation behaviour. In addition to linear scaling, data transformation may include user-defined mathematical expressions, automatic calibration, muting, and clipping (See section 2.6).

Default connection properties: The following steps are used to determine initial connection properties, which have been designed to allow fast experimentation provided the data ranges are well-specified:

1. Any properties that are specified as part of the connection will take precedence.

2. Otherwise, any connection preferences specified by the signals involved will be added. Some signals may have default clipping behaviour to prevent damage to equipment, for example.

3. If the connection processing is still unspecified and the ranges of the source and destination have been provided, processing will default to linear scaling between the input and output ranges.

4. Otherwise no data transformation is provided; data is "passed through" unaffected. (So-called "bypass" mode.)

In all cases, if the types do not match and type coercion is possible, then data types will be automatically transformed. Type coercion follows the rules of the C language: integers are automatically promoted to floating-point numbers, and conversely floating-point numbers are truncated to integers if necessary.

Monitors

We also support network nodes called *monitors* that can send and receive data, but do not have any signals of their own. These nodes are used for observing and managing the network of libmapper-enabled devices, typically in the form of a graphical user interface (see section 2.7). Due to the fact that libmapper administration is performed on a shared bus, an arbitrary number of monitors can be used simultaneously on a given mapping network, and they can be used from any computer in the local network.

A libmapper *device* can also use the monitor functionality, for example responding to remote activity on the network by dynamically adding or removing from its collection of signals, or by automatically creating links and connections to remote devices when they appear on the network.

2.5.1 Data types

Signals are associated with a particular data type, which must be a homogeneous vector of one or more values. Values may be 32- or 64-bit integers, or 32- or 64-bit floating-point numbers, for example.

The choice to support only homogeneous vector types may be seen as inflexible. Indeed, many aspects of libmapper could be adapted to support heterogeneous vectors, however we chose to support only homogeneous vectors because signals are not intended to represent *data structures*, but rather values associated with properties of a system. The reason for introducing vectors is that certain values *are* vectors from a semantic point of view, but heterogeneous types imply something that can be broken up into pieces. We wanted to encourage as much as possible the use of *short* vectors, limited to, for example, 2- or 3D position data, but not used to organize an entire system state vector. Rather, such a structure representing a system state should be split up into its components, each as a separate signal (Figure 2.4).

```
/tuio/2Dobj set s i x y a X Y A m r

/session/object/id/position x y
/session/object/id/angle a
/session/object/id/velocity X Y
/session/object/id/angular_velocity A
/session/object/id/acceleration m
/session/object/id/angular_acceleration r
```

Fig. 2.4 Top: example data encoding specification from the TUIO protocol [3] using a large heterogeneous vector to carry the entire state of an object. Below: the same data exposed as separate "signals" with semantically strong labels and only short, homogenous vectors where appropriate.

To enable flexible mapping design, it is necessary to access as much as possible individual

pieces of information that can be mixed and matched by the user on the fly. Specifying large amounts of data at particular indexes of a large state vector is comparable to "hiding" the natural-language semantic specification enabled by Open Sound Control, and we feel such a choice sacrifices one of the main advantages of OSC over MIDI. Being able to assume homogeneous vector types also allows more succinct expression of element-wise mathematic functions.

Even 3D position data may be usefully represented as separate (x, y, z) components, but some values such as quaternions have terms that are rarely referred to individually. At this point the reader may be asking, why not also support a matrix type? Indeed, why not types for multidimensional tensors? There is certainly a case to be made for higher-dimensioned arrays, such as transmission of pixel data for example, or handling of rotation matrices. However, from a practical standpoint we felt that it was necessary to draw the line somewhere, as libmapper – intended as a lightweight library – cannot provide a full scientific programming language for data processing. Moreover, it is often inefficient to transmit large bundles of data in real-time, and it is preferable to extract properties of this data and expose these as signals.

As an example, a video feed could be transmitted as a 2D matrix signal, but a bare video feed has little use for audio control. It is far more efficient and useful to extract video features such as body or face position using computer vision techniques, and to transmit signals such as "/eye/left/position" as 2-valued vectors.

We believe these arguments are sound, but support for high-dimensioned data is not entirely out of the question, and could perhaps be added in the future if the need arises. In the meantime, matrices can of course be transmitted as flattened vectors, which scalar support alone would not have allowed.

2.5.2 Metadata

Several properties of connections were mentioned in section 2.5. These included the data transformation expression, the connection mode, and the boundary behaviour. Devices and signals also have properties, such as their name, type, length, and range.

In libmapper, this metadata can be extended by the user to include any extra information that may be useful for visualization or analysis of the network. Devices and signals may be extended with named values that can be of any data type supported by OSC.

For example, the user may wish to assign position information to each device in order to identify its physical location in the room. This may aid in the development of a visualization tool for the design of an art installation. Key-value pairs `x=`x and `y=`y may be used for this purpose. In other cases perhaps location is more meaningfully communicated by indicating the name of the room in which a device is situated, or perhaps the name of the person to which a wearable device is attached.

Another example might be to mark certain connections as special, for example because they are related to a recording device. When displaying the connected network topology, it may be desirable to avoid including such connections in order to simplify the visual display. Alternatively names could be used to semantically group certain collections of devices as belonging to a particular user, or being components in a particular subsystem. Since the admin bus is shared by all devices, identification of who made which connection could be important for collaborative scenarios.

2.5.3 Queries

In addition to explicit connections between source and destination signals, machine learning techniques can be used to map *implicitly* between collections of source and destination signals. In the case of supervised techniques, it is necessary to learn the value of the destination signals in order to train the system – information that is not available through unidirectional mapping connections. Often, the value of the destination would have been *set* by the connection in question, but the value could also have been changed by another libmapper peer or locally by a user or by some automated process (as in the case of *play-along learning* [32]).

Two facilities are provided by libmapper for this purpose. Firstly, it is possible to query the value of remote signals. A *monitor* can query remote signals directly, since it has access to information about the mapping network (the existence and network location of the remote signal, for example), but a *device* has knowledge only of its own signals. In the libmapper application programming interface (API), remote values can still be retrieved by calling a remote query function on a local output signal; libmapper will query the remote ends of any mapping connections originating from the specified local output signal and return the number of queries sent out on the network. In order to process the responses to value queries, the local device must register a query response handler function for the local

output signal. The query protocol can also handle the case in which a queried signal does not yet have a value.

Secondly, libmapper also provides the ability to reverse the flow of data on a mapped connection. When this behaviour is enabled, every update of the destination signal, whether updated locally or remotely by another connection, causes the updated value to be sent "upstream" to the connection source. In this case no signal processing is performed other than coercing the data type if necessary. The sampled destination values can then be used to establish interpolation schemes or to calculate errors for supervised training (Figure 2.5).

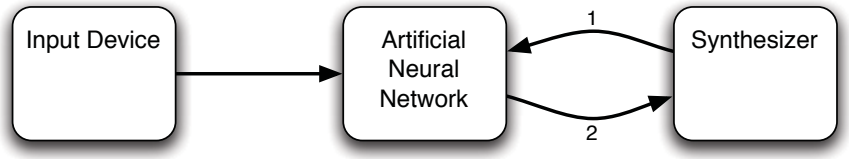

Fig. 2.5 Example "supervised" implicit mapping scenario: connections from an input device are routed through an intermediate device rather than directly to the synthesizer. During training, the values of connected destination input signals are sent upstream to the intermediate device (1) using either individual value queries or "reverse"-mode connections. After training, the connections from intermediate to destination devices are reset to "bypass" mode (2). Note that the arrows marked (1) and (2) actually represent the same data structures; only the dataflow direction changes. A typical implicit mapping scenario might use many such connections rather than the simplification shown here.

2.6 Signal Processing

The first two connection modes ("linear" and "bypass") have already been described in the context of determining default behaviours for new connections. For simple use, the "linear" mode may suffice if the signal ranges have been well defined; for more advanced usage two other options are provided.

The third connection mode is called *calibrate*, and it can only be enabled if the destination range has been defined. While a connection is in this mode, libmapper will keep track of the source signal extrema (minimum and maximum values) and use them to dynamically adjust the linear scaling between source and destination. Re-entering *linear* mode has the effect of ceasing calibration while keeping the recorded extrema as the source range. Ranges are stored independently for each connection, and can be added or edited as part

of the connection metadata; different connections can thus map to different sub-ranges of the destination, for example.

The final connection mode is *expression*, in which the updated source values are evaluated using a user-defined mathematical expression in the form "y=x". This expression can contain arithmetic, comparison, logical, or bitwise operators. The linear and calibrate modes automatically populate the expression metadata, so that when switching into expression mode the user can start by editing the expression defining the previous mapping.

In addition to the active modes mentioned, each connection can be independently muted, allowing users to temporarily prevent data transmission without losing the connection state.

2.6.1 Indexing delayed samples

Past samples of both input and output are also available for use in expressions, allowing the construction of FIR and IIR digital filters (Figure 2.6). These values are accessed using a special syntax; some examples are shown in Table 2.6.1. We currently limit addressing to a maximum of 100 samples in the past.

Function	Expression Syntax
Differentiator	`y = x - x{-1}`
Integrator	`y = x + y{-1}`
Exponential moving average	`y = x * 0.01 + y{-1} * 0.99`
Counter	`y = y{-1} + 1`

Table 2.1 Some example expressions using indexing of delayed samples.

2.6.2 Boundary Actions

A separate "boundary" stage is provided for constraining the output range after the expressions are evaluated. Actions can be controlled separately at the minimum and maximum boundaries provided, and can take one of five different modes:

none values outside of the bound are passed through unchanged.

clamp values outside the bound are constrained to the bound.

mute values outside the bound are not passed to the output.

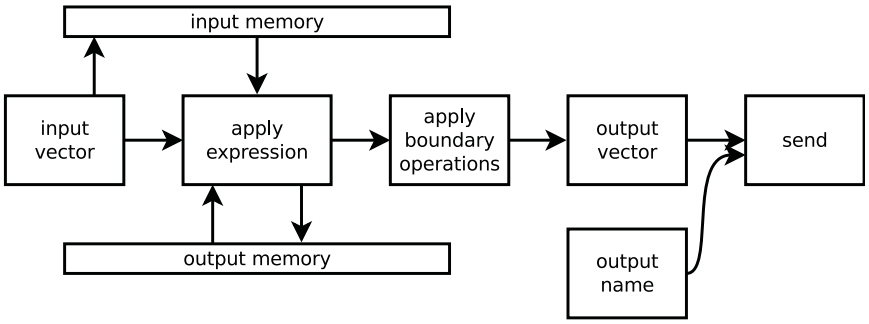

Fig. 2.6 Processing pipeline used by libmapper.

wrap values outside the bound are "wrapped" around to the other bound.

fold values outside the bound are reflected back towards the other bound

2.7 Mapping Session Management

When designing mappings between libmapper devices it is necessary to interact somehow with the network (via the admin bus); typically this is done using one of the graphical user interfaces (GUI) we have developed in parallel with the library itself. These interfaces are also used to save and load pre-prepared mappings, for example when performing a piece in concert.

2.7.1 Graphical User Interfaces

Currently our working GUIs use a *bipartite graph* representation of the connections, in which sources of data appear on the left-hand side of the visualization and destinations or sinks for data appear on the right (Figure 2.7). Lines representing inter-device *links* and inter-signal *connections* may be drawn between the entities on each side, and properties are set by first selecting the connection(s) to work on and then setting properties in a separate "edit bar". They use a multi-tab interface in which the leftmost tab always displays the network overview (links between devices) and subsequent tabs provide sub-graph representations of the connections belonging to a specific linked device.

We have also explored alternative visualization and interaction techniques, which allow more informed and flexible interaction with the mapping network. Crucially, we believe

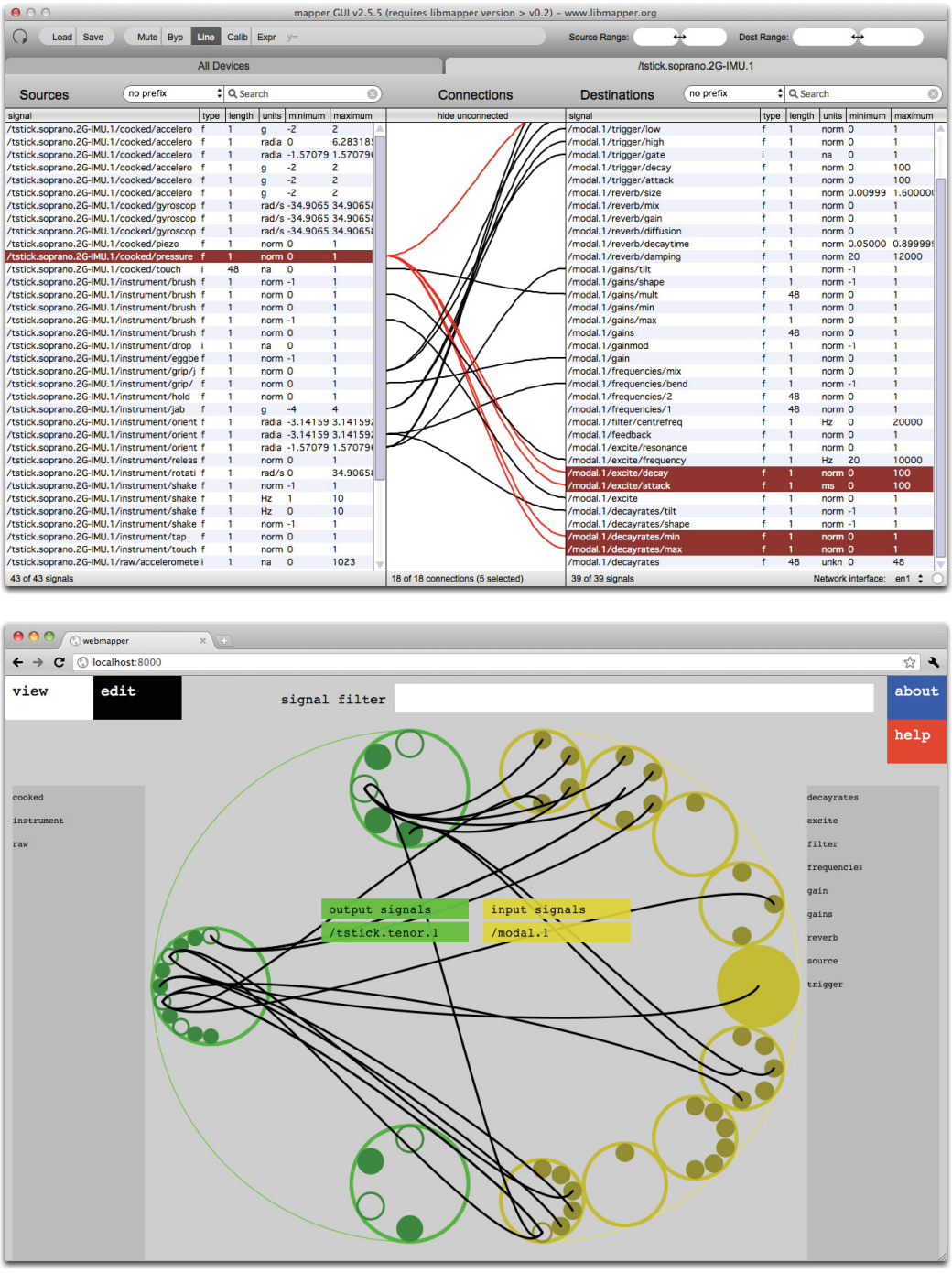

Fig. 2.7 Two different graphical user interfaces for managing the mapping network. Top: *mapperGUI* bu the first author, bottom: *vizmapper* by Vijay Rudraraju [4]

that there is no need for a single "correct" user interface; rather, different network representations and interaction approaches may be useful to different users, for different mapping tasks, or at different times.

All libmapper GUIs function as "dumb terminals" – no handling of mapping connection commands takes place in the GUI, but rather they are only responsible for representing the current state of the network links and connections, and issuing commands on behalf of the user. This means that an arbitrary number of GUIs can be open simultaneously supporting both remote network management and collaborative creation and editing during the mapping task. This approach has brought our attention to interesting research into other collaboratively-edited systems (e.g. [87]); our approach to undo/redo functionality is based on this work.

2.7.2 Automatic Session Recovery

Apart from explicit saving and loading, for specific projects we commonly implement small session-recovery programs to save time during working sessions. Sometimes while programming a particular device it is necessary to frequently recompile and relaunch the device, and recreating the mapping links and connections would be tedious. Using the *monitor* API, it only takes a few minutes to write a small program that watches for specific relaunched devices and recreates the desired mapping.

Eventually, we plan to create a general-purpose session logging and recovery system based on libmapper, backed by a version control database allowing the recovery of any previous configuration of the mapping network.

2.8 Mapping scenarios – libmapper use cases

In this section we briefly describe some examples of use cases for which libmapper has been designed.

2.8.1 Explicit Mapping Scenario

Imagine that Bob has an interesting synthesizer, and that Sally has been working on a novel physical input device. Since they both used libmapper bindings, collaboration is simple – they use a GUI to view the mapping network and create a link between the two

devices. Over the course of their session, they experiment with different mappings by creating and editing connections between the outputs of Sally's controller and the inputs of Bob's synthesizer.

2.8.2 Implicit Mapping Scenario

Sally and Bob decide to try a different approach: they are happy with some of the results from the explicit mapping approach but they want to jump quickly to controlling more parameters. They remove the direct link between their devices in the mapper network and instead link them through an intermediate machine-learning module/device and connect the signals they want to include in the mapping. Sally clicks on the button labeled "snapshot" on the intermediate device each time she wants to indicate that the current combination of gestural data and synthesizer configuration should be associated. Lastly, she clicks on a button labeled "process", and the device begins performing N-to-M-dimensional mapping from the controller signals to the synth signals.

Finally, Sally and Bob might decide to use a combination of the approaches outlined above, in which many "integral" parameters of the synth affecting the timbre of the resulting sound are mapped implicitly using machine learning, but the articulation of sounds (attacks, releases) and their overall volume are controlled using explicitly chosen mapping connections. They decide that for this particular project this approach provides the best balance between complex control and deterministic playability.

2.9 Support for Programming Languages and Environments

libmapper is written in the programming language C, making it usable as-is in C-like languages such as C++ and Objective-C, and by users of media programming platforms such as openFrameworks[4] and cinder[5]. Bindings for the Python and Java programming languages are also provided, the latter intended principally for compatibility with the Processing programming language which is popular in the visual digital arts community.

We have made every effort to ensure that the libmapper API is simple and easy to integrate into existing software. In most programs, the user-code must make only four calls to the libmapper API:

[4] http://www.openframeworks.cc/
[5] http://libcinder.org/

1. Declare a device (optionally with some metadata)
2. Add one or more signals (inputs and/or outputs, again with some optional metadata)
3. Update the outputs with new values
4. Call a polling function, which processes inputs and calls handlers when updates are received.

Figure 2.9 shows a simple (but complete) program using the libmapper C API. For simplicity here we will not show the monitor functionality, since for the most part only GUIs and other session managers need to instantiate monitors. Full API documentation is available online.

```
#include "mapper/mapper.h"

void handler (mapper_signal msig, mapper_db_signal props,
              int instance_id, void *value,
              int count, mapper_timetag_t *tt ) {
  // do something
}

void main() {
  mapper_device dev = mdev_new("my_device", 0, 0);
  mapper_signal in = mdev_add_input(dev, "/out", 1, 'i',
                                    0, 0, 0, handler);
  mapper_signal out = mdev_add_output(dev, "/out", 1, 'i', 0, 0, 0);

  int my_value = 0;
  while(my_value < 1000) {
    msig_update_int(out, my_value++);
    mdev_poll(dev, 100);
  }

  mdev_free(dev);
}
```

Fig. 2.8 Simple program using the libmapper C API to declare one input and one output signal.

2.9.1 Max/MSP and Pure Data

Max/MSP and PureData – two graphical patching environments for music programming – are supported via a `mapper` external object. This object instantiates a libmapper device with the name of the object's argument, and allows input and output signals to be added or removed using messages sent to the object. Output signals are updated by simply routing the new value into the object's inlet, and received inputs emerge from the object's left outlet. The object's metadata (IP address and port, unique name, etc.) are reported from the object's right outlet (Figure 2.9).

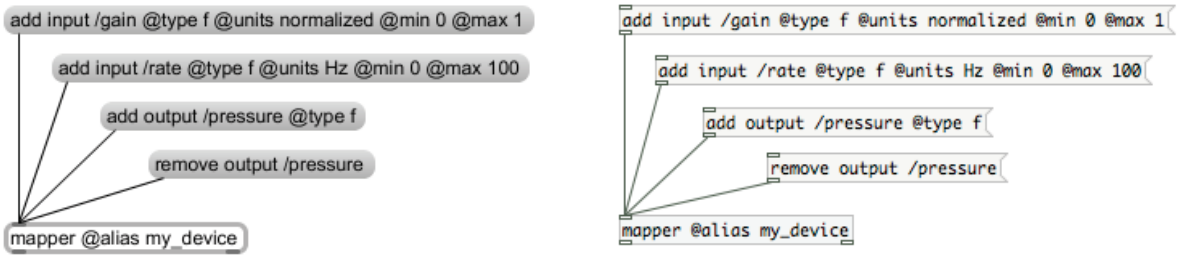

Fig. 2.9 Screenshots of the `mapper` object for Max/MSP (left) and PureData (right).

The `mapper` external object also provides an opportunity to provide parameter bindings for the popular music sequencing and production software Ableton Live[6], since it is possible to load Max/MSP patches as plugins in Ableton. Using the provided support for parameter discovery, one can quickly scan the open project and declare all the Ableton parameters as mappable signals on the network.

We also provide the `implicitmap` external object as a reference implementation for support of implicit mapping techniques with libmapper, and was built specifically for bridging libmapper and the MnM mapping tools from IRCAM [88]. This object instantiates both a libmapper *device* and a *monitor*, which it uses to observe its own connections and dynamically adjust its number of inputs and outputs as necessary.

[6] https://www.ableton.com/

2.10 Device and Software Support for libmapper

The most complex devices we use for mapping are invariably the ones developed in our lab – complex in terms of numbers of signals available for mapping, and also in terms of their departure from standard input devices such as piano keyboards. There are relatively few commercial offerings in the "alternate music controller" space. In addition to the lab-developed prototypes and research instruments, we also use a number of commercial input devices such as joysticks, depth-sensing camera systems, and various optical, magnetic, and inertial motion capture systems. We maintain a public collection of device drivers and utilities for working with libmapper that we hope will grow alongside the community making use of the library.

2.10.1 Protocol Bridges

In the interests of compatibility with other communication standards and with legacy hardware, we are developing a series of software daemons that function as protocol bridges to the libmapper mapping network. The MIDI protocol in particular is of interest to us, since almost all music hardware built since the 1980s is compatible with MIDI, and it remains the standard for most commercial music software today. Our MIDI bridge makes use of libmapper and the open-source cross-platform MIDI library RtMidi[7] to expose MIDI signals to the mapping network. If the software is running as a daemon, any MIDI devices recognized by the operating system will be dynamically added to the available pool of mappable devices without any user intervention.

Popular computer peripheral input devices such as gaming joysticks and graphics tablets are also commonly used for multidimensional control of music software. For this reason, we are also developing a software daemon for automatically exposing the parameters of peripherals using the Human Interface Device (HID) standard for mapping.

Finally, the Arduino microcontroller platform[8] is immensely popular for creating DIY electronics in general, and new control interfaces for music in particular. Students in our lab frequently use Arduino circuit boards for sampling sensors and communicating with a computer running audio synthesis in PureData or Max/MSP; usually custom firmware is written to enable the microcontroller to efficiently perform specific tasks, however a generic

[7] http://www.music.mcgill.ca/~gary/rtmidi/
[8] http://www.arduino.cc/

firmware – "Firmata" – is sometimes used instead to allow dynamic reconfiguration of the Arduino at run-time [89]. For users of Firmata, we make available an adaptation of the application "firmata_test"[9] that exposes the configured pins for mapping[10].

Sending and Receiving Open Sound Control from libmapper

Although we argue that calling libmapper from your application is the best route towards intercommunication, libmapper-enabled devices/applications can be used to send and receive Open Sound Control messages for compatibility with tools lacking libmapper bindings but including OSC support. Sending OSC messages from a libmapper-enabled application to an OSC-only application simply involves manually providing enough routing information (receiving IP and port and the expected OSC message path) to create a dummy libmapper connection. Sending messages from an OSC-only application to libmapper is also simple; since libmapper uses OSC internally for message passing one can simply send properly-formatted OSC messages to the destination device. The routing information, OSC paths, data types and vector lengths for the destination signals can be easily inspected using any mapper GUI on the network.

Naturally, these tricks only work for fairly simple scenarios, and will not support device discovery, value queries, or much of the session management features of libmapper. However, in the libmapper-to-OSC scenario the spoofed connection will support signal processing and session management.

2.11 Conclusions and Future Work

In conclusion, we believe that designers of interactive systems should use the most suitable, interesting representations of their systems, and that translation should be used for providing compatibility rather than standardization and normalization of signals. We offer libmapper to the community as tools for accomplishing this goal: a cross-platform, open-source software library with bindings for an increasing number of popular programming languages, platforms, and environments. The future of libmapper includes lots of exciting work for anybody interested in aiding or guiding development – a few of the items on our roadmap are described in the next section.

[9] http://firmata.org/wiki/Main_Page
[10] https://github.com/IDMIL/firmata-mapper

Our own experience using libmapper for system interconnections on a variety of large and small projects has been largely positive. While libmapper does not miraculously make mapping easy (of course!), it greatly streamlines our work. Systems are now much less tedious to set up and interconnect, even when working with student projects that were not conceived to be compatible in any way. The use of libmapper and surrounding tools also acts to democratize collaborative mapping sessions, since participants can experiment with mappings between systems written in unfamiliar programming languages, and often even while the devices themselves are being modified. Most importantly, we find that the time we save means that we can try more ideas in the available workshop time, easily compare and contrast them, iterate and refine them. We use libmapper in workshops, demos, concert performances, and daily in our lab.

2.11.1 Future Work

Current work on libmapper focuses on finalizing the function and programming interfaces for dealing with *instances* of signals, for the handling of polyphony in synthesizers as well as "blob tracking" and other methods that involve virtual entities that pop in and out of existence. This work will be treated in a future publication (cf. Chapter 3 of this thesis).

In tandem with the *instances* functionality, we are progressing on the full integration of absolute time-tagging of mapped data. Once the groundwork is finished, this functionality will enable automatic jitter-mitigation in data streams since libmapper will be able to dynamically determine the amount of latency in a given link or connection. Perhaps more interestingly, connection processing will be extended to allow arbitrary processing of the message time-tags, enabling flexible delays, debouncing, resampling, or ramping to be part of the designed mapping connections.

Finally, we are continuously working on adding support for new programming languages and input devices, including progress on interfacing libmapper with embedded systems.

2.12 More Information

More information on libmapper and related projects can be found on the project website `libmapper.org` or by subscribing to one of the mailing lists (developer or user). The website includes online documentation of the libmapper application programming interface, pre-built binary versions of the library for various platforms, and links to projects and

utilities using libmapper. All libmapper development is performed in open consultation with the community mailing list, and anyone interested can join to participate in defining and implementing the future of the library and its surrounding ecosystem.

2.13 Acknowledgements

Development of *libmapper* has been supported by funds from the Fonds de recherche sur la société et la culture (FQRSC) of the Quebec government, the Natural Sciences and Engineering Research Council of Canada (NSERC), and the NSERC/Canada Council for the Arts New Media Initiative.

Chapter 3

Generalized Multi-Instance Control Mapping for Interactive Media Systems

The following chapter continues where Chapter 2 finished, and addresses the design of extensions to our mapping tools for mapping of signals with multiple instances. This chapter was prepared as the manuscript:

J. Malloch, S. Sinclair, and M. M. Wanderley, "Generalized multi-instance control mapping for interactive media systems, " Manuscript prepared for submission.

3.1 Abstract

We articulate a need for supporting the representation of entities with multiple instances in tools for designing and using interactive media systems. A list of system requirements is compiled from examination of existing tools, practical use-cases, and abstract considerations of node connectivity and information propagation within a graph of connected devices. The addition of support for mapping multi-instance signals to the open-source software *libmapper* is described, along with practical examples of the new features in use.

3.2 Introduction

3.2.1 Digital Musical Instruments

For the purposes of our research and this article, a "digital musical instrument" (DMI) is a system which affords real-time control over digital audio synthesis algorithms using some sort of user interface [7]. While approaches to user-interface and media synthesis may vary considerably, a unifying feature of DMIs is that – unlike acoustic systems – there exists no *physical* relationship between user input and its effect on the system state. Instead, associations between gesture and sound (or other media) must be *designed* or otherwise generated before the system becomes functional; this *mapping* from sensed gestures, postures or other phenomena to media control dramatically impacts the response of the system and thus the experience of the performer and audience [90].

There are several typologies for considering mapping approaches. Mapping topologies may be *one-to-one*, *convergent* (in which multiple source parameters affect a single destination), *divergent* (in which one source parameter affects multiple destinations), or a combination [30]. Hunt and Kirk described mappings as *simple* or *complex* [91] depending on the inter-relations between dimensions of input and output; a similar distinction is made between *integral* and *separable* approaches to control [92]. A mapping might remain *static* over a period of time (e.g. the duration of a performance), or change *dynamically*, and it could be defined through *explicit* configuration of connections or be generated *implicitly* through the use of machine learning [93]. Several different layers of mapping might be conceived between two entities, either for the purposes of system abstraction/organization [33] or to combine several mapping approaches.

3.2.2 libmapper

libmapper (cf. Chapter 2) is an open-source, cross-platform software library for providing discoverability and compatibility between interactive real-time systems for the purposes of designing mapping layers between them [41]. It is written in C, and bindings are provided for Python, Java, Processing, SuperCollider, Max/MSP, and Pure Data. Each application that uses *libmapper* becomes a node in a distributed network, and real-time data can be streamed between nodes in a peer-to-peer manner. Additionally, metadata describing the nodes and their capabilities can be discovered by peer nodes or by *monitors*, which are

usually embedded in some sort of user-interface for viewing and managing the mapping network state. For simple connectivity, *libmapper* ensures compatibility between data-stream endpoints by coercing, rescaling, and translating data representations as necessary. The automatic transformations can be over-ridden by the mapping designer, assigning custom signal-processing expressions that are applied to the data streams. The collection of *libmapper* connections and their configurations forms a *mapping* between associated sets of nodes, and can be saved and recreated later for further modification or for live performance or other types of interaction (e.g. interactive media installations).

A Note on Terminology

For the *libmapper* project we have adopted certain terminology which we will also use throughout the rest of this paper. For clarity we will define a few terms here:

signal a named data-stream with associated properties. A signal may be either an *input* or an *output*, however when discussing mapping we usually find it less confusing to consider data flow from the perspective of the mapping layer, for which signals are either *sources* of information, or *destinations* for information.

device a container for signals often corresponding to a physical object or a software program; conceptually a logical collection of signals, though a device may also have other associated properties. Signals could be considered as dynamic properties of a device.

connection a mapping association between one *source* signal and one *destination* signal. For *libmapper*, connections represent both data transport and representation transformation including arbitrary processing.

link a conceptual representation of "router" software objects for forwarding and accepting data flow between devices; a network-layer association between two devices.

3.2.3 The Case for Multi-Instance Mapping

Since *libmapper* provides run-time configurability to inter-device mapping, connections can be created and destroyed on-the-fly by users or by automated processes by sending session-management messages to the relevant devices. In a typical scenario, the devices and signals available for mapping remain constant for the duration of a concert or mapping session;

even in the case of pieces or installations with "dynamic mapping" the changes to mapping configurations usually take place at intervals greater than a second.

The are also scenarios, however, in which it is necessary to support multiple distinct copies of signals – or groups of signals – that dynamically appear and disappear. The most obvious musical example is the support of polyphonic synthesizers, capable of producing more than one sound simultaneously. There are however many other examples: multitouch interfaces, computer vision systems that extract objects of interest (faces, people, "blobs", etc), and tangible user interfaces involving multiple identical objects. For example, we cannot known ahead of time how many touches there will be on a multitouch interface at a given time, only a possible maximum permitted by the sensing hardware or software drivers. The requirements for mapping with these types of signals are discussed below.

In the following sections we will outline our approach to this problem through the examination of several hypothetical use-cases, and attempt a formalization of our requirements for supporting multi-instance mapping in *libmapper*. In sections 3.6 and 3.7, we describe the integration of multi-instance support into *libmapper* and demonstrate its use for several associated software tools.

3.3 Our Concept of Signal Instances

Our aim is to flexibly support mapping scenarios which deal with instances of signals or objects, and we considered many use-cases to help form a general concept of instances (figure 3.1). Since it is impossible to anticipate all of the possible specific systems and use-cases, we will attempt to first outline the general scope of the concept and identify systems in which it should not be applied. In section 3.5 we will use these general rules to formalize the requirements of the system.

3.3.1 Defining New Objects by Mapping

We have decided not to support the creation or definition of new higher-level objects from aggregates of signal instances (e.g., "barycentre of three touches on a touchscreen"). We intend libmapper to be a lightweight library providing interconnections between existing systems rather than a programming language/environment in itself. With this in mind, some complex mapping scenarios may be outside the scope of the library, however we fully endorse the idea of using interesting, complex *intermediate* mapping devices/layers

which might provide complex processing and expose new or aggregate objects for mapping downstream.

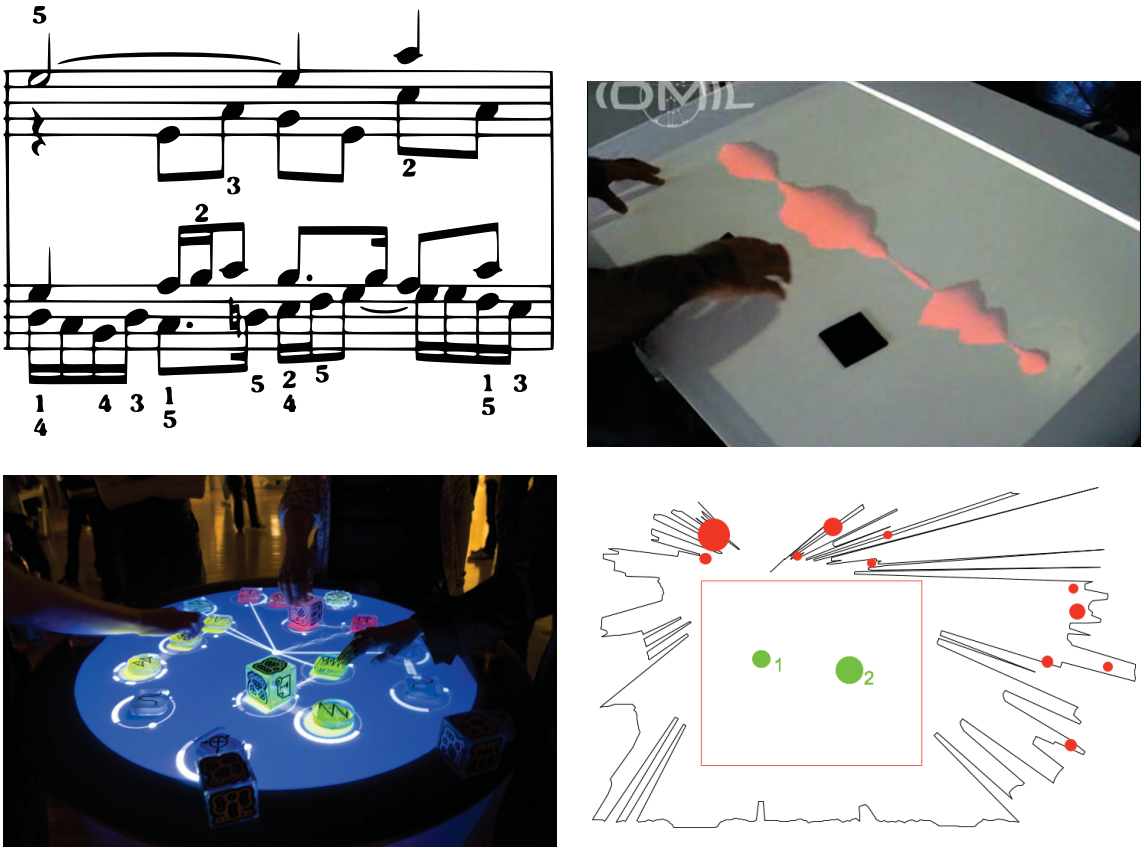

Fig. 3.1 Use-cases for multi-instance mapping. Clockwise from top-left: polyphonic audio synthesis; multitouch interfaces (tabletop multi-user interface from the IDMIL); computer vision (blob tracking using data from a scanning laser rangefinder); tangible user-interfaces such as the Reactable [5] (image: Daniel Williams/Wikimedia Commons).

3.3.2 Instances are Interchangeable

The first important feature of our definition of instances is that they are completely *interchangeable*. It is central to the system-design philosophy espoused/encouraged by libmapper that strong semantics are attached to all entities whenever possible (whenever such information exists). For example, we believe that using two instances of some signal la-

beled `/arm/position` to represent left and right arms is not a good approach, since from the interactor's point of view (and likely that of the audience also) there is a strong, persistent difference between their two arms and they are unlikely to confuse the two. In this case separate signals – e.g. `/arm/right/position` and `/arm/left/position` – is likely a better approach. The instance concept should only be applied when there are no differentiating characteristics between the entities involved from the perspective of the system designer, and hopefully also from that of the potential interactors.

3.3.3 Instances can be Dynamically Created and Destroyed

Instances of a phenomena may be ephemeral and their creation and destruction may be driven by unpredictable control signals. "Destruction" in the case of virtual entities (e.g. touches) means actual destruction, for persistent entities (e.g. physical objects) it implies removal from the scope of the system (e.g. removed from an interactive tabletop).

3.3.4 Instances can be Serial and/or Parallel

We also consider "gestures" – or other segmentations of continuous signals – to be instances of temporal objects (figure 3.2). In music this also applies to the abstract but useful concept of "notes" – from a signal-processing point of view we can see that we are still dealing with continuous time and not discrete events. Perceptual segmentation into notes occurs at two different levels: the performer's intention to play a note, and the auditory system of a listener identifying some sort of sound "object".

3.3.5 Mapping Once

For many cases, the very *existence* of signal instances is an important part of their state, and needs to be mappable to destination devices. However, if we dynamically create and destroy mapping connections as instances pop in and out of existence we are essentially abusing the administrative data bus for mapping this signal state rather than administrating connections. The information required to map *all* instances is already contained in the message to connect *one* instance, so it becomes obvious that dynamic connections are the wrong approach. Instead, we would like to publish a signal once for an entire class, no matter how many instances it might have. In order to simplify configuration, we would also like to be able to map all instances of a signal with a single connection.

Fig. 3.2 A comparison of parallel instance lifetimes (top) and serial instance lifetimes (bottom).

3.3.6 Dynamic Re-assignment of Resources

Designers of polyphonic synthesizers have long had to deal with situations in which the synth does not possess adequate resources to simultaneously produce all of the sounds it is commanded to play. Supporting many synth voices is less problematic now that most synthesis is handled by software, but in earlier days a synthesizer would possess a limited number of oscillators or voice circuits, even after the adoption of the MIDI protocol for controlling the synths. The MIDI protocol uses "note number" to index the instances of the represented object "note," and is capable of representing 128 *simultaneously sounding* notes per channel. Since the protocol doesn't require the control hardware to track which notes are sounding, it is trivial to build a keyboard that can send more notes than the destination synth would be able to handle.

The typical approach to dealing with this situation is to employ *voice stealing*, so that a newly triggered note would "steal" the resources of an already-sounding note. Often the oldest note would be stolen: many sounds decay as they progress and the oldest sound is likely to be the quietest and have the least impact if released; more sophisticated synthesizers might identify and release the quietest note regardless of its age. The synth designer

must also decide whether to play the end sample of the note to be released (in the case of wavetable synthesizers) or activate the "release" portion of an amplitude envelope (in the case of typical additive, subtractive, or FM synthesizers) before starting the new note.

For more general mapping of interactive systems it is still necessary to consider that some devices may have constrained resources similar to fixed polyphony in sound synthesizers. Even for systems that dynamically use an arbitrary number of instances it may be more efficient to preallocate some of the necessary resources.

3.4 Support for instances in existing tools

3.4.1 Instances in MIDI

The Musical Instrument Digital Interface (MIDI) protocol and connection standard has been the de facto standard for connecting commercial electronic musical equipment since the early 1980's [66]. Designed initially for connecting piano keyboard-style controllers to sound synthesis gear with severely-constrained communications bandwidth (by modern standards), the protocol semantics and capabilities are heavily biased towards keyboards. The protocol includes 16 different *channels*, and supports polyphony (multiple instances of "note") by labelling most messages with a 7-bit *note number* – up to 128 different notes can be active on a channel at the same time – most remaining messages are intended to address all active notes on a channel. This scheme hard-codes the semantics of the controller, conflating key number with a musical pitch rather than considering the separability of input device and synthesizer, and forces musical control to fit into the "note" concept: atomic temporal objects with explicitly defined pitch, beginning and end. For general mapping purposes these constraints are unacceptable, since "hacking" the protocol to represent more flexible or more complex systems would abandon its main advantage – compatibility with existing hardware and software.

For our general solution, we prefer not to think of note-on/note-off events, but rather a *key-activation* as being an instance of a gesture that ends when the key is lifted. In practice we might also desire to consider the trajectory of the hand/fingers when not in contact with a keyboard.

Handling Dropped Messages

Since MIDI was designed for transport over dedicated cables, issues with loss of instance synchronization were not directly addressed. The MIDI specification does include a message for *all notes off* which will turn off active notes without sending specific NOTEOFF messages for each one. Adaptation of the MIDI protocol for packet-switching networks resulted in RTP-MIDI [94], which maintains the message types and semantics of standard MIDI but defines a container packet format. More importantly, RTP-MIDI adds a recovery journal system to allow dynamic recovery from lost packets, which are much more likely to occur than over a dedicated wire.

3.4.2 Instances in TUIO

TUIO is a protocol designed for communicating the state of table-top "tangible" user interfaces for real-time control of media synthesis [3]. It is a higher-level protocol built on top of Open Sound Control [95] – as is *libmapper* – and naturally supports working with multiple objects on the table. Each object is assigned a unique id, and there are set message types for different kinds of objects. The protocol is hard-coded to use certain representations, coded as heterogeneous data vectors. While it works very well for its designed use-case and enjoys fairly widespread use, TUIO is ill-suited for representing arbitrary control systems.

To mitigate the effects of dropped UDP packets, the presence of objects on the table is not communicated using explicit messages, but is deduced by the receiver by tracking the stream of object updates and a stream of `alive` messages reporting the ids of all objects on the table. Additionally, redundant system state information (an additional `alive` message) is included in each packet.

3.5 Formal Requirements

In order to formally consider the effects of various connection topologies on multi-instance mapping, we will make use of some concepts from graph theory. Briefly, in graph theory, a graph consists of *nodes* which may be connected by *edges*; a subset of graphs possess edges that are *directed*, making possible loops that feed back on themselves, termed *cycles* [96]. Links and connections formed by libmapper can be considered to be directed edges,

and cycles are possible if defined by the user; nothing more specific can be said without considering some examples. Starting with simple graphs, we will try to build a specification for supporting instances in libmapper. Here, we will consider graph nodes to be libmapper devices; since multiple connections can be made between two devices the general form will be a *directed multigraph* when considering connections, and simply a *directed graph* when considering links.

In the diagrams to follow, devices/nodes will be drawn with a fill if they can initiate the creation of new instances. The creation of a new instance by node A can (but does not necessarily) cause an associated instance to be created by any other node connected directly downstream. We can consider this as information propagation within the graph originating at node A; in the following exploration we will refer to the node which initiated this chain of events as the *instance origin*.

3.5.1 Acyclic graphs

The simplest graph we are concerned with includes two nodes connected with no cycles – data flows in only one direction. The creation of new instances by node A may cause the activation of corresponding instances by node B. Our only system requirement at this stage is that the instances of different signals need to be synchronized across the link between nodes, so that downstream nodes can update themselves appropriately.

There are four possible acyclic graphs that can be formed with three nodes, shown in figure 3.3

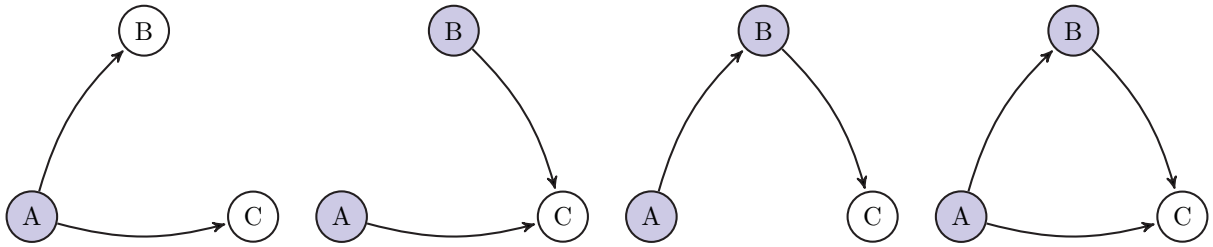

Fig. 3.3 Three nodes with no cycles. From left:divergent, convergent, chain, chain + convergent.

Divergent – This scenario is essentially the same as the simple two-node graph, collisions between instance ids generated by different devices are not possible. Example: one input device controls two synthesizers.

Convergent – Two nodes are both routing instances to the same destination – we will need to avoid collisions between instance identifiers generated by each device. Example: two input devices connected to the same synthesizer.

Chain – If the intermediate node can also activate new instances, we need to ensure that the final destination can distinguish between instances generated by each node. This is the same as the convergent graph, except the instances from two nodes are carried on the same link. Example: a virtual physics engine is used between input device and synthesizer to add dynamic behaviour to the controlled synth parameters.

Chain+convergent – This graph adds another requirement, since the final node must be able to associate instances from the first node that have arrived directly *and* through an intermediate node. At the output of an intermediate node, instances will need to be tagged or labelled with information identifying the instance origin. Example: one input device is mapped to one synthesizer, but with a subset of its connections connected through the virtual physics engine mentioned in the last example.

Acyclic graphs with more than three devices can be seen as combinations of the above possibilities, and do not create any new requirements. If instance identification is already required to persist through one intermediate node, it doesn't matter if the chain of nodes is made arbitrarily longer. Our only modification to the requirements for convergent topologies is that we must handle N incoming edges rather than only two.

From this investigation so far, we can conclude that we require globally-unique instance identifiers in order to allow re-association of instance causal chains that re-converge while avoiding collisions between instance identifiers.

3.5.2 Cycle Graphs

There are many more possible variations of the above graphs if cycles are permitted. For the most part, the introduction does not affect our requirements, since unique instance identification is already required for acyclic scenarios. There are several special cases which may help inform our requirements, depicted in figure 3.4.

loops on non-origin nodes – In this scenario we need to support instance updates generated by nodes other than the instance origin, adding a simple requirement that non-origin nodes have access to the necessary information for properly labelling outgoing

updates, whether or not the update is triggered by an upstream signal. Technically, we would have to handle this situation anyway, since the *rate* of updates on a given edge is not specified in our graphs and is not constrained to match that of any other edge. Only the propagation of instance existence/lifetime is considered here, and not updates to its value. Example: the virtual physics engine from previous examples updates locally and outputs messages at its own rate regardless of the input message rate.

subgraphs with a shared node – This graph combines the convergent and divergent examples given for acyclic graphs. A real-world example of this scenario could involve two input/output interfaces (such as haptic controllers) mapped to a shared virtual environment. In the interests of minimizing wasted bandwidth, we will need to ensure that instance updates from the shared node are sent only to the correct instance origin.

subgraphs with 2 or more shared nodes – Similar to the last example, but highlights the requirement that the instance information transported by edges must be configurable. The symmetrical graph shown third in figure 3.4 could be implemented by declaring subgraphs ADB and CDB and enforcing the following rule: an edge can only transport an instance if the instance origin belongs to the subgraph. Example: a similar scenario to the last one but with two virtual environments connected in a chain (nodes D and B).

arbitrary edge scopes – These examples have demonstrated the need for each edge to transport or block instance information in a configurable manner. Since we cannot know how users will interconnect nodes in practice, we must allow this configuration to be defined arbitrarily. The fourth graph in figure 3.4 cannot be implemented using the subgraph rule identified above, since the mapping designer has arbitrarily chosen that the edge CA should only transport instances with B as their origin. An alternate rule will suffice however: we define *directed hyperedges* as the group identity for deciding whether an instance update should be transported over a given edge.

3.5.3 Node Autonomy

In a libmapper network, the origin node is not necessarily in a master/slave relationship with downstream nodes, which may run simulations of dynamic systems, implement media

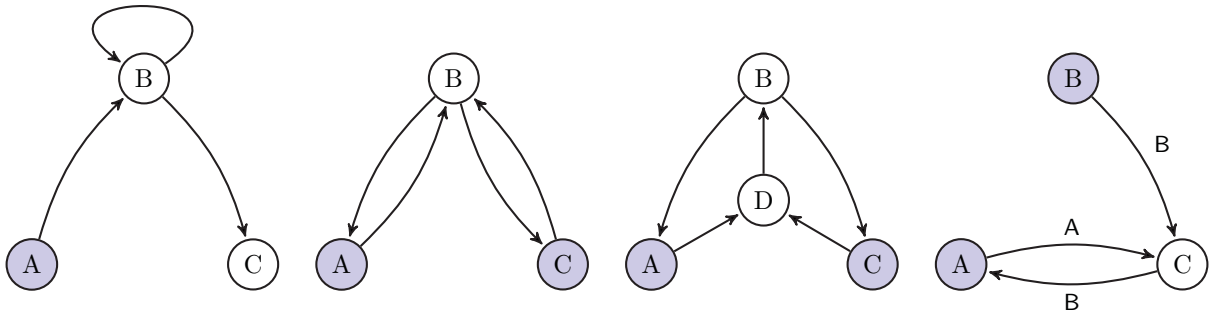

Fig. 3.4 Several examples of cycle graphs. From left: a 3-node chain with a cycle on the intermediate node; two 2-cycle cycle graphs sharing node B; two 3-cycle subgraphs sharing nodes B and D; an "arbitrary" graph with edges labelled with instance scope.

synthesis models with their own timing requirements, or otherwise have their own internal logic regarding instance updates and lifetimes. For example, an incoming instance may cause the creation of an object in a simulation that will persist after the corresponding instance at the origin has been released/destroyed. Perhaps the mapping designer wishes to map touches on a multitouch screen to the creation of objects downstream, but wants the lifetime of the objects to be administered by the local device. We need to add some requirements to allow for autonomous decisions by arbitrary nodes:

- Any node can release its local instances at any time, and must propagate the instance release downstream even if it is not the instance origin node. It may also be interesting to inform any upstream nodes of the release – in this case information must flow in the opposite direction of our defined directed edges.

- Any node can ignore instance releases from upstream. After this point it may be useful to know that the instance release must be generated locally.

- We also need to consider that nodes do not necessarily have the same internal resources, for example an origin node might allocate 10 instances while a node downstream only has the ability to activate 5 instances. The behaviour of the downstream node when the 6th instance is received is important – for some scenarios it should probably ignore the update, but for others it should release an active instance to free resources for the new instance. This "stealing logic" will need to be configurable.

3.5.4 Summary of formal requirements

Based on the the scenarios explored above, we can compile a short list of requirements for building a multi-instance capable mapping system.

1. Globally-unique instance ids – unique across graph and across time.
2. Edges belong to groups (hyperedges) – we require configurable propagation of a given instance across an edge based on inclusion of the edge in a defined hyperedge. Edges can belong to multiple hyperedges.
3. Autonomous nodes – we will allow nodes autonomy in activating and releasing instances from upstream. Upstream devices will be notified of early instance releases.
4. Stealing – configurable "stealing logic" for coordination of nodes with mismatched resources.

3.6 Adding Multi-Instance Support to libmapper

Support for multi-instance signals is included in libmapper versions 0.3 and greater. In this section we will briefly outline our specific implementation decisions, with a focus on the user-facing application programming interface (API). As in the previous section, we will use the term *instance origin* to refer to a node (now device) that starts the propagation of a given instance. It is worth noting that in a directed graph, the *origin* is the only node for which all other nodes touched by the hyperedge are necessarily downstream; messages sent by other nodes will not necessarily reach the entire group/subgraph.

3.6.1 Instance Identification

Globally-unique two-part instance identifiers are created by the origin device when an instance is first activated (explained in section 3.6.4), generated from a CRC32 hash of the device name and an incremented counter stored in the device data structure. Device names are already guaranteed to be unique within the libmapper network: if multiple copies of the device class are running, a unique ordinal is appended to the given device name. The allocation algorithm responsible for ordinal negotiation was modified to ensure that collisions of the name+ordinal hash cannot occur.

The second part of the id ensures that instances generated serially by a device can be differentiated in the network (until the counter overflows), even when the API is called with the same id by user code. This two-part instance id is used by all devices and persists for the lifetime of the instance in the network, which may be significantly longer than in the origin device, since downstream nodes decide for themselves when to release their instances.

Arbitrary instance ids can be used by the user code; each device maintains a map of local instance ids to global ids and ensures that outgoing instance updates are tagged with the correct information. This allows instances of different signals to be coordinated. These maps are reference-counted and are freed when no longer needed.

3.6.2 Link Scopes

Devices have no knowledge of the configured network structure beyond their own links and connections. Membership of a link in a given hyperedge is defined as the link property `scope`, which is configurable using the standard libmapper protocols for setting object properties. When a given signal instance is updated, the router object checks to see if the origin hash in the instance's id_map matches the scope of each outgoing link before sending over that link. One of the libmapper session-management interfaces has been modified to provide configuration of link scopes by providing a drop-down list of available instance origins.

3.6.3 Reserving Instances

The standard signal creation functions `mdev_add_input()` and `mdev_add_output()` now automatically allocate one instance of the signal with the local id 0; if the pre-existing libmapper signal update function `msig_update()` is called, it will access this default instance. Any time after creating a new libmapper signal, additional instances can be allocated. We recommend doing this on application startup, but the number of instances can be adjusted at any time. Specific instances ids (for coordinating instances across signals) and user context pointers can be specified per instance if needed.

3.6.4 Updating Instances

An instance lifetime begins when the signal instance update function is called for a specific id for the first time. There are no explicit messages for activating a new instance. Internally,

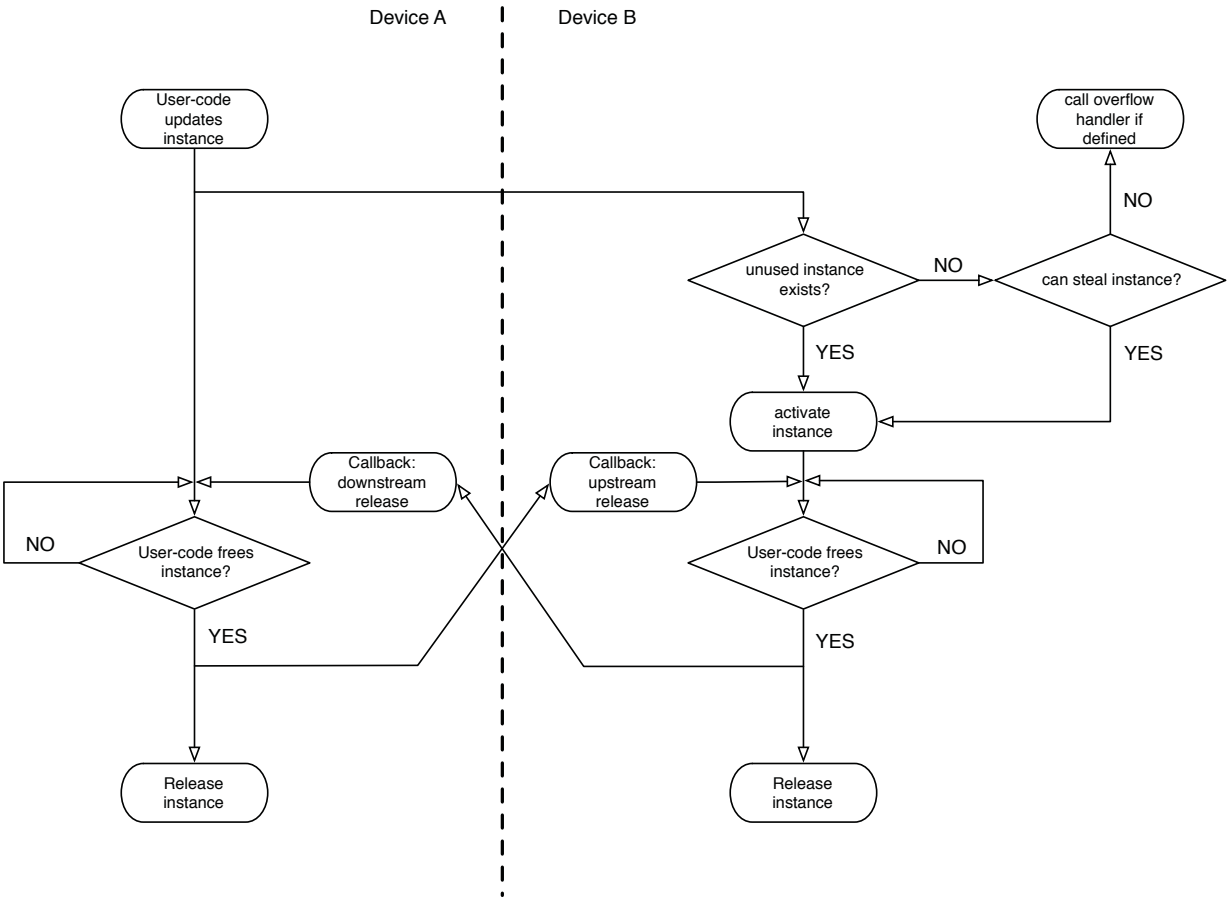

Fig. 3.5 Flowchart showing instance lifetime relationship between two connected signals.

libmapper first checks if the instance is already active; if not it will try to activate a reserved instance or steal an active one (see section 3.6.7). If an id_map does not already exist for this local instance id, a new id_map will be generated with the local device as the origin, otherwise it will associate the stored map with the new instance. As mentioned above, the signal update message sent on the network includes the global/public part of this id_map.

3.6.5 Receiving Instance Updates

On the receiving side, a second message-hander function has been added internally for processing instance updates. This function follows a similar process to the update function: if a signal instance matching the public/global ids is not already active, a reserved instance will be activated if available. The device id_maps are consulted to see if another signal has already used the public ids; if so, libmapper will activate the instance matching the private/local part of the id_map, otherwise a new id_map will be stored associating the public ids with an unused private id available in the pool of reserved instances.

If an instance is successfully activated (or was already active) a user-defined handler function is called with the local instance id and the new value.

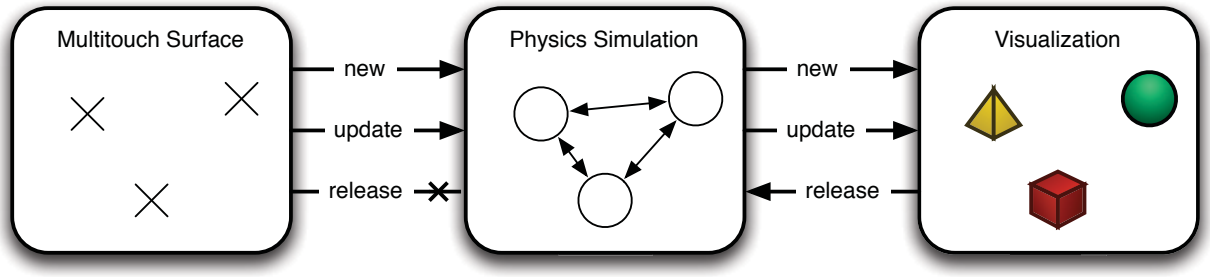

Fig. 3.6 Example scenario with downstream instance release. New touches and position updates to existing touches on the multitouch screen are mapped to a physics simulation, which creates virtual objects for each touch, however when the touches are lifted the physics simulation persists. The outputs of the simulation are mapped in turn to a visualization engine with ephemeral objects; when the visualization instances fade out, a release message is sent upstream and the physics model removes its corresponding local instance.

3.6.6 Releasing Instances

An active instance can be released simply by calling the function `msig_release_instance()` with the appropriate signals and local instance id as arguments. This causes libmapper to propagate the instance release to any downstream nodes and return local resources to the instance reserve pool. If the local device is not the instance origin, an instance release notification is also sent upstream – a possible scenario demonstrating this feature is outlined in figure 3.6.

In order to implement downstream release notifications, an internal "receiver" data structure was added to handle the receiving end of a link. This data structure mirrors the "router" structure on the sending end, and stores information necessary for sending messages upstream (host IP addresses, ports, and link scopes). Adding the receiver structure also allowed us to implement a new connection property that optionally reverses the flow of updates on the connection, so that the "sender" receives updates (if a signal handler has been defined) every time the signal is updated at the "receiver" – either by local API calls caused by user interaction (e.g. setting a parameter on a synthesizer) or by incoming remote updates on other connections. Such "reversed" updates can be used during training of supervised machine learning algorithms, when examples of both source and destination state are needed. The new feature complements the existing query/response system implemented in libmapper.

3.6.7 Instance Stealing

An *instance allocation mode* can be configured for each libmapper signal, specifying the desired behaviour when a reserved instance is not available for activation. The options `steal_oldest` and `steal_newest` cause the release handler to be called for the oldest or newest active instance, respectively. If a more sophisticated method is required for determining which instance to steal, or if more instances should be allocated, the user code can also provide an "instance management callback" function. In addition to *instance_overflow*, this function can be called when a new instance is activated, or when a downstream or upstream release message is received; the function can be configured to ignore any subset of the possible instance event types.

When an instance is "stolen" the device will mark the associated id_map as locally-released and decrement the local reference count, but will maintain a record of the id_map

until any upstream nodes have also released the instance. This ensures that incoming updates for an already-stolen instance do not trigger the stealing logic again.

3.6.8 Multi-signal "Objects"

Suppose we have a scenario in which a message arrives with a previously-unseen public identifier, and we have a local instance of the receiving signal available, but an instance of *another* local signal with the same id is already in use. Without being informed somehow by the user-code, libmapper has no way of determining whether the two signals are two properties of the same entity or "object" (in which case the instances should be synchronized) or whether they are unrelated (in which case it is safe to activate a new instance in our scenario). In the latter case, if we store a new id map associating the new public identifier with a previously mapped local id we have now complicated things for local signal updates, since we have to choose between two conflicting id maps when choosing a public identifier.

To avoid these situations, we need to indicate any sibling-relationships between signals. This can be specified implicitly by using different id ranges for different objects, e.g. signals representing properties of object A are assigned instances ids from 0–9 and signals representing object B get instances ids in the range 10-19. When designing devices with possibly-large, dynamic instance populations this is a bit awkward and hard to track, and some users may wish to use automatically-generated instance ids without worrying about collisions between objects. The API also allows explicit definition of *multisignal objects* – in the code below libmapper is informed that `sig1` and `sig2` are properties of the same object:

```
mapper_multisig mms = mdev_add_multisig(dev);
msig_set_group(sig1, mms);
msig_set_group(sig2, mms);
```

Internally, libmapper creates a separate id map for each `multisig`, meaning that id map collisions cannot take place between instances of different objects. A signal can be reassigned to use the default device-wide id map by passing `NULL` for the second argument to `msig_set_group()`

3.6.9 Handling Orphaned Instances

Since libmapper devices form a distributed network, it is inevitable that at some point instances will be accidentally "orphaned" from their upstream progenitors. For devices with passive instance lifetime logic – always keeping local instances synced to upstream – this will result in local instances that no longer receive updates but still consume memory resources. To handle such situations, local instance release callbacks are also triggered when the connection or link from the origin is removed, if the upstream device logs out, or if the link scope property is edited to effectively cut-off the instance from upstream. In the event of an entire upstream device disappearing from the network without properly logging out (due to an application crash or network interruption) we will also need to release orphaned instances – this scenario can be detected by tracking the timing pings sent on the admin bus by active devices.

3.7 Examples

In order to "exercise" the libmapper instances API, we looked for possible applications in our own work and that of our colleagues. For each test we required only that the source code was available, that is was written in a language compatible with libmapper (currently C/C++, Python, Java, SuperCollider, Max/MSP or Pure Data), and that the application included some entity for which multiple instances could exist. Several examples are outlined below.

3.7.1 Different Strokes

"Different Strokes" (DS) is a software application for performing music on a computer through drawing, usually with a digitizer tablet as the input device [97, 98]. When a freehand line is drawn into the interface, the application stores the position and velocity trajectories enabling the stroke to be re-played by the system. Several stroke colours can be selected from a palette, each of which is associated with a particular sound sample; the duration of the sample is normalized to the length of a drawn stroke, and the speed of the sample playback is mapped to the velocity trajectory. "Playback heads" (analogous to a multitrack tape recorder) are represented by *particles* which travel along the drawn strokes. New particles are triggered by either the creation of a new intersection, or intersection

with another particle; in this way, complex feedback structures and periodic loops can be constructed using very simple rules (figure 3.7).

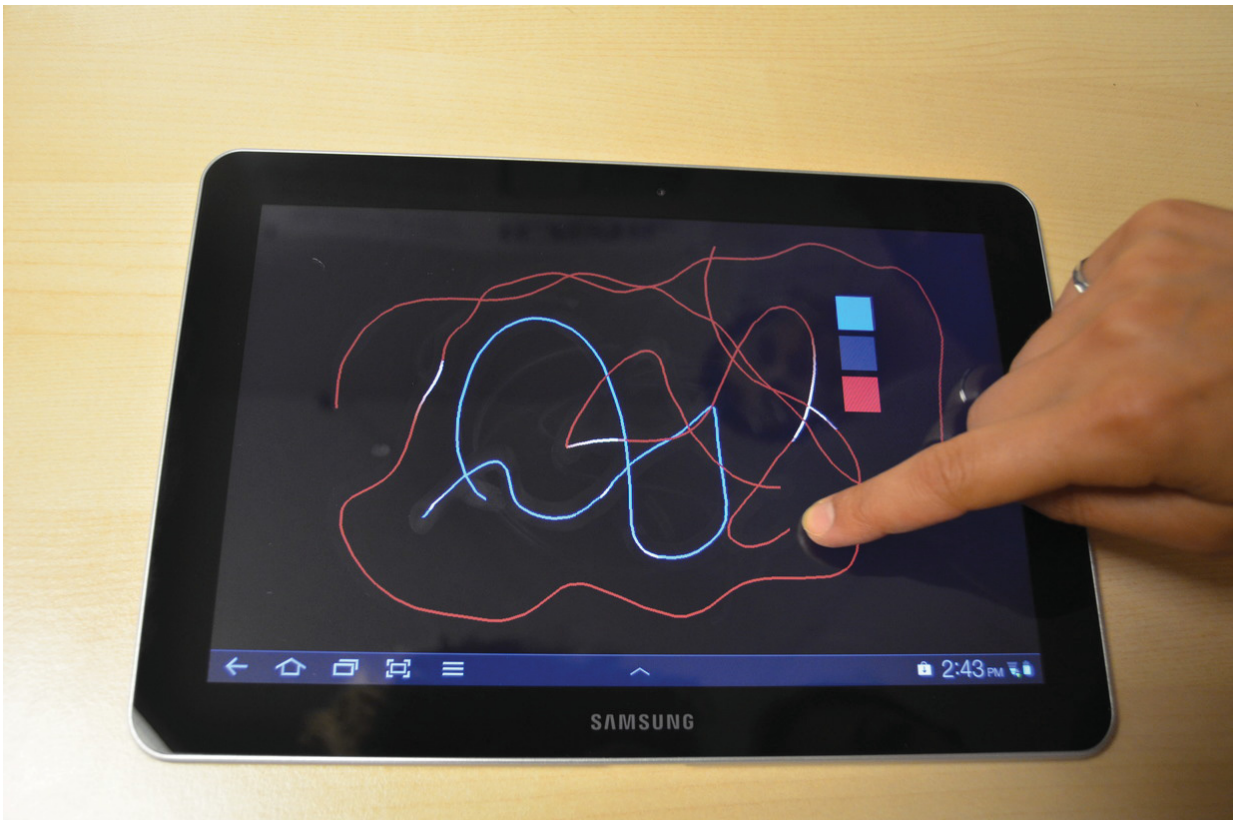

Fig. 3.7 Different Strokes running on an Android tablet, from [6] (used with permission).

As originally conceived, the particle engine in DS is hard-mapped to an included synthesizer, however in discussion with the designer it became clear that it would be desirable to allow for more flexible mapping, to allow both alternative synthesis engines and the control of different musical parameters and structures. An analysis of the DS system includes several data structures and concepts to which we might apply instances:

cursors – Recent versions of Different Stokes support multitouch/multicursor input. The input-side of DS could declare a libmapper signal `cursor` with dynamic instances to allow mapping connections from libmapper-enabled multitouch controllers.

stroke types – DS has a fixed number of stroke types available through shortcut keystrokes or a palette. Since the stroke type is static once chosen, and has associated semantic

meaning (colour), it seems better to implement stroke types as different signals rather than instances of the same signal.

strokes – Each stroke could be represented as an instance of some signal. Again, in DS strokes are static once drawn, and thus do not provide much scope for real-time mapping.

particles – The most useful application of instances. Using system defaults, each stroke can have 0–3 active particles at any given timestep, each associated with a playback pointer into the audio sample buffer associated with the parent stroke type. Mapping of particle position and/or velocity is necessary for recreating Zadel's original mapping concept.

In [6], Zadel describes in more detail the integration of libmapper into DS, a project which also involved constructing language bindings for the SuperCollider programming language. For this work, both input and output signals are declared for mapping, however support for multiple instances was only added for the output-side. DS particles are represented as instances of the signal `stroketype/N/particle/position` and `stroketype/N-/particle/rate`, where `N` is the id of the stroke type. Integration of input-side multi-instance mapping (to DS cursors) is left for future work.

3.7.2 Influence

Another of our long-term collaborative projects concerns the design of interactive systems using distributed sensing and media synthesis [44]. As part of this project, we have been working on a program called Influence[1] using libmapper to form mappings between distributed agents (each is a separate process running somewhere on the network) and a centralized environment where they can interact with each other while consuming minimal bandwidth (figure 3.8). Connected software agents set their position in a 2D space, and video convolution is used to propagate influence as a vector-field between them; the output of the environment is a vector observation of the environment at each agent's position.

Since its purpose is to handle multiple agents, this was an obvious test application for the use of libmapper's instance functionality. With multiple client programs connected, the connection topology should resemble a star but with cycles defined through from each

[1] http://github.com/idmil/influence

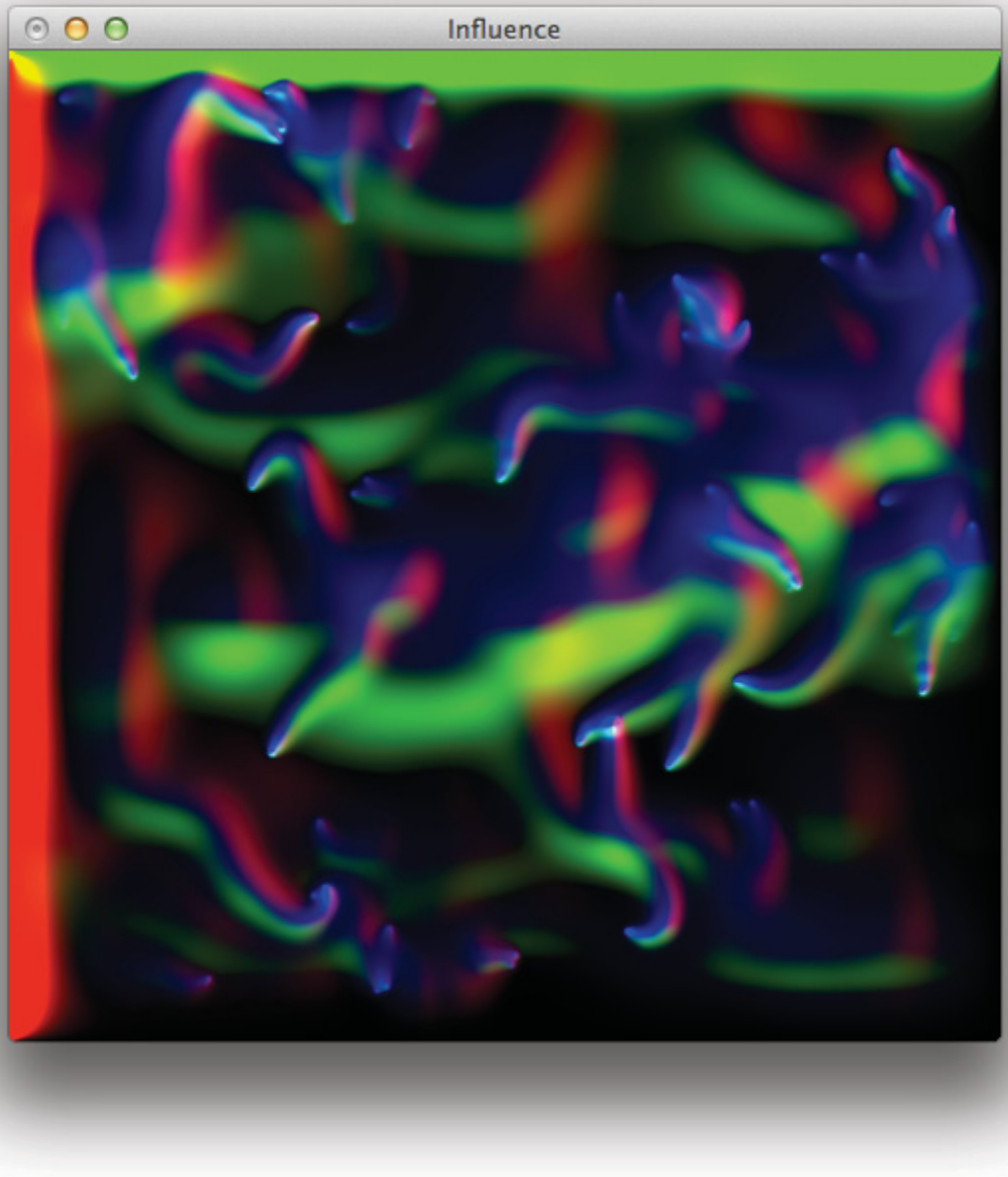

Fig. 3.8 The program Influence running with 50 remote agents connected.

client, through Influence and back; and libmapper needs to ensure that the observations for each local instance are sent back to the correct client.

A client program was implemented for testing interactions with the Influence environment: *passiveAgent* instantiates a variable number of agents, each of which outputs its position (with a randomized starting point) and exposes input signals for velocity, acceleration, and force. By default, the program automatically connects itself to a running copy of Influence, mapping client agent position to Influence agent position, and Influence observations to the client agent force; links are automatically scoped to carry instances initiated by the client programs. This implements a rough physical model in which agents are repelled by each other.

3.7.3 The T-Stick Digital Musical Instrument

The T-Stick digital musical instrument (cf. Chapter 4) features inertial movement and orientation sensors, pressure sensors, and a one-dimensional array of capacitive touch sensors [1]. In the software driver for the instrument, raw multitouch data is processed to identify the position and width of individual touches and grips, and also to extract "gestures" such as brushing the surface with a finger or a hand. In early mappings this information was used only in aggregate form (e.g. amount of coverage, or amount of brushing over a sliding window) since libmapper did not support generic references to instances.

Additionally, a "jab" gesture with the instrument has been used prominently in several mappings to trigger controllable excitation of synthesis models or ballistic "launching" of some musical process. Subsequent "jabs" would either add to the pre-existing excitation level, or re-trigger the process, whether or not the process had come to an end. The ability to trigger multiple overlapping notes of processes with serially-performed gestures did not exist.

To remedy this situation, support for libmapper instances was added to the language bindings for Max/MSP[2] (in which the T-Stick driver software is implemented), and the driver modified to make use of the new capabilities. Touches, brushes, and jabs are updated as instances of their respective signals, and have been tested in simple mapping to our demo synthesizers; adaptation of the synthesizers used for public performances of the T-Stick is still in progress. It is important to note that individual connections can be configured to

[2] http://github.com/idmil/mapper-max-pd

transport updates as multiple instances or force them all to update the same destination instance, and that our legacy mapping designs are not "broken" by the changes to the T-Stick driver.

3.8 Conclusions

In this article, we have identified the need for representation of entities with multiple instances in tools for designing interactive systems, a feature that is only partially present in existing tools. After formalizing a specification for such support based on abstract consideration of node connectivity, we have implemented a solution and integrated it into an existing open-source software library for forming configurable runtime connections between distributed hardware and applications. Finally, we provided several examples showing adaptation of existing software and hardware to utilize the new multi-instance features.

We hope that this work will benefit designers and users of interactive media systems by providing interconnection and protocol-compatibility between different tools while allowing them to be represented logically and intuitively within their specific contexts.

3.8.1 Future Work

Moving forward, we must continue to exercise and test the multi-instance implementation and API against real-world use-cases. In particular, scenarios with very large numbers of instances or in congested networks present challenges for any real-time control system, and will highlight any remaining deficiencies in our approach or implementation. Our recent work bringing support for TCP streams between libmapper peer devices will be tested for the transmission of instance release requests.

The implementation of bridge software between libmapper and the MIDI and TUIO protocols is also underway, and will allow more widespread use with existing commercial input devices and tangible user interfaces (TUIs) while bringing more flexible mapping support to users of these protocols. Although not yet implemented, software tutorials showing how to interface popular computer vision platforms (e.g. OpenCV) to libmapper will also demonstrate the utility of these tools to another community.

3.9 More Information

More information on libmapper and related projects can be found on the project website libmapper.org or by subscribing to one of the mailing lists (developer or user). The website includes online documentation of the libmapper application programming interface, pre-built binary versions of the library for various platforms, and links to projects and utilities using libmapper. All libmapper development is performed in open consultation with the community mailing list, and anyone interested can join to participate in defining and implementing the future of the library and its surrounding ecosystem.

3.10 Acknowledgements

Development of *libmapper* has been supported by funds from the Fonds de recherche sur la société et la culture (FQRSC) of the Quebec government, the Natural Sciences and Engineering Research Council of Canada (NSERC), and the NSERC/Canada Council for the Arts New Media Initiative.

Part II

Case Studies: New Digital Musical Instruments

Chapter 4

The T-Stick Digital Musical Instrument After Eight Years of Development

The following chapter presents the first of two case-studies for development of new digital musical instruments for use with the libmapper software ecosystem. This DMI was developed in parallel with early versions of our mapping tools, development that has helped to drive the further development of libmapper by providing a consistent user-base and a demand for concert-grade dependability. The chapter has been prepared as the following manuscript:

J. Malloch and M. M. Wanderley, "The T-Stick digital musical instrument after eight years of development, " Manuscript prepared for submission.

4.1 Abstract

This paper describes the conception of a new family of digital musical instruments — the "T-Sticks" — and their subsequent development and refinement over the course of the eight years. More than twenty copies of the instrument have been made, and it has been performed at dozens of public concerts in multiple countries and by multiple performers. Three generations of hardware design improvements are explained, and the refinement of gesture extraction and mapping approaches is outlined.

4.2 Introduction

Digital musical instruments (DMIs) can take many different forms, but the fact that physical quantities are sampled and represented in abstract digital form means there are no hardwired associations between gesture and sound. Unlike acoustic instruments, there is no physical reason for blowing or bowing with more force to cause a louder sound, or for small instruments to produce higher pitches. Instead, any connections between performed gesture and produced sound (or other synthesized media) must be *designed*; we usually refer to the collection of these connections as a *mapping* between sources of data such as sensors, and destinations for data such as the parameters of a synthesis algorithm.

In this paper we will focus on the history and development of one DMI project in particular: the "T-Stick" (figure 4.1). Development of this instrument began towards the end of 2005; since then it has achieved a rare sort of "success" in that — unlike most new instruments — it has been performed publicly dozens of times and in many countries, including solos, duets and trios with other DMIs and with traditional instruments, ensembles and even as concerto soloist with laptop orchestras. While it remains an academic project and has never been produced commercially, nearly 20 of the instruments have been built. Over the years we have published some brief descriptions of the early designs [99, 100, 101] and our experiences practicing and performing with the T-Stick [102]. This article will attempt to fill in some substantial holes in the public documentation of this project, outlining the motivations, conception and subsequent development of the instrument and its use in performance and pedagogy over the last eight years.

4.3 Conception

Our past experiences developing new DMIs and working with those developed by others has been interesting and rewarding, however we have observed that there are certain recurring problems with many DMIs. In particular, the majority of new DMIs we encounter are not robust enough to support extensive rehearsal and performance. The flaws are sometimes mechanical (parts come unglued, exposed wires break, etc.) or they may be technological, for example sensors which work well under controlled laboratory conditions but do not work so well in the "real" world. The increasing availability of digital fabrication (e.g. laser cutters and 3D printers) is helping in this regard, as designers can create custom

Fig. 4.1 Composer/performer D. Andrew Stewart playing the soprano T-Stick. (Photograph: Vanessa Yaremchuk, used with permission)

cases or shells for their instruments.

A second issue concerns *commitment* to the new instrument. Inventing new instruments is obviously a fascinating pursuit, and we do not wish to be discouraging, but we should admit that creating a new "musical instrument" and then putting it on a shelf is not particularly meaningful. Too often grand claims are made in reports on new instruments (often including the dreaded term "expressive"), but to evaluate a new instrument it first needs to be *played*. We are not saying that the instruments aren't excellent, the point is that we don't yet know, and *neither do their creators!* In fact, if the instrument is not played we will never know if robustness would have been a problem, let alone whether the instrument would have been interesting to play.

The T-Stick started with a proposal written by the first author and the composer D. Andrew Stewart, applying for project funds from the Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT) to develop a new DMI. At the discussion stage before designing the T-Stick we decided that we would try to produce an instrument that would be robust enough to support many hours of practice each day, *without the creator present*. To achieve this goal, we decided that all sensors and electronics would be properly protected and that there would be no moving parts. A strong shell or case would be essential, and would also play a role in "hiding" the sensors from the performers and

encourage them to perform gestures of sounds rather than playing the sensors.

Since we were strongly interested in exploration of mapping approaches (both as academic exercise and in search of interesting possibilities for performance), we decided to adopt a simple geometric shape for the new instrument. By avoiding idiosyncrasy in the physical appearance (we use the term "low specificity") we hoped to maximize the extent to which the mapping could define perception of the instrument by performer and audience. We decided on a simple cylindrical shape with no protrusions or attachments, which would allow us to keep all of the electronics safely tucked away in the interior.

4.3.1 Sensing

We also decided to design the sensing for the new instrument such that as many of its affordances as possible would be measured. Our instrument would allow multiple ways of exciting and modifying sound rather than attempting to design "correct" playing techniques. Since the sensors would all be attached to the same simple object, their signals would be highly inter-related [14], perhaps in a similar way to the interconnectedness of acoustic systems. To avoid the risk of excessive revisionism, perhaps it is interesting to directly quote our original project proposal:

> Our experience of traditional acoustic instruments corresponds to a physical view of the universe (a simple example: big instruments make low-pitched sounds, small ones make high- pitched sounds; large and fast performance gestures produce loud sounds, etc.) As this perception forms part of the context of our experience of music performance, and forms the basis of a performer's interaction with their instrument, it is of utmost importance that this be considered carefully by a prospective instrument-builder. In this project, we propose to use our different viewpoints and experiences to take a holistic approach: the gesture vocabulary, sensor placement, mapping, and synthesis, while not bound to exactly replicate the response of purely physical systems, should have an internal consistency and logic that permits a prospective performer to quickly and easily begin to make sense of the interface.

To this end, we designed to explore sensing of touch, pressure, movement, and orientation.

4.3.2 Family

Lastly, the new DMI would actually consist of a *family* of related instruments with the same basic structure and playing technique but differing in size. Obviously since the mapping is separable from the physical interface all of the family members could easily be made to sound exactly the same, however we planned to adjust the mapping and synthesis so that relationships between performer gesture, energy-expenditure and the pitch, volume, and timbre of the sound would be related in ways that are familiar from our experience with acoustic systems. We considered that a family of DMIs could be used to explore similar ensemble dynamics to traditional practice (e.g. string quartets), and might help "parallelize" the process of creating contexts in which the new instrument could be perceived and experienced: creating opportunities to observe multiple performers, each with their own subtle variations of performance technique.

4.4 First Generation

The first generation of T-Stick hardware was developed from September 2005 to March 2006, and took place in the context of a seminar on development and performance of DMIs, as well as the launch of the McGill Digital Orchestra project [103, 104]. This project brought together composers, performers, and instrument-designers from our lab to collaboratively develop physical instruments, performance techniques and vocabularies, and compositions over a three-year period. Developing the T-Stick in this context meant that we had access to extremely accomplished instrumental performers, who were tasked with mastering the new instruments.

4.4.1 Development

Following our concepts of a cylindrical instrument, we began exploring different materials for construction and settled on 5cm diameter PVC plastic plumbing pipe as an affordable, but fairly strong substrate for our instrument, easily held in the hand but massive enough to require some deliberation and effort to wield. For the first instrument to be built – the tenor – we decided on 1.2 meters as a starting length (see table 4.1).

Experiments with small mockup sections of pipe allowed us to refine an implementation of capacitive multitouch sensing using an array of sensors made from strips of copper tape

Model	Length	Diameter	Touch sensors (1st generation)
Tenor	1.2 m	5 cm	48
Alto	0.8 m	5 cm	32
Soprano	0.6 m	5 cm	24
Bass	1.8 m	11.5 cm	72

Table 4.1 The family of T-Stick instruments consists of three different models (sizes). A planned fourth model – the *bass* – has not yet been built.

fastened to the outside of the pipe. To allow for installing the sensors and other electronics inside the pipe while keeping trace lengths short, we cut the pipes in half lengthwise with a saw; small holes were drilled through each copper strip close to the edge for wires to pass through before being soldered to the tape. The copper strips were connected in groups of six to overclocked Quantum Research Group QT161 integrated circuits (since discontinued) to provide the capacitive sensing (figure 4.2); their outputs were connected to daisy-chained shift registers allowing very fast acquisition of touch data. Later, a sleeve of large-diameter shrink tubing would cover the entire structure to hold the sections of pipe together and provide strength and protection from sweat.

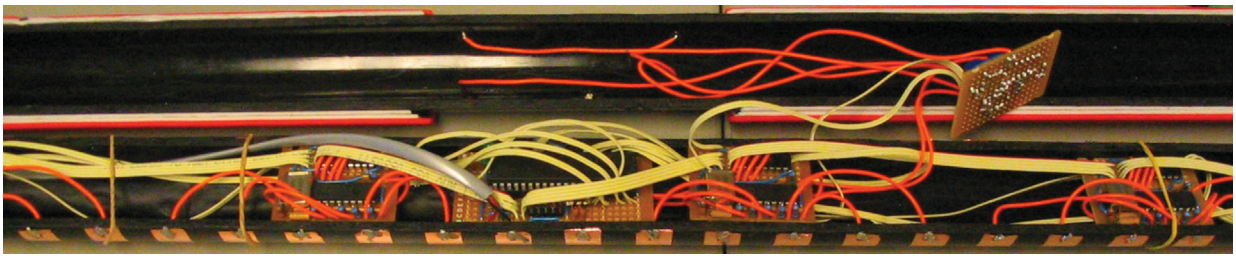

Fig. 4.2 A view inside the first T-Stick prototype, showing touch sensors formed using copper tape wired to the small circuit boards. The small board wired to the top half of the instrument contains voltage dividers for the pressure sensors and a buffer/envelope-follower for the piezo contact microphone. The large IC in the middle is the main micro-controller tasked with sampling analog signals, controlling the shift registers and providing USB communications.

The other half of the PVC pipe was covered with a custom pressure sensor constructed using conductive paper and foil [105], which was covered in turn with a layer of closed-cell foam padding. This padding served two purposes: to improve the controllability of pressure by providing displacement feedback to the performer, and to suggest a "squeezing"

affordance when the instrument is held.

A three-axis accelerometer was also included to sense movement and orientation (by sensing acceleration due to gravity). All of the analog sensor signals and the digital touch data are sampled by an ATMega16 micro-controller running the AVR-HID firmware [106]. Lastly, a piezoelectric contact microphone was added in a effort to sense the impact of new touches, to distinguish between touches, taps, and hits of thumps.

4.4.2 Frets and Spikes

We explored the use of "frets" between the touch sensors to enable tactile navigation of the playing surface. These were formed simply, by wrapping medium-gauge wire around the touch-sensing half of the instrument body before encasing it in shrink tubing. While the frets certainly added a tactile dimension to brushing and navigation of the surface, we quickly found that the frets also encouraged performers to perceive the individual touch sensors as discrete points of interaction rather than as part of a (albeit fairly low-resolution) multitouch-sensitive surface. Considering the individual sensors as "keys" as if on a piano is certainly an affordance of the chosen hardware, but neglects more interesting interaction modes such as brushing and rubbing. Additionally, the specific spatial resolution chosen for the first prototypes was intended to be improved in later models, and we wanted to avoid establishing performance techniques that depended on the original resolution. As a compromise, the second T-Stick to be produced (an "alto") featured only a single fret placed at the centre point.

The first tenor T-Stick also featured a 0.5m "spike" that could retract into the body of the instrument or be adjusted in length like that of a cello. The spike could be used to support the instrument at a comfortable height for playing with both hands simultaneously, while still freely allowing the instrument to be tilted and rolled; for hand-held performance modes we usually removed the spike entirely to reduce the weight of the instrument.

4.5 Second Generation

Improvement of the T-Stick hardware began with the touch sensing — the first two instruments to be constructed involved an astonishing amount of work wiring and soldering the 48 touch sensors (for the first tenor prototype) and associated electronics. We wished to increase the resolution of touch sensing, but sensibly decided to find a more efficient design.

To this end, we designed custom printed circuit boards (PCBs) to integrate 24-channels of capacitive touch sensing (figure 4.3). With a doubling of touch resolution, this enabled a multiple of the boards to be daisy-chained for each model of T-Stick: two for each soprano (48 channels), three for an alto (72 channels), and four PCBs for each tenor (96 channels). Narrower copper tape (0.25") was used to form the sensing array as before.

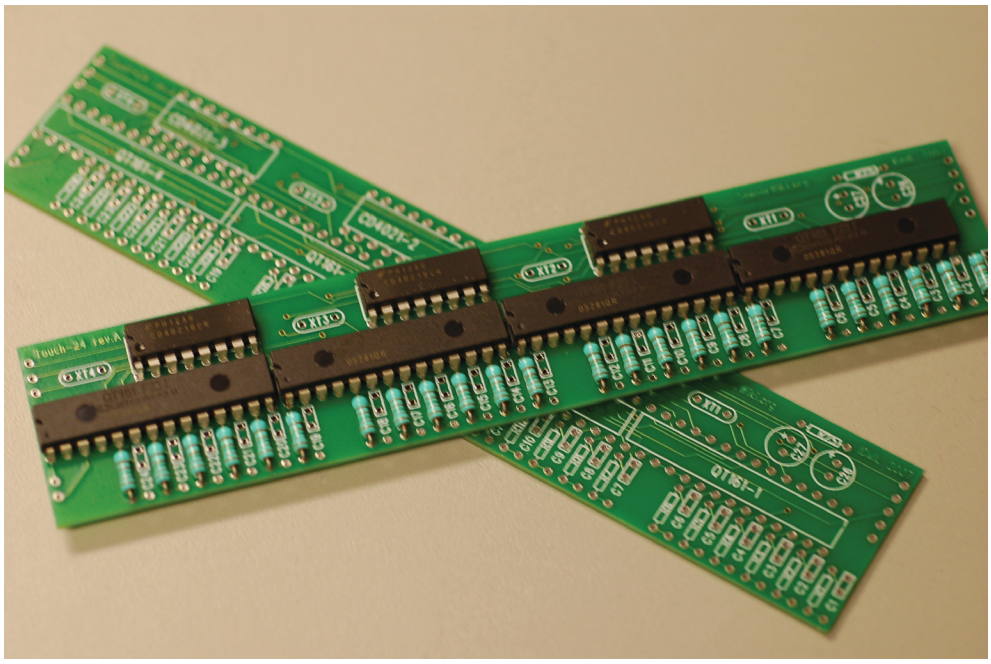

Fig. 4.3 Printed circuit boards developed for the second-generation T-Stick hardware. Each board handles 24 touch sensors.

4.5.1 Firmware Improvements

Starting with the third T-Stick, we began using the commercial Arduino Mini board for the T-Stick micro-controller, allowing firmware to be written and compiled more simply than our previous solution. The first T-Sticks appear to an operating system as Human Interface Devices (HID), however the Arduino Mini does not have this capability and necessitated use of a custom serial protocol for communicating with our driver software written in Max/MSP. Software bugs in the available serial objects for Max/MSP 4.x forced us to use polled communication, in which the software requests sensor data at a fast rate: in this case the fastest rate possible was approximately 100 Hz[1]. After upgrading to Max 5 polling every

[1] "Pushing" data continuously from the T-Stick micro-controller inevitably led to software crashes

6ms became possible, making gesture-to-sound latency slightly better, but the situation was still far from ideal - especially since there is inevitably additional latency from sensor data processing, synthesis, and audio output buffers. Jitter was an even greater problem, since unlike predicable latency it cannot be internalized and anticipated by the performer. Polling the T-Stick results in increased jitter since it is unlikely that performer gesture will be timed to the sampling frequency - a performed gesture might be communicated immediately, or up to the polling period (6ms) later.

The effects of latency and jitter were particularly obvious in the touch data, especially when mapped to sound onsets or other perceptually-salient parameters. This drastically decreased timing-controllability (a major metric for DMI usability [22]), resulting in extreme difficulty when trying to accurately play complex rhythms or play in synchrony with another performer. An examination of the effects of sensor sample rate on timing controllability for the T-Stick revealed that there were different requirements for different signals. The touch data are a collection of binary states ("touched" or "not-touched" for each position) and only need to be communicated when they change, whereas pressure and acceleration are better transmitted continuously at some rate. The firmware was revised for multirate communications, with touch transmitted as soon as it occurs resulting in less than 1ms of latency; other data are sent at an adjustable rate, usually every 10ms, with a "heartbeat" messages sent from the driver periodically to prevent crashes. For now, higher-level parameters such as grip width and brushing trajectories are calculated in Max/MSP, but in the future will likely calculated on-board the T-Stick.

4.5.2 The T-Stick in Class: Pedagogy

Starting in 2009, T-Sticks were constructed as part of several seminars on the design and construction of digital musical instruments at McGill University (figure 4.4). During the first class, eight soprano T-Sticks and one tenor were constructed, under the supervision of the authors and following the 2nd-generation designs. While learning about the design and construction of the T-Stick, soldering and wiring, the student's work also served to drastically increase the number of instruments available for performance projects in our lab and for outside loan. Building T-Sticks as a pedagogical exercise also required much more extensive documentation, including part lists, supplier information, step-by-step instructions, photographs and diagrams.

As some of the parts have been discontinued by the manufacturer, for subsequent seminars students have constructed a simpler and smaller "sopranino" version of the T-Stick hardware. While unfortunately not resulting in a performance-grade instruments, this project still teaches the basic soldering, wiring, and firmware programming skills; after which the students typically move on to building a new DMI of their own design.

Fig. 4.4 Class photo after the first Digital Musical Instruments seminar to involve the construction of T-Sticks.

4.6 Vibrotactile Feedback for the T-Stick

It is generally accepted as fact that performers of traditional musical instruments receive important feedback from their instruments through the sense of touch [107, 108, 109]. The kinaesthetic part of this information is generally preserved for performers of digital musical instruments — even contact-less musical interfaces such as the theremin make important use of the performer's proprioception. Vibrations transferred from instrument to performer, however, are usually missing when performing with DMIs, since the user interface is typically separated from the sound production machinery and thus does not vibrate acoustically.

Various approaches have been taken by researchers and instrument builders to reintroduce vibration feedback to digital musical instruments. This section describes exploration of vibration feedback for the T-Stick beginning in the fall of 2007, and discusses issues and implications of this addition for mapping and user-interaction.

4.6.1 Examples of Vibration Feedback in DMIs

Bert Bongers has made use of several actuator technologies for providing vibrotactile feedback in DMIs [110, 111, 112]. Versions of the *Tactile Ring*, using solenoids and loudspeakers, have been used for playing various "In-Space" instruments including gloves and the LaserBass.

Rovan and Hayward also designed and used ring-based actuators, in addition to stronger, foot-sensed vibrotactile actuators, to improve feedback when playing open-air musical controllers [113]. Their software, *VR/TX*, generates *tactile stimulation events* according to input from the open-air Dimension Beam controller.

Marshall [114] took the approach of reintegrating control surface and sound production: by including amplifiers and speakers in his *Viblotar* and *Vibloslide*, the user is exposed to vibrations inherently linked to the sound produced. It is notable that while this approach emulates traditional acoustic instruments, it does not allow supplementary feedback to be transmitted inaudibly.

Birnbaum designed and constructed the *Tactilicious Flute Display*, an interface for tactile display resembling an end-blown flute [115]. He also implemented the Max/MSP software package *FA/SA* for extracting perceptual features of Break-Beat audio and performing signal transformations to make the features perceptible to the fingertips. This software is designed to drive small voice-coil actuators, but has also been adapted for experiments with other actuator types, including electric motors loaded with eccentric masses as used in "rumble packs" for video game controllers. The *FA/SA* software was later improved by [116].

4.6.2 Actuator Choice

Initial prototypes of actuated T-Sticks were intended for exploration of possible hardware configurations as well as difference mapping strategies; as such, future T-Sticks may be constructed with a variety of different numbers and types of actuators. Although a hypo-

thetical future design may include hard-coded feedback and actuators capable of only very specific responses, a more flexible, general purpose actuator is required for experimentation with mapping strategies. Specifically, we were looking for large frequency and amplitude ranges, a high maximum amplitude, good transient response, and control over the phase in a small form-factor.

Marshall provides a detailed comparison of different vibration actuator technologies [117]. Using this information and the list of desired characteristics above some actuator types can be ruled out for use in this project. The Tactor, for example, has a low maximum amplitude and thus is probably unsuitable for driving the mass of the T-Stick. Rotary electric motors, while providing appropriate maximum amplitudes, do not permit suitable control of phase or independent control of frequency and amplitude, and most solenoids are constructed so that they are either open or closed, and thus do not allow control of amplitude.

Another actuator type, used in the MicroTactus actuated probe [118] and an experiment in haptic perception of a virtual rolling stone [119], was eventually chosen for adding vibrotactile feedback to the T-Stick DMI. Most importantly, it provides high maximum amplitudes, large amplitude range, and control over phase, all in a form factor that easily fits inside the case of the T-Stick. At the time of construction this actuator was not available commercially and had to be constructed by hand-winding electromagnet wire onto machined plastic sleeves; an improved model is not available[2].

4.6.3 Integration

The actuator was encased in shrink-tubing for protection and bonded strongly to the interior of the T-Stick using epoxy adhesive, in order to ensure coupling between the actuator and the instrument. It was decided to use the same actuator orientation as that used for the MicroTactus and the rolling-ball experiment, in which the actuator is driven in the axis corresponding to the length of the tube. This makes the effects of actuator placement negligible: the stiffness of the ABS plastic pipe means that vibrations are not noticeably damped from one end of the pipe to the other.

The actuator was driven using audio signals generated by software running on a laptop computer (see figure 4.6.3). The actuator exhibits impedance similar to that of an audio

[2] http://www.tactilelabs.com/products/haptics/haptuator/

speaker, and for this initial investigation a simple 1W 5V audio amplifier circuit was used, using a Philips TDA7052 mono amplifier IC. More powerful amplifiers may be used for future work with the actuated T-Sticks.

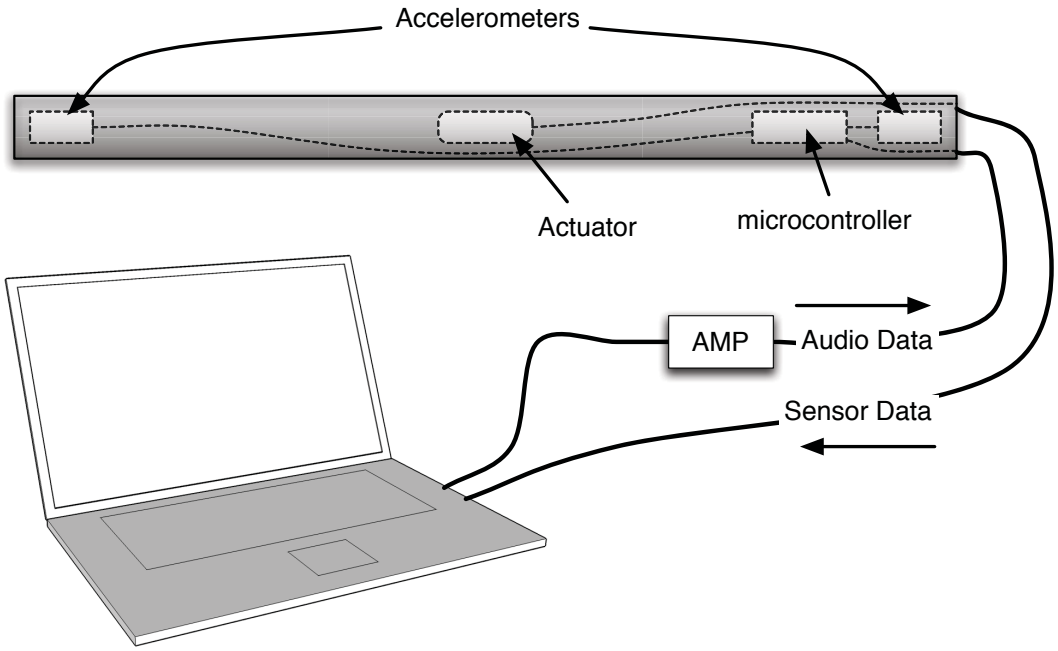

Fig. 4.5 The equipment used for driving the actuated T-Stick.

4.6.4 Mapping

For the purposes of this initial investigation, it was decided to create two competing implementations of vibrotactile feedback mapping, corresponding to two different interaction domains from [20].

Sign-based mapping

The first mapping approach involves the use of discrete vibrotactile cues provided to the user/performer to help them navigate the multidimensional sensor-and-sound parameter space implemented for the T-Stick. As an initial test of this approach, an amplitude-enveloped waveform was sent to the actuator, creating a "buzz" sensation intended as a discrete cue. This was mapped to a discretized version of the controller tilt data, such that

crossing boundaries between discrete zones of tilt results in a vibrotactile cue. Large zones (45°) were used initially; the use of much smaller zones is only limited by sampling noise in the tilt data.

Signal-based mapping

The second mapping approach instead uses a simple signal model to create a haptic illusion, convincing the user that the controller exhibits physical dynamics it does not in fact possess. While not actually changing the controller's physical dynamics, haptic illusions can profoundly affect the way in which the user interacts with the device.

A signal model of a virtual rolling ball was implemented in Max/MSP following the description in [119]. Since the controller already contains acceleration sensing, it is simple to link this to virtual physical dynamics consistent with real-life gravity and user-interaction. The acceleration signal is integrated to approximate velocity, and the resulting signal is used to control the frequency of a periodic signal mimicking the rolling of a ball of a set circumference. By varying the scaling of acceleration data and the scaling of velocity data, the mass and circumference of the virtual ball may be altered. Performing waveshaping of the final signal (or altering the stored waveform if look-up tables are used) alters the perception of the interaction between the virtual ball and the inside of the tube, creating the impression that the motion is smooth or bumpy, or that the inside of the pipe is ribbed. Integrating a second time approximates the position of the ball; this data is used to stop the virtual motion and set the velocity back to zero when the ball reaches the end of the modelled pipe.

4.6.5 Multi-actuator Version

In 2008 a more complex integration of actuators was produced for research on Enactive Interfaces[3]: this version included four separate vibration actuators of the same design, arranged two at each end and mounted orthogonally to the length axis of the instrument. Although the direction of a particular vibration is not likely to be well-perceived, this arrangement allows for perceivable panning effects along the length of the pipe.

[3] http://www.enactivenetwork.org/

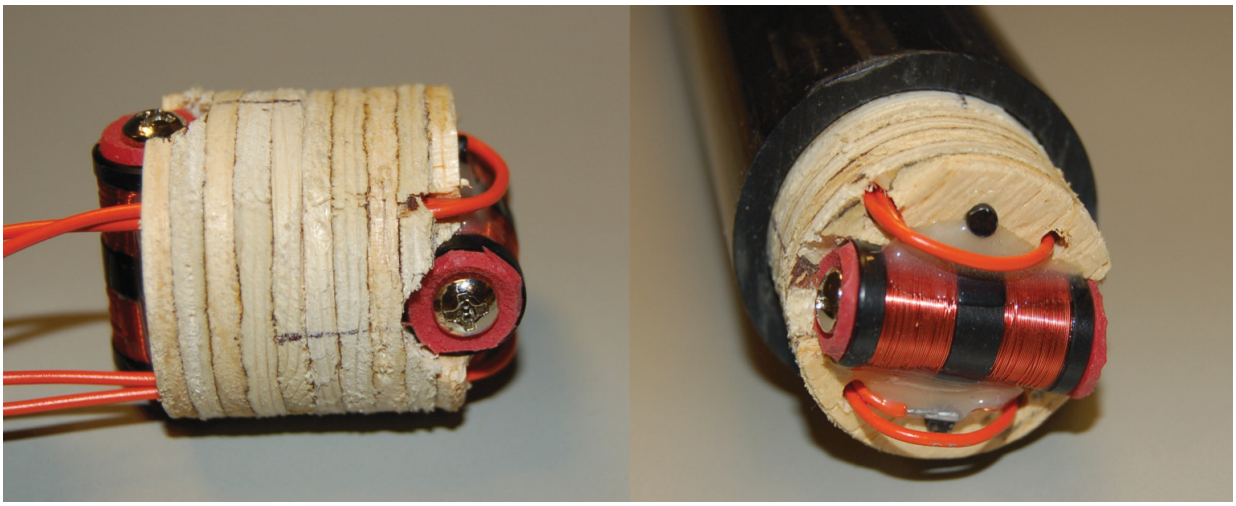

Fig. 4.6 A prototype with multidimensional vibration feedback built for the Enactive Interfaces project: a wooden harness is used at each end of the T-Stick to mount two linear vibration motors orthogonal to the length axis of the pipe. This allows stereo panning effects along the length of the pipe.

4.7 The SpaT-Stick

In the autumn of 2008 we were approached by composer Sean Ferguson and choreographer Isabelle Van Grimde regarding a possible collaboration using the T-Stick for a new performance of live dance and music: *Duo pour un violoncelle et un danseur* ("Duet for a 'cello and a dancer"). In term of instrument requirements, there would be two main departures from the existing version: the instrument would be performed — at least part of the time — by a professional dancer rather than a musician, and the composer wished to make use of sound spatialization as an important dimension for real-time control. The fact that the instrument would be used by the dancer also necessitated that the instrument be able to operate wirelessly, since the choreography would include working in a fairly large area and it was judged that long cables would be awkward and unsightly. During planning for the piece, the choreographer indicated that she wished to use the T-Stick as a metaphorical limb in ground-work, meaning that the instrument would be required to support the dancer's body weight as an extension to his arms.

With these considerations in mind, we decided that a new construction technique would be required, since the existing approach involved cutting the PVC body structure in half in order to build and connect the capacitive sensors. Although the PCV pipe began as a fairly

sturdy object, cutting increased its flexibility — and reduced its strength — considerably. A new approach using flexible circuits for the capacitive sensors was designed, in which the sense electrodes are patterned on a sheet of flexible plastic and "rolled up" around the remaining electronics before being inserted into an (intact) PVC body. Commercial flex circuits of the size required (approximately 60 cm x 15 cm) were much too expensive for our limited budget, so we built our own using plastic sheets and copper tape connected to a revised version of the touch24 boards mentioned earlier (figure 4.7). This rough approach resulted in lightly noisier touch sensing, since there was both more capacitive coupling through internal components (since they are in closer proximity) and a weaker signal from actual touch events since the plastic shell is thicker than the previously-used shrink-tubing. These problems necessitated increased smoothing of the touch data before use, however the tradeoff for increased mechanical strength was judged wise in this context.

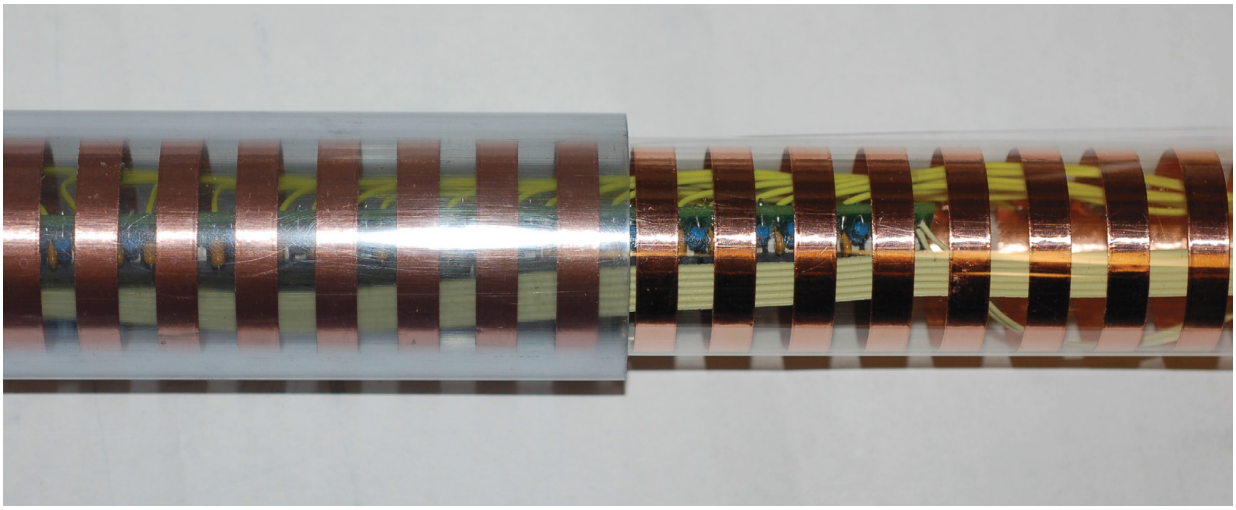

Fig. 4.7 A flexible version of the capacitive sensor layout enables the electronic components to be rolled up and inserted into a transparent PVC body.

The resulting structure was of course slightly bulkier than the circuit boards and wiring alone, so we moved to a slightly larger diameter of pipe for our new prototypes. This new approach also allowed us to experiment with different body appearances, including a version with a bamboo body. The body material chosen for the final instrument was transparent PVC (figure 4.8).

Fig. 4.8 Dancer Elijah Brown and 'cellist Chloé Dominguez using the Spat-Stick in a performance of *Duo pour un violoncelle et un danseur*. The orange end-cap indicates the "pointing" end of the instrument.

4.7.1 Sensing Orientation

The collaborating composer planned to use a ring of loudspeakers around the audience, and for the performers to be able to direct the spatialization of sound around the ring during the piece. The standard T-Stick design was capable of sensing *tilt* (the angle of the body with respect to a plane approximating the Earth's surface) and *roll* (rotation/orientation around the major axis of the instrument's body, again expressed with reference to the Earth's surface), but it was not capable of measuring rotation or orientation around the axis perpendicular to the Earth's surface. This meant that we would need to add another mode of sensing apart from accelerometers for sensing orientation, that could relate the orientation of the instrument with respect to something different from the gravity vector.

A three-axis magnetometer was added to the sensing hardware to act as a "digital compass" measuring the orientation of the instrument in relation to the Earth's magnetic field. The data from this sensor needed to be compensated for orientation in other axes using values calculated from the accelerometers, and also calibrated for distortion of the field caused by the adjacent electronics [120]. One end of the instrument was arbitrarily

chosen as the "pointing" end and marked with an orange end cap so that the performers would know the direction they were pointing unambiguously.

This solution allowed the dancer to steer sounds around the performance space by pointing and moving the instrument. Unfortunately, in-budget magnetometers proved to be extremely slow (around five samples per second) meaning that interpolation of the trajectories was required to prevent "jumping" of the controlled sounds. If the instrument was turned more than 180 degrees between samples the interpolator would naturally assume that the opposite trajectory had been performed and spatialize sounds accordingly; for the purposes of the piece the performer simply avoided very fast gestures when controlling sound spatialization. In section 4.8.1 a better solution is described.

4.7.2 Wireless Communications

In order to provide wireless communication, a class 1 Bluetooth radio transceiver was added instead of the USB connector, and a battery pack and power management circuit were constructed and embedded in the end of the instrument away from the magnetometer. This type of transceiver is rated for 100 meters range line-of-sight which was judged sufficient for our needs; in practice the radio was somewhat difficult to work with but worked satisfactorily in rehearsal and performance. The short project schedule did not allow for experimentation with alternative wireless modules or protocols such as WiFi or ZigBee.

4.8 Third Generation

In June of 2011, the first author had the opportunity to work as a visiting researcher at the Universidade Federal de Minas Gerais in Belo Horizonte, Brazil. This period resulted in the development of a new mapping — developed in collaboration with percussionist Fernando Rocha — to enable T-Stick instrumentalists to play parts of a multi-percussion piece. Perhaps more importantly for the development of the instrument, nearly six weeks were put aside to work on refinement of the sensing platform and signal processing, especially to improve the direction and orientation-estimation problems encountered during the *Duo* project.

4.8.1 Hardware Improvements

To this point the only inertial sensors included in the T-Stick were 3-axis accelerometers, used for sensing both movement and orientation with respect to gravity. The major gestures and postures used are "shake," "jab," tilt (elevation), and roll. Of these, "jabbing" has the closest relationship with timing controllability as it is often used as a sound-excitation gesture. Shaking was calculated using a leaky integration of all accelerometers with the gravity component removed with a high-pass filter.

For sensing azimuth, a magnetometer was added to some models of the instrument as mentioned in section 4.7.1; although the acceleration and magnetic field signals are complementary in that they are both affected by the sensors' orientation, magnetometers are notoriously noisy since they are affected by any local disturbances in the Earth's magnetic field, caused by electrical cables, transformers, metal building structures, and natural anomalies. Also, estimation of the gravity vector using accelerometer data is complicated by accelerations due to other forces. Misestimation of the gravity vector will result in a faulty rotation of the magnetic field vector and a faulty calculation of azimuth.

Using only these two sensors, there is a natural tradeoff between noisy data with a fast response and stable data with a slow response. This means that orientation cannot be mapped to salient musical or sound parameters without obviously unintended consequences. Orientation and movement of the T-Stick are visually very striking and as such beg to be mapped to similarly dramatic sonic parameters.

The typical solution is to add another type of sensor, the *rate-gyroscope*, which measures angular velocity around one axis; we used three of them oriented orthogonally. The combination of 3-axis accelerometers, magnetometers and gyroscopes in a single package is commonly called an Attitude and Heading Reference System (AHRS). These systems are used for aircraft stabilization and navigation, automobile navigation systems, and for inertial motion capture systems for virtual and augmented reality [121]. The gyroscopes used here operate much faster than the magnetometers, and could obviously be used alone for mapping angular velocity to synthesizer control. Integrating the gyroscope data, however, can provide an estimate of orientation separate from that calculated using accelerometer and magnetometer data, and unaffected by either linear acceleration or local magnetic field distortion.

Affordable (vibrating-mass) gyroscopes suffer from *bias drift*, especially when the ambi-

ent temperature changes [122][4]. Here the *bias* is the sensor output voltage which represents zero angular velocity, and which is subtracted from each sample to calculate the true velocity - if this value changes, then our velocity integration will also be biased, and will slowly increase or decrease even if the sensor remains stationary. Since the bias changes very slowly, we can consider if to be very low-frequency (almost DC) noise, while our estimates of orientation from acceleration suffer from high-frequency noise from movement of the instrument (for the purposes of estimating orientation this signal is noise, obviously for sensing linear movement it is extremely useful). An adaptive filter can fuse these two orientation estimates into an estimate that is better than either on its own — in our case we use a complementary filter [123] running either in the instrument driver software or preferably on-board the micro-controller since we can achieve tighter timing when integrating the gyroscope measurements.

Mapping of Instrument Orientation

In our sensor-fusion code all orientations are represented as *unit quaternions*, since successive rotations require less computation and the representation doesn't suffer from "gymbal lock" problems [124]. Quaternions are a four-dimensional representation of orientation, which can be very hard to visualize or intuit when approaching the mapping of the orientation data to media synthesis. Nevertheless, we do expose the quaternion signal for mapping (it could be perfect for controlling some naturally four-dimensional parameter-space), and we also expose a translation into more familiar Euler angles: *tilt*, *roll*, and *heading*.

4.8.2 Gesture-Processing Improvements

The addition of the rate gyroscope alone greatly improved the quality of movement processing since it allows us to completely separate rotation and orientation of the instrument from linear accelerations/displacements such as swinging, shaking, and jabbing. In addition to the sensor fusion described above, improvements were also made to signal-processing routines used for extracting brushing gestures, jabbing, and shaking of the instrument[5].

[4] Gyroscopes of much higher quality exist, and are used for dead-reckoning in Inertial Navigation Systems (INS) in missiles for example. The prices of these sensors, however, make them unsuitable for prototyping digital musical instruments.

[5] All of the functions and algorithms for T-Stick gesture processing — including sensor fusion algorithms for estimating orientation, quaternion arithmetic, etc. — are publicly available as part of the Digital

Brushing

In the early days, an extremely basic method was used to extract brushing gestures, which was susceptible to false detection on first touch. The improved algorithm generates five different signals for mapping, using bitmasks to track new touches and releases, and leaky integration to extract signals corresponding to single brush strokes. Another leaky integrator is tuned to produce smooth output over a longer timescale, enabling a performer to build up its internal state with multiple sequential brushing gestures (exposed as `brush/energy`).

Jab Detection

Our original approach to extracting "jab" gestures from the T-Stick was to simply use the derivative of the acceleration signal from either the x-axis or the polar amplitude, depending on whether we wanted jabs only in line with the length axis or in any direction. If the absolute value of this derivative — commonly termed "jerk" — exceeded a threshold, a "jab" would be considered to have occurred and exposed for mapping as `jab/amplitude`.

The addition of gyroscopes greatly improved the quality of the jab direction signals since the orientation could be calculated quickly while minimizing noise from linear accelerations. As we increased sample rates through improvements in the T-Stick firmware and serial drivers on the computer, we also designed an improved algorithm for jab-detection (see results in Figure 4.9). The new algorithm uses lower thresholds for denoising, and can thus detect jab gestures with lower amplitudes.

Shaking

The improved algorithm for processing shaking gestures is based on that for jab-detection, starting with the polar roll signal calculated from the raw accelerometer data to measure movement in the plane orthogonal to the length axis of the instrument. An extra step is also added to force shake events to alternate between positive and negative signs. In order to avoid wrap-around errors (where the raw signal jumps by 2π due to the representation rather than the orientation), the raw signal is first "unwrapped" so that subsequent samples are forced to follow the shortest polar distance. Starting with the `shake/grain` signal, we also calculate and expose the frequency of shaking and its deviation over time — a low

Orchestra Toolbox for Max/MSP at http://idmil.org/software/dot.

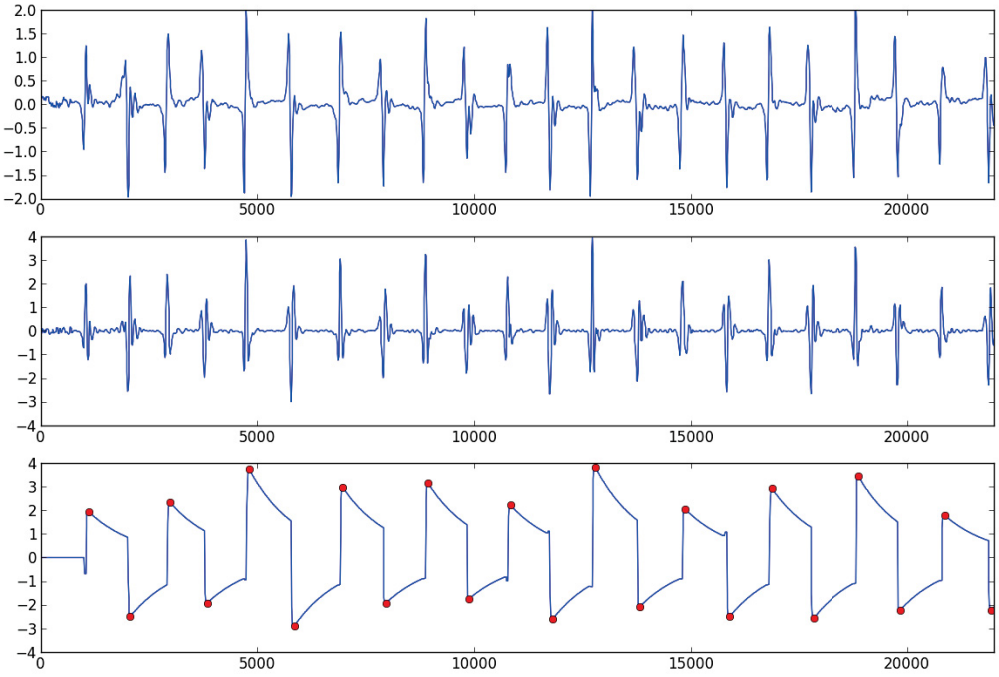

Fig. 4.9 Graphs showing the process of extracting "jabbing" gestures from acceleration data. Top: x-axis accelerometer measurements in g as the instrument is "jabbed" in alternating directions approximately every 1000 milliseconds; middle: windowed maximum difference of the acceleration data; bottom: debouncing envelopes generated from the middle graph using a leaky integrator, with identified "jabs" located at the peaks.

deviation indicating high periodicity and vice versa.

4.8.3 Mapping Support

The T-Stick software makes use of another of our projects — the open-source *libmapper* toolset (cf. Chapter 2) — for supplying discoverability and compatibility with media synthesizers. Specifically, the T-Stick driver software is written in Max/MSP and uses the libmapper bindings for this environment. Over the years we have been making constant improvements to the efficiency and capabilities of libmapper, and these improvements have also improved the usability and reliability of the T-Stick. The addition of support for OSC bundles in the libmapper and the Max/MSP bindings, for example, dramatically reduced the number of IP packets streaming between driver and synth, which makes our software run more efficiently (leaving more cycles for intensive media synthesis algorithms and applications) and reduces the probability of packets being dropped. Continuing improvements to the libmapper graphical interfaces have supported the process of designing new mappings for the instrument, and the support for "querying" the values of remote devices has allowed the use of interesting implicit mapping layers between the T-Stick driver and the synth, for example using machine learning algorithms to "learn" to interpolate between high-dimensional examples [35].

One of the newest features in libmapper is support for signals with *multiple instances* (cf. Chapter 3 in this thesis), a powerful feature beyond the scope of this paper but which has several applications to the T-Stick. The most obvious application is for the representation and mapping of multitouch data, since the capacitive sensing array on the T-Stick can sense multiple simultaneous grips, taps, or brushes. A more subtle use allows the mapping designer to decide whether successive *serial* gestures such as jabs are targeted to one instance of the destination synthesizer, or to automatically use multiple instances of the synth if the natural lifetime of the synthesized sound is longer than the period between gestures.

4.9 Discussion

Our attempt to create a new DMI has been quite successful — the T-Stick has been performed dozens of times and on several continents including conferences, concerts, and

music festivals. Around a dozen performers have played the instrument publicly for audiences, and countless people have played casually at school demonstrations and science & technology expositions. The "T-Stick Composition Workshops" held in 2010 by the composer/performer D. Andrew Stewart involved six additional composers and resulted in the public performance of several new pieces.

Obviously we cannot know for certain which of our decisions helped to make these things happen and which may have hindered our project, but we believe that some factors strongly influenced the success of the result.

4.9.1 Robustness

Building a *robust* instrument was essentially our first priority, well before any specific design decisions had been made for the T-Stick. Prior experiences with new glove-based instruments and video-tracking had left us with a sense of frustration: although the interfaces were interesting in a technological sense, the musical results were typically "cartoonish," more demo than performance. There would certainly be other roadblocks on the way to more interesting performance with DMIs, but even these were out of reach if the performers couldn't spend more time playing and less time waiting for repairs, or worrying about breaking the instrument.

The T-Sticks have proven to be extremely robust, even the first prototype is still working after eight years. They have been played for countless hours, shaken, packed in luggage, shipped, and dropped (even down an escalator, twice) and for the most part continued to work perfectly. Of course, like all instruments they are not indestructible — when the rare repair needs to be made to one of the instruments, the shrink tubing cover can be cut off and replaced after the repair work is completed; the new cover costs approximately $8.

4.9.2 Playing the Sensors

Another of our early aims was to physically hide the sensors and electronics on our new instrument. We often speak of "interaction metaphors" when designing the mapping for DMIs [125], and it seemed to us that maintaining belief in the mapping/metaphor would be easier if we keep the technical aspects of the instrument abstract. This is not to say that we think performers are not intelligent enough to think about engineering or electronics — quite the opposite! The problem is that *somebody* needs to be focussing on the music, and

if everyone is obsessing over the technological aspects of the instrument, important roles are not being filled. The evolution of the instrument and its surrounding contexts may be delayed or stunted as a result.

In the early days of T-Stick development, we quickly realized that our performer-collaborators had already adopted a technical/technological vocabulary for discussing the instruments; instead of framing a concept from the perspective of gesture/body/sound they would typically reference a sensor! We saw this as symptomatic of a reductionist view of the instrument and its performance technique, exactly opposite to our goal of integral, complexly inter-related sensing and mapping.

Although the original performers were already familiar with the guts of the instruments, adding the opaque cover helped us to reframe the discussion. The instrument became a single cylindrical object; once we eliminated frets on the multi-touch surface, it became almost featureless. At this point most our work and discussion revolved around mapping approaches and decisions, and here another strategy helped us reframe the work away from sensors and towards gesture, instrument, and sound [126]. It was around this time that we began the development of what would become *libmapper* — an open-source software library for interconnecting the parts of interactive systems and designing the mapping between them [41] (cf. Chapter 2). The library and surrounding tools are designed around the premise that it is useful to abstract mapping from instrument or synthesizer design, and that multiple perspectives of a particular instrument or system can be beneficial. In the Digital Orchestra Project and for early T-Stick mapping efforts, this meant that for the most part no direct interaction with the driver software was necessary, and that mapping could be defined from actions (e.g. "shaking", "jabbing") or postures (e.g. "tilt") rather than thinking about sensor signals.

4.10 Conclusion

We have presented the conception, design, and subsequent evolution of the T-Stick digital musical instruments over the last eight years. While played for countless hours of practice by a dozen different performers, and public performances in several countries, this instrument has also passed through three main generations of technological development, as well as specific modifications for experiments with vibrotactile feedback and for dance performance.

Development of the T-Stick has not ceased, new mapping designs and new compositions

are still being produced and performed publicly. Additionally, a fourth generation of the hardware is slowly taking form, that will address improvements in touch-sensor resolution, construction, and manufacturability.

4.11 Acknowledgements

Development of the T-Stick digital musical instrument and the the surrounding repertoire and performance practice has been supported by funds from the Fonds de recherche sur la société et la culture (FQRSC) of the Quebec government, the Social Sciences and Humanities Research Council of Canada (SSHRC), the Canada Council for the Arts, the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT).

The authors also wish to thank D. Andrew Stewart for his continuing work composing for and performing the T-Stick DMI, and all of our performers over the years in Canada, Brazil, and Portugal.

Chapter 5

Instrumented Bodies: Prosthetic Instruments for Music and Dance

The following chapter presents the second of our use-cases: the rapid development of new instruments for musicians and dancers within the context of intensive collaborative workshops. The chapter was prepared as the manuscript:

J. Malloch, I. Hattwick and M. M. Wanderley, "Instrumented bodies: prosthetic instruments for Music and Dance," Manuscript prepared for submission.

5.1 Abstract

This paper describes the process of conception, construction, development, and use of a new family of *prosthetic* digital musical instruments. The instruments were produced in active collaboration with dancers, musicians, composers and a choreographer, with feedback and documentation gathered in a series of intensive workshops. The various design constraints are discussed, including issues related to physical form, materials, sensing, ergonomics and aesthetics. The collaboration culminated with an international tour of a contemporary dance/music performance composed and choreographed specifically for the new instruments.

5.2 Introduction

This article concerns the conception, design and fabrication of "prosthetic digital instruments" for music and dance. These instruments are the culmination of a three-year long project in which we worked closely with dancers, musicians, composers and a choreographer. The goal of the project was to develop instruments that are visually striking, utilize advanced sensing technologies, and are rugged enough for extensive use in performance.

The complex, transparent shapes are lit from within, and include articulated spines, curved visors and ribcages. Unlike most computer music control interfaces, they function both as hand-held, manipulable controllers and as wearable, movement-tracking extensions to the body. Further, since the performers can smoothly attach and detach the objects, these new instruments deliberately blur the line between the performers' bodies and the instrument being played.

Starting with sketches and rough foam prototypes for exploring shape and movement, they progressed through many iterations of the design before arriving at the current versions. We made heavy use of digital fabrication technologies such as laser-cutters and 3D printers, accessed first through associated labs and research groups, and later commercially as we increased the number of instrument prototypes produced.

Each of the nearly thirty working instruments produced for the project has embedded sensors, power supplies and wireless data transceivers, allowing a performer to control the parameters of music synthesis and processing in real time through touch, movement, and orientation. The signals produced by the instruments are routed through an open-source peer-to-peer software system the we have developed for designing the connections between sensor signals and sound synthesis parameters (discussed in section 5.7).

Although evolution of the new instrument designs has not ceased, the current versions were featured in recent productions of the piece "Les Gestes" for two dancers and two musicians. The piece was developed in collaboration with our team, and toured parts of Canada and Europe during the spring of 2013.

5.3 Background

In this section we situate our work within the fields of digital musical instrument design and interactive dance performances, and describe the project within which the work takes

place.

5.3.1 Digital Musical Instruments

Throughout human history new technologies have always been adopted and adapted for the production of art. In the case of musical instruments, advances in wood- and metal-working, materials, and the understanding of acoustics have allowed the invention and evolution of music-making tools. Many of the "traditional" instruments we appreciate today are examples of extremely refined technological development.

With the advent of computing technology, it became possible to synthesize and record sound algorithmically rather than depending on physically-vibrating materials. Early work in this field was constrained to off-line applications by the limited computing power available, but as faster computers were developed it became possible to synthesize sound in real-time for at least some algorithms.

"Digital musical instruments" (DMIs) are systems designed for interacting with digital sound synthesis in real time, just as a traditional instrumentalist interacts with an acoustic system in real time. Since the inputs to a given synthesis algorithm are numerical rather than continuous physical quantities, such systems have essentially limitless flexibility as to the composition of the control interface - it's weight, shape, colour, materials may affect *how* the performer interacts with the interface, but they have no bearing on its connectibility. Once the physical aspects of the interface are measured and sampled, the resulting numerical values are just as abstract as the inputs to our synthesis algorithm, and (unless we decide to attempt to model an existing system) there is no *a priori* correctness in any particular configuration of connections between them. For a designer this flexibility can be a bit alarming, but there are still plenty of constraints to consider, and the human-machine interaction research community has much to say on ergonomics, perception of affordances, etc.

This open definition also invites a more structured understanding, and the DMI community is not lacking in classification schemes. Over the years, there have been many efforts to categorize digital musical instruments. Some examples of metrics used include who is interacting [127], the typologies of interaction [30, 11, 128], user expertise [30, 125], location and scalability [129], and type of musical control [16, 130, 20]. A straightforward taxonomy of gestural controllers is that used by [10]:

- augmented or extended instruments
- instrument-like gestural controllers
- instrument-inspired gestural controllers
- alternate controllers

5.3.2 Interactive Interfaces for Dance Performance

The term *interactive dance* typically refers to systems in which the movement of a dancer somehow influences the manifestation of electronic or computer-based music.[1] One of the earliest important examples of interactive dance was Cage and Cunningham's 1965 piece "Variations V" [131].

The vast majority of interactive dance performances utilize motion-sensing technology, which detects the relative motion of the dancer's gestures or their position in a room. The most common methods for this detection are computer vision, infrared motion capture, and inertial measurement. Computer vision systems such as Eyesweb[2] [132], EyeCon[3] and the Very Nervous System [133] use video from one or more cameras and extract information such as the location and velocity of movement in front of a static surface. Infrared motion capture systems from manufacturers such as Qualisys[4] or Vicon[5] use reflective markers placed on the body and an array of cameras to create skeletal models of the movement of a human body. Inertial measurement systems use accelerometers and gyroscopes placed on dancers' bodies to detect the movement of different body parts. Advanced versions of these systems[6] can also create the same skeletal models as infrared motion capture systems [134].

One important conceptual aspect of interactive dance regards the intentionality of the dancers' control of music [135]. To what degree should the dancer's movements be constrained by their desire to achieve a specific musical result? As opposed to musicians, a dancer's movements are predominantly intended to function within a visual aesthetic. The

[1] It is assumed that the resulting sound in turn affects the movement of the dancer; thus, the systems are interactive.

[2] www.infomus.org/eyesweb_ita.php

[3] eyecon.palindrome.de

[4] www.qualisys.com

[5] www.vicon.com

[6] For example, the Xsens bodysuits. www.xsens.com

tension between their ability to stay within this aesthetic while simultaneously interacting with synthesized sound pervades the creation of interactive dance systems.

Crutches and other prostheses have been incorporated into non-interactive dance performances in a variety of ways. Handicapped dancers such as Zhai Xiaowei and Reynoldo Ojeda use functional prostheses to allow for their participation in traditional dance forms[7]. In "bODY_rEMIX/gOLDBERG_vARIATIONS", Marie Chouinard's dancers use crutches and a variety of other prosthetic devices to create distorted and unconventional bodily shapes[8]. However, the use of physical props or prostheses in interactive dance is relatively rare as the dominant aesthetic tends to be to preserve the freedom of motion of the dancer as much as possible.

5.4 Conception and Planning

5.4.1 The "Gestes" Project

The development discussed here took place as part of a project titled *Les Gestes: une nouvelle génération des instruments de musique numérique pour le contrôle de la synthèse et le traitement de la musique en performance par les musiciens et les danseurs*[9]. This project brings together our group – the Input devices and Music Interaction Lab at McGill University – with the choreographer Isabelle Van Grimde, her dance troupe *Corps Secrets*, and composers Sean Ferguson and Marlon Schumacher in a collaborative research-creation project. It is based on an earlier project with the same collaborators (*Duo pour un violoncelle et un danseur*) in which a dancer controlled live processing of a 'cellist using a digital musical instrument – the T-Stick[100] – developed by the first author in our lab (cf. Chapter 4 of this thesis).

Although we used a special wireless, reinforced version of the T-Stick DMI for the *Duo* project, it was essentially very similar to the versions we use for live music performance – differing more in the mapping decisions than in basic structure or appearance. The *Gestes* project was intended to be quite different, since when planning the project we made a collective decision to explore new forms for the instrument, specifically playing

[7] mentalfloss.com/article/22289/dancing-crutches

[8] http://www.mariechouinard.com/body-remix-golberg-186.html

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

with physically attaching the instruments to the dancers at times to form metaphorical prostheses used to generate or modulate real-time audio synthesis.

The basic timeline for the project involved in-lab development and refinement of new digital instruments, frequent workshops with the choreographer and dancers to explore movement and evaluate the instruments in-use, and working with the composers to integrate our software mapping tools into their workflow.

5.4.2 Designing Gestural Controllers as Prostheses

One of the possible typologies of digital musical instruments we find useful arranges them from the perspective of the "physical embodiment" of the instrument. Although the very term "instrument" might strongly suggest a physical instantiation, the term takes on additional baggage in the context of musical performance and is commonly stretched to encompass a variety of abstract and non-physical systems or phenomena. As can be seen in Figure 5.1, we consider one extreme of the typology to be inhabited by instruments with a strong physical instantiation – these instruments are identifiable, localizable objects, "played" using direct interaction with the instruments' parts. Moving along our continuum we might place an instrument such as the Theremin, since while it has an identifiable locus it is "played" by interacting with the space *around* this object; the "instrument" (in the metaphorical sense) must be considered to include this amorphous, invisible region. Moving even further along our continuum the perceptual impact of the instrument locus diminishes further (e.g., the Microsoft Kinect) until we arrive at "fully immersive" systems [136] in which no physical instrument can be identified at all.[10]

Another design direction that blurs a system's identity is to merge it with the performer's own body, usually by attaching sensors to the body and using their posture, movement, or other signals to control sound synthesis. Sensor-glove projects are particularly common - see [137] for an overview. In our earlier overview of interactive dance we mentioned systems which merge with the performer's body so completely as to minimize their perceived presence, both by the dancer and the audience.

For this project we decided we were particularly interested in designing instruments

[10] As an aside, we consider there to be another possible extrapolation to the left of the illustrated continuum which addresses systems in which the object itself is the focus or medium of the interaction. *Infra-instruments* (observed on their path to destruction[13]) and *circuit-bending* are examples inhabiting this region – the instrument object in these cases is not merely identifiable but identified and fetishized.

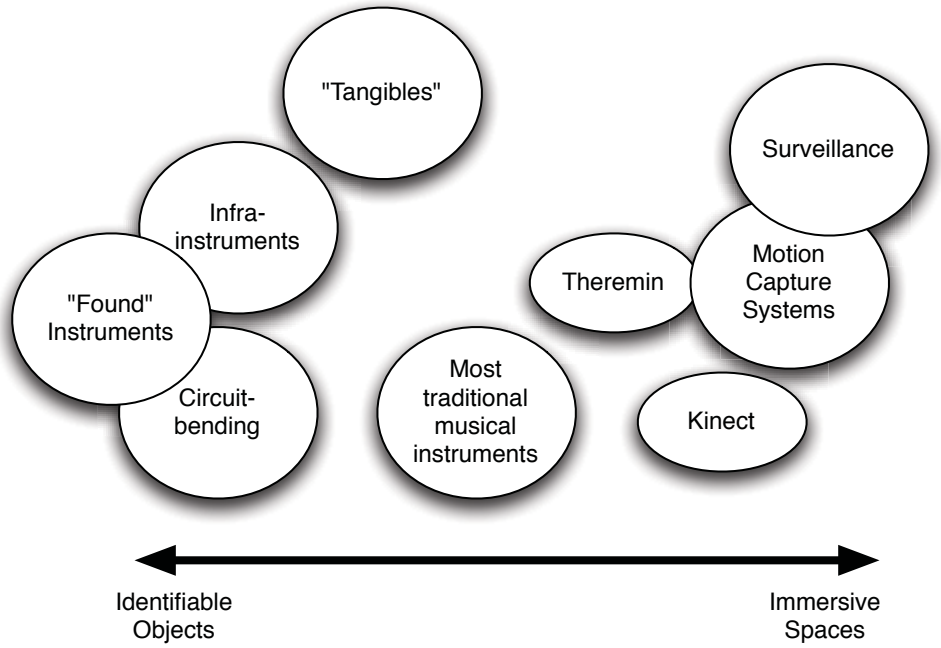

Fig. 5.1 One possible typology of digital musical instruments, organized by their degree of (perceived) physical embodiment.

that could be perceived as *either* objects or part of our performers' bodies, and enable this perception to manipulated or switched smoothly during a performance. The dancers would be able to add and remove our new digital prostheses from their bodies as if they could remove an arm and use it as a separate tool. It is important to realize that this approach is very different from simply adding sensors to the performer's body, since the physical configuration of the prosthesis both adds its own dynamic behaviour to the system (meaning phenomena may be sensed that do not exist on an un-augmented body) and alters the dynamic behaviour of the performer's own body. Certain movements may become more difficult, while others perhaps become easier; the natural balance and resting points of the body may also change. The shape, mass, structure, and even appearance of the prosthetic instruments are not merely decorations, but would be necessarily incorporated into the performance technique and profoundly affect the qualities of any media generated using the sensed data.

5.4.3 Cultural Considerations

In every-day life, prostheses typically correct or improve some supposed *disability*. This potentially adds weighty "cultural baggage" to our new instruments, and needs to be considered carefully when approaching the aesthetic design, e.g. do we want to explore concepts of disability/suffering/awkwardness in our design? In robotics the "uncanny valley" effect – in which the robots which appear "almost" human elicit feelings of fear or revulsion – is well known [138]; we might therefore wish to avoid prostheses that too-closely mimic the appearance of human flesh, or we might wish to deliberately play with these perceptions.

There are also futurist/science-fiction ideals of the "augmented" post/super-human. The connotations here are not related to disability but rather additions or extensions to existing human ability. Cyborgs are often depicted as villains in Western pop-culture, but there are also positive depictions. Technological prostheses in this category are becoming visible in media, e.g. augmented-reality head-up display systems à la Steve Mann [139] are quickly becoming practical for everyday use (e.g. Google Glass [11]).

5.4.4 Users

For the *Prosthetic Instruments*, we were designing for three different levels of users:

The dancers and musicians using the instruments. For the performers we needed to consider the ergonomy of the instruments and its possible effect on fatigue or injury. Other concerns include the degree to which the playing techniques resemble the performers' pre-existing skills, the learning curve [140], and cognitive bandwidth required [141]. The instruments should also make the performers feel that they have scope for expressing themselves. Beyond the familiarity of the physical appearance and dynamics, these concerns bear strongly on the *mapping* design.

The composer and choreographer composing for the instruments could be considered to be "meta-users" of the instruments, since they are tasked with designing aesthetic and performative contexts in which they will be used. Important concerns at this level include programmability vs. specificity, changes (even improvements) brought through design iteration, finding compatible aesthetic goals, and flexibility vs. constraint in both sound production and movement affordances.

[11] http://www.google.com/glass/start/

The characters played/embodied by the dancers. In the context of the piece *Les Gestes* being developed in parallel with the instruments, dramaturgical concerns might also impact the design. Even in abstract, considerations of believability, the perceived skill and effort required to play the instruments, and their aesthetic appearance play a strong role even in non-programmatic performances. In our case, one design goal was to suggest that our new artifacts might have an existing (though foreign to the audience) cultural and practical context and are not merely props or costumes.

In addition, the instruments need to be suitable for two different modes of interaction: as object and as body. For interaction with the instrument-as-object, the design needed to be interesting as an artifact in its own right in terms of appearance and interaction affordances, and not simply suggest that it is waiting to be worn. As prostheses, we are interested in the dynamics of the body-instrument system rather than either on its own. Ergonomics is important in this process but is less important than some other concerns, since the dancers *should* move differently while wearing them. These prostheses are not *costumes*, but rather new limbs which happen to be detachable.

5.4.5 Design Schedule

Our experience in earlier projects has been that while the feedback of the composers and performers during the instrument design was invaluable, not enough time and focus was provided for the composition of the works. For *Les Gestes*, we attempted to remedy this by basing the instrument design upon the T-Stick and allotting time for the creation of the choreography and composition simultaneously with the instrument design.

The *Gestes* project took place over 18 months, followed by several months of rehearsals before the public performances. The research focused on four workshops in which the choreographer, composers, and instrument designers met for two weeks and work would be created drawing upon each of their contributions. Our intention was that this process would enable the three different artistic research processes to influence each other. The first workshop was in August 2011 and the project culminated in public performances in March and April 2013.

As instrument designers the workshops presented hard deadlines by which functional instruments had to be ready. The creation of functional prototypes by the second workshop

was necessary because all of the research conducted by the various collaborators depended upon actively working with the instruments: the choreographer and dancers to develop appropriate choreography and integrate the physical forms into the dancer's body-dynamics; the composers work on mapping the dancers' gestures to sound; and the instrument designers so we could re-evaluate and iterate on aspects of the design affecting movement, appearance, and robustness. The workshops were also an opportunity to address specific design problems by the iteration of prototypes during the two week span of each workshop.

5.5 The First Prototypes

Our first workshop brought together the instrument designers, choreographer, dancers and composers for the first time. Held over a week in August 2011, this workshop was intended to be exploratory in nature, with no expectation of creating functional instruments.

In planning for the workshop, we selected various materials with which we could construct mock-up prototypes of the instrument shapes that would be wearable by the dancers. In this way we could begin exploring the aesthetic effects of augmenting the dancers' bodies, the ergonomic and kinaesthetic issues experienced by the dancers when so augmented, and potential movement and choreographic material. These physical materials consisted of long strips of packing foam with a square cross-section approximately 10cm across, sheets of corrugated plastic, approximately 2m of flexible transparent PVC tubing with a 6cm diameter, and various types of fabric strapping and velcro with which to attach our prototype objects to the dancer's bodies.

We used the foam to construct frames around the dancers' bodies, extensions to the head, tails, and second "spines" mirroring the dancer's own spines. The latter in particular seemed promising, and the "spine" object went through several iterations within the workshop refining its shape and articulating its curvature and behaviour during movement by adding and removing small amounts of material to the form (Figure 5.3).

We also explored many directions with shapes constructed of corrugated plastic, cut, bent and folded into various shapes including spikes, extensions to the shoulders, arms, ribcage, and jaw and head. One particular shape was found to be effective as both a type of 'visor' and as used in groups to define an asymmetric 'cage' extrapolating from the ribcages of the dancers (Figure 5.5).

We gradually converged on a refined set of instrument/objects through a repeated pro-

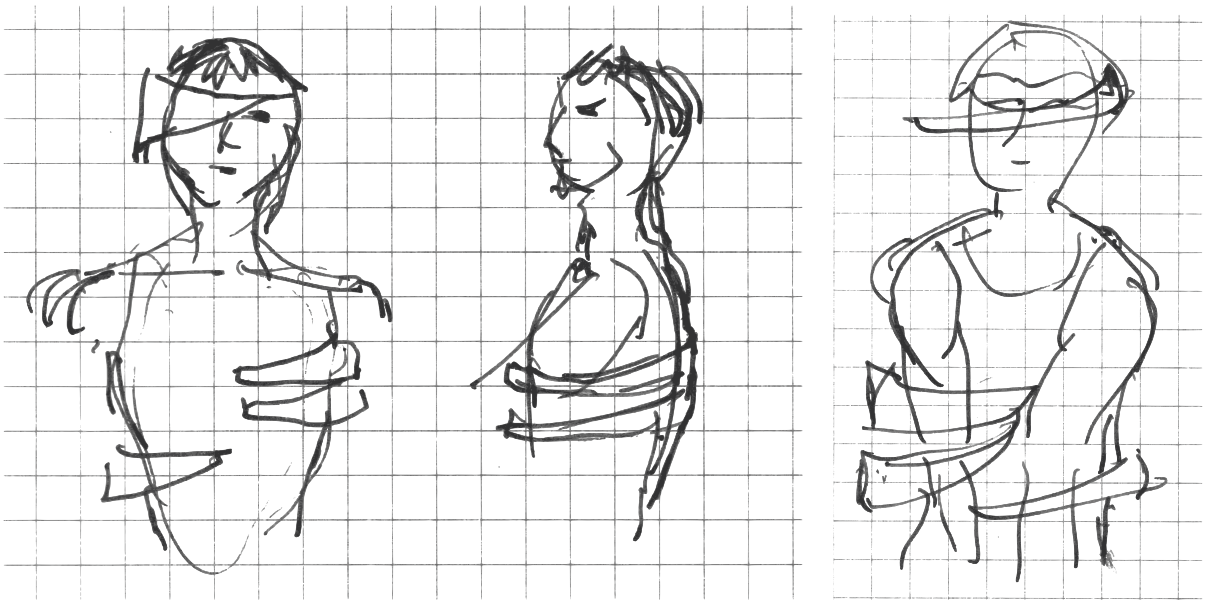

Fig. 5.2 Planning sketches for the "Rib" and "Visor" instruments produced for the first workshop.

cess of sketching, modifying the physical objects, and exploration of movement when the dancers were wearing them. At this point we deliberately kept considerations open regarding the kinds of materials we might use for the final versions, or sensing possibilities for making them interactive. Some discussion of mapping concepts occurred and were documented with the sketches.

5.5.1 Sensing

Even during the initial exploratory workshop, we began researching possible sensing solutions for the foam objects we were creating. Once we were happy with the basic shape and dynamics of the foam and plastic prototypes, work on designing and integrating sensor and wireless communication began in earnest. Our goal was to have fully-functional instruments as early in the project as possible, so that our collaborators could begin exploration of mapping configurations in tandem with exploration of choreographic and musical material. Meanwhile we would have the freedom to explore design, materials and construction approaches for the next iteration. Essentially our requirements for sensing were:

- Low-latency to enable responsive control.

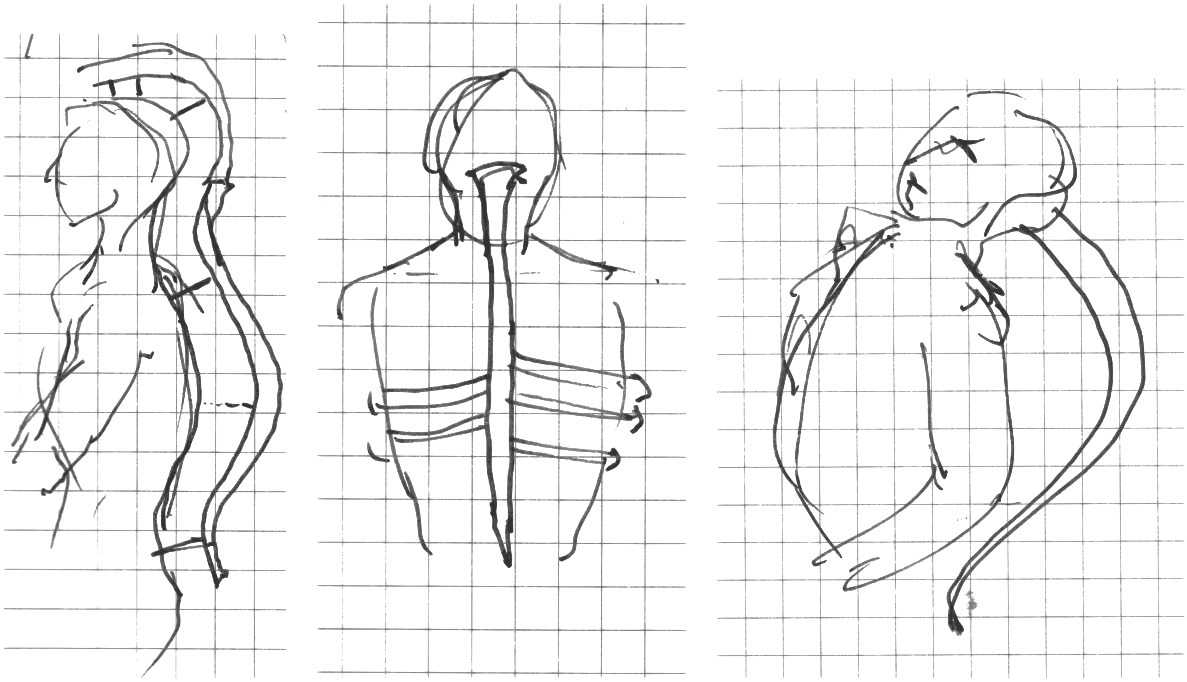

Fig. 5.3 Planning sketches for the "Spine" instrument produced for the first workshop.

- High resolution sensing of phenomena to enable nuanced control.
- The solution must be very robust, both to physically survive the rigours of live dance performance and resistant to environmental interference (e.g., high levels or ambient light (visible and infra red) from stage lights).
- Not visually distracting, although we were open to incorporating the sensors into the instrument's visual aesthetic.
- Not dependent on the instrument having a specific stiffness or be constructed of a particular material, so that we would not have to re-solve sensing problems as the physical design evolved.
- Allow the dancers completely free movement without occlusion, therefore wireless and relatively light-weight.
- Able to operate with constrained electrical power (<100mA) to avoid concerns with large batteries or frequent recharging.
- Reasonably inexpensive.

For the Spine, we decided that we would like to track the orientation and deformation as it moves with the dancer. This choice fit well with our interest in conceptually blurring instrument-as-object and instrument-as-body; the Spine prototypes follow the dancers' bodies closely but also stand out with their own tension and dynamics. Since at this stage in the design we had not committed to any particular materials, it was important that we choose a sensing approach that would allow us maximal flexibility in future iterations.

Based on these criteria, we opted to combine data from accelerometers, rate-gyroscopes and magnetic-field sensors to independently estimate the orientation of the two ends of the Spine[12]. By noting the difference between the two ends, we can also estimate bending and twisting of the entire structure. Including a wireless radio transceiver for communication, this solution cost approximately CAN$250 and provides updates of the orientation as quaternions at approximately 200Hz. We calculate the orientations using a complementary

[12]This arrangement of sensors is sometimes referred to collectively as an inertial measurement unit, or IMU.

filter running on-board the Spines to provide a high sample-rate and tight timing for integration of the gyroscope data. This solution was embedded in existing foam prototypes in time for the second workshop (Figure 5.4).

Fig. 5.4 A foam spine prototype with embedded sensing of orientation and shape in use during a public post-workshop presentation. Dancer: Sophie Bréton.

We also experimented with different spatial resolutions for the orientation sensing, since this determines the resolution of measurable deformations. With three equidistant sets of sensors we can sense simple compound "S" curves, for example, which cannot be distinguished using only two. Future versions of the Spine DMI may use higher-resolution sensing, as the price and availability of IMUs are improving rapidly. The spines used for the public performances of the Gestes project used only two IMUs located near the mounting locations behind the head and at the bottom of the back.

For the Ribs and Visors, since the objects present interesting, wide curved surfaces for touching and brushing we decided to use capacitive touch sensing building on technology already developed for the T-Stick DMI. The corrugated plastic material used for initial prototyping was not suitable for mounting sensors: it would wear out fairly quickly and did not hold a particular shape very well once bent or curved. Since we planned to inte-

grate lighting into the instruments, we began testing capacitive sensing using a variety of transparent conductive materials (ITO, graphene) as well as various conductive paints and tapes that would allow us to draw or print sense electrodes onto the ribs and visors. In parallel, we began vetting different plastics for use as the basic construction material.

We decided to use the MiniBee micro-controller boards for sampling sensors and wireless communication with the composers' computers. We developed these boards for an earlier project [44] as low-cost, Arduino-compatible circuit boards with the footprint for a ZigBee wireless transceiver [142]. We considered ZigBee to be an appropriate choice for this project, since it provides low-power, sufficient range for our needs (around 100m line-of-sight), support for many simultaneous nodes, and doesn't require any intervention to reconnect after a node has been power-cycled or has lost signal. As another known benefit, the MiniBee nodes include a three-axis accelerometer, meaning that all the ribs and visors could also output some orientation and movement data.

5.6 Refinement

5.6.1 The Spines

Our third prototype of the Spine instrument began our exploration of different materials and segmented structures. Keeping the roughly square cross-section used in the foam versions, we constructed it from cut segments of thin transparent rectangular PVC tubing connected with hinges made from the same plastic. Support for the overall structure was provided by rubber bands connecting adjacent sections. This structure was a vast improvement in appearance over the foam and was used in early press photographs for the project. Mechanically it proved problematic for several reasons. First, it's construction was very intricate, which not only made it difficult to build but also increased the chances of mechanical failure. Second, the hinged design did not support side-to-side or twisting motions. In addition, the use of distinct hinged parts pushed the aesthetic firmly into the realm of the mechanical, more *robot* than *cyborg*.

The fourth prototype was again designed as a segmented structure, but with the separate segments bonded to a single flexible plastic substrate which ran the length of the object. This greatly simplified and lightened the object structure and restored the natural curvature from the foam prototypes, although it was much less flexible when twisted. We considered

this approach promising and visually much more appealing, however a great deal of strain is placed on the flexible backing and we had difficulty resolving the conflicting demands of materials with appropriate flexibility, strength, and density.

Moving to Vertebrae

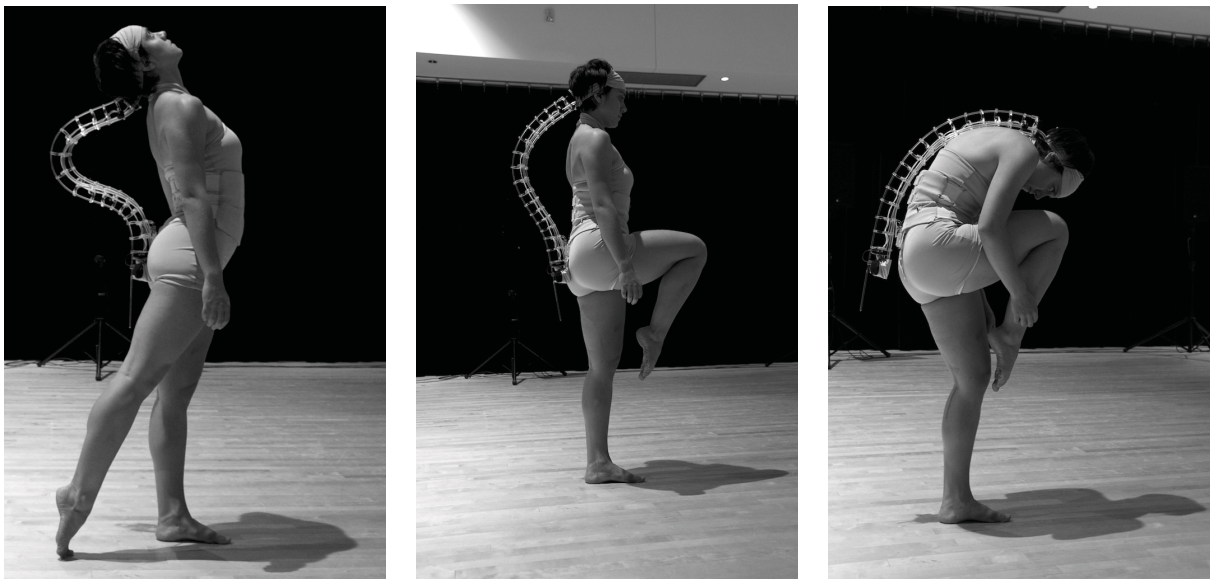

Fig. 5.5 The prototype spine in use by dancer Sophie Bréton. (Photograph: Vanessa Yaremchuk, used with permission)

Starting with the fifth prototype design we started using three-dimensional structures to resolve our strength-vs.-weight problems without needing a miracle material to satisfy all our demands. In this version, separate "vertebrae" were fashioned using 0.25" thick laser-cut acrylic and threaded onto flexible PVC hoses forming a truss-like structure with a triangular cross-section. In order to allow the spine to bend, the third rail in the structure is replaced with a very flexible PET-G rod; while the holes cut in the acrylic segments were sized for interference-fitting with the PVC hose, the PET-G rod is considerably narrower and able to easily slide through. Initially, the PET-G rod was fixed only at the head, with a length extending past the length of the PVC. With refinement, the arrangement of two fixed rails and one sliding rail provided the object bending affordances remarkably similar to the foam prototypes, while also providing greatly increased strength/robustness since

strains are naturally distributed (figure 5.5).

We proceeded to design and construct prototypes 6-8 using the building blocks from prototype 5, while exploring alternate constructions including helical rails and heat-modified vertebrae, scale tests using slightly different materials, and optimizing the vertebra designs for increased strength. Final explorations involved increasing the length of the Spine past the lower mount and altering the dynamics of the instrument when bent or twisted by fixing the PET-G rod at both top and bottom, meaning that both top and bottom rails had fixed lengths. This modification prevented the Spine from forming simple curves, instead forcing it into interesting compound-curve shapes when bent. The total length of the PET-G rod sets the natural resting curve of the Spine.

The friction fit of the tubing proved reliable for most of the length of the Spine. However, we encountered several problems at the head of the spine in which the friction wasn't sufficient to counteract the weight of the Spine and the tension created when bending and twisting. We considered several methods of securing the PVC tubing in the top vertebrae involving pins and collars; however, we were wary of drilling holes through either the vertebrae or tubing due to fears it would compromise their durability. Our solution was to use superglue between the PVC and the holes in the top vertebrae, which proved to work reliably but didn't create such an unbreakable bond that we couldn't remove the tubing if we needed to make adjustments.

5.6.2 The Ribs

In the move towards functional prototypes, the decision was made to utilize rigid 1/8" thick acrylic panels as the basic material. Our material requirements were:

- The instruments needed to be durable enough for an extended run of performances. Flexible paper and plastic materials develop wear quickly and are easily damaged.
- The cantilevered design of the Ribs necessitates that they be both lightweight and rigid, as the entire weight of the instrument needs to be supported on one end.
- The material needed to be able to keep its form; even with the paper shapes we identified the need for specific curvatures based upon the location of the instruments on the body. We also decided that a rigid shape would place less wear on the embedded electronics.

- The machinability of the material needed to be within our capability. Being primarily an electronics and HCI lab, our workshops contain only basic tools; any additional machining needed to be within the capability of tools we could acquire or gain access to through university resources or outsourcing to private manufacturing facilities.
- The material needed to be cost-effective to stay within our relatively limited research budget.

Clear acrylic plastic had many properties which suggested its use to us. Its basic material, being transparent, does not have a strong visual identity, causing it to be less immediately recognizable than wood, aluminum, or copper. Its thermoplastic properties permit heat-forming to different shapes at a temperature which is easily attainable in a non-specialized lab. It is machinable with hand tools as well as with general purpose laser cutters without creating toxic gases [13]. It is lightweight and relatively impact-resistant - more than glass, for example, although less impact-resistant than polycarbonate.

First Functional Prototypes

The first round of functional prototypes were hand-formed from acrylic panels to roughly the same dimensions as their paper predecessors. Their curvature was shaped by hand using a standard heat-gun. Capacitive touch-sensing was implemented with pads and traces made out of copper tape.

These initial plastic prototypes were found to have several aesthetic issues. The most obvious was that they were too small, and the curvature too tight to the body. This prevented them from having a satisfying visual presence, and also influenced the ways in which the dancers interacted with them – larger shapes being seen as giving more room, inside the Ribs and outside, for the dancers to move their hands and arms. The large copper pads for touch sensing were also seen as problematic. While we had no problem with electronic components forming part of the visual presence of the instruments, the initial layout of copper pads and traces were both functionally problematic and aesthetically problematic. Our primary aesthetic concern was the they were too obviously hand-formed, but our research led us to the conclusion that digitally fabricating patterns in metal foils would be prohibitively difficult.

[13] Laser cutting many plastics releases toxic fumes, particularly chlorine. As a result, many laser cutting facilities, including the McGill architecture faculty, won't laser cut these materials

Iterative Design

From the course of the second workshop until June 2012, the aesthetics of these rib-shapes were changed through a process of iteration. During this period we gained access to the laser cutter at the McGill faculty of architecture in the spring and used it for the creation of a wide variety of rib shapes. We explored a succession of different approaches to fabricating copper shapes for pads before settling on a transparent conductive plastic material. The traces connecting these pads went from copper strips to ribbon cable to magnet wire.

The basic visual aesthetics for the Ribs came to be laser cut 1/8" acrylic panels with grooves etched for the magnet wire, with transparent conductive plastic for touch sensing. When we were using copper strips for the traces we were happy that their layout should be straight, orthogonal lines – visually evoking a circuit board pattern. Once we moved to a transparent material for both the basic shape and the touch pads, we decided that a more organic, curved shape to both the rib outline and the magnet wire traces was more desirable.

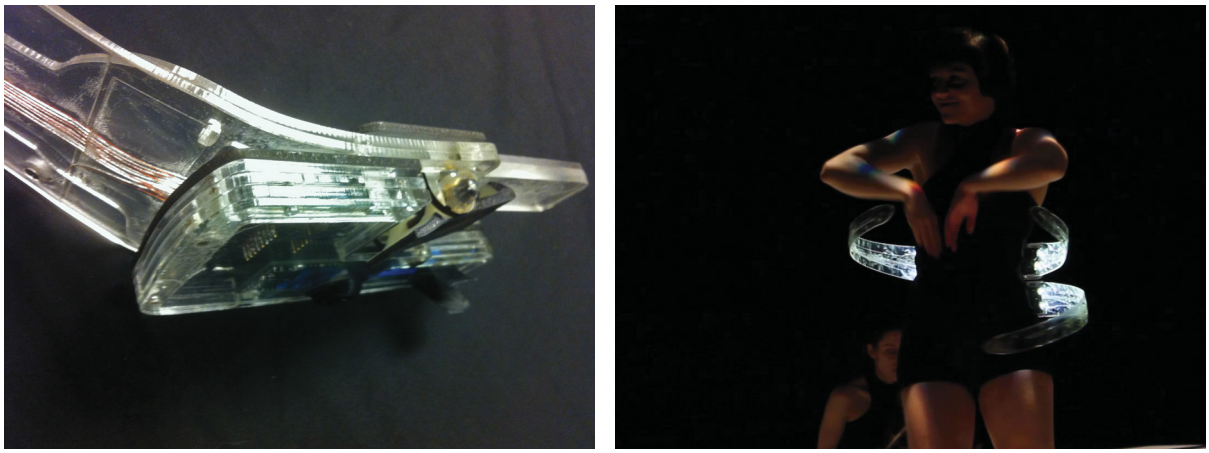

Fig. 5.6 The final Rib designs showing their laminated composition and final shapes and configuration. (Dancer: Sophie Bréton, Photographs: Ian Hattwick & Audréane Beaucage, used with permission)

The Final Rib Prototypes

Once the Rib design became fully transparent it was necessary to continue making their overall shape larger in order to maintain a strong visual impact. As they grew larger they

began to suffer from structural issues due to their cantilevered design. In order to resolve these issues we moved to a multi-layer laminated structure that both added structural rigidity and strength and also incorporated a protective enclosure for the electronics.

The final design called for three differently sized Ribs with a specific shapes and curvatures in order to complement the dancers' bodies, as seen in figure 5.6. Many iterations were created in order to find the correct curvatures and ultimately jigs were designed to allow for their accurate reproduction.

5.6.3 Visor

Fig. 5.7 Soula Trougakos wearing the Visor in a performance of *Les Gestes*. (Photograph by Audreane Beaucage, used with permission.

These Visor shapes are based closely on the functionality of the ribs, utilizing the same

materials both for the physical and electronic construction. The primary differences are their shape and structure and the way in which they are mounted.

Because the visors remained close to the original size of the ribs, and were also subjected to less interaction by the dancers' hands and arms, we did not find it necessary to add the secondary support layer that the ribs required. However, due to differences in the shape of the dancers' heads we found it necessary to give each visor a unique shape. This is in contrast to the Ribs, for which the same shapes were found to work for both dancers.

Our original conception was for the visor to use the same mount as the top of the spine, directly behind the dancer's head. However, as the visor extends in front of and around the face, mounting only behind the head makes the visor extremely cantilevered. This didn't create a stable connection and allowed the visor quite a bit of movement separate from the movement of the dancer's head. The addition of a second mounting point near the dancer's left ear stabilized the visor a great deal, especially since most of the mass of the visor is contained in the electronic enclosure situated between the two mounting points.

5.7 Discussion

5.7.1 Discussion I: Fast Iteration using Digital Fabrication Techniques

Over the years we have designed and constructed many DMIs for exploring the potential of different shapes, materials, sensors, and control concepts. Since these new instruments are not intended for commercial sale, they are usually produced in extremely small numbers (typically only one or two). The T-Stick DMI is the most extreme exception from our work, with the production of around 20 instruments over the course of five years. At these small scales most construction and assembly work is done by hand, since it is usually not economical to use industrial processes for small runs.

The advent of affordable digital fabrication technologies is quickly changing this balance, since they bring many of the benefits of industrial production – notably precise control over output – without requiring commitment to manufacturing thousands of units. Since the parts are manufactured directly from digital models, making a small change to a part is much simpler than rebuilding moulds or jigs. Additive manufacturing is still quite expensive, however we estimate that its use saved us both money and time; it is doubtful whether we could have developed and improved the design of our mounting hardware as

Fig. 5.8 Marjolaine Lambert and Sophie Breton playing the Ribs during a rehearsal for the piece *Les Gestes*. Photo by Michael Slobodian, used with permission.

quickly without access to the 3D printer.

In addition to producing more consistent parts and saving us time, the use of laser-cutters for producing acrylic parts also enabled a rapid solution to structural issues in the design of the Ribs. When it became clear that the longer Ribs required additional rigidity with a minimal increase in cantilevered weight, it was relatively easy to modify the CAD model to create a narrow support layer which follows the Ribs' shape. Similar alterations were used in the creation of the laminated electronics enclosure for the Ribs, which includes a interlocking magnetic closure allowing the top to be removed for installation and repairs.

Scaling to Production

As production deadlines approached and the number of instruments required for the overseas tour of *Les Gestes* increased, it became clear that we had to optimize the instrument production as much as possible. To satisfy the (perhaps justified) paranoia surrounding productions depending on new untested technology, we committed to manufacturing a full set of back-up instruments in addition to those scheduled for use in the actual performance. To make production faster, we outsourced the laser-cutting orders to an outside company instead of doing it ourselves, and recruited a number of students from the lab as assistants for soldering and some assembly.

Once the acrylic vertebra segments have been laser-cut, the Spines are relatively easy to assemble. Threading the vertebrae onto the PVC rails is simply a process of stretching the PVC to reduce its diameter and adjusting the locations of the acrylic segments. The interference fitting between PVC and acrylic is sufficient to hold in place even during the rigours of performance, but still relatively easy to adjust when desired. We also outsourced the IMU fabrication, using two of the ckDevices Mongoose IMUs per spine. Some effort was still required wiring the IMUs and fitting them in project boxes, preparing another box for the wireless transceiver, loading firmware and testing.

Even with the fabrication of all of the Visor and Ribs' components outsourced, they are still much more time-consuming to produce. The specificity of the curved shapes, intricate laminated structure with twelve interlocking parts, and custom capacitive sensing all demand significant labor to reproduce. Due to these requirements, all shaping and assembling of the acrylic parts as well as the electronic design and assembly was all done in the lab. The careful refinement of the CAD models and electronics, as well as fabrication

aids such as bending jigs, helped to streamline the process, but the successful manufacturing of the final 16 Ribs and Visors still took a considerable effort.

5.7.2 Discussion II: Integration with Mapping Tools

After designing and constructing the physical instruments, adding sensors and applying signal processing to extract interesting gestural signals, there is still a crucial link missing: we still need to design the "aesthetic" mapping layer which links gesture or movement to specific control of media synthesis or processing. Our experience on past projects has made it clear that this design process is very difficult and time-consuming, especially in a collaborative setting. The process of mapping also necessarily involves the establishment of at least a minimal gestural vocabulary or style of interacting with the instrument in order to supply some performance data to be mapped; an ideal approach might involve workshop experimentation with a variety of mapping configurations, choreographic material, and media synthesis tools. In practice, we have found this kind of scenario often results in frustration while performers wait for technical changes, and also severely limits the amount of mapping experimentation that can realistically take place in the workshop context[14].

Starting during the *McGill Digital Orchestra Project*[104], we began building a software infrastructure for aiding in the mapping design process. Our tools are built on a central software library – called *libmapper* – which supports a "plug-and-play" approach to mapping, in which various devices such as physical controllers, synthesizers, etc. represent their inputs and outputs on a network, and connections can be freely created between them (cf. Chapter 2 in this thesis). The software takes care of all necessary translation between endpoints, meaning connections are fast to implement, but can also be customized to calibrate to a particular gesture or to apply some data transformation to the signals. Crucially, users interact with a *semantic abstraction* of the network: all entities use descriptive names and connections are created by simply drawing lines between them on one of the graphical interfaces. Knowledge of networking protocols, computer IP addresses and ports, and even which computer an interface is connected to, are not required.

The use of libmapper and some related tools allowed us to focus fully on the instru-

[14]It is important to note that recording data from the performers for use in off-line mapping experimentation is of limited use, since we are attempting to create new performable musical instruments rather than "sonifying movement", and that the performers will presumably move very differently when using different mappings.

ments themselves without concern for connection- or protocol-compatibility with the media synthesis side of things. This division of work occurs on a technical level but also a conceptual level, which avoids confusion and discourages premature assumptions about the eventual mapping. The software design approach was to simply expose *all* interesting signals describing the state or dynamics of the instruments, whether or not we thought at the time that they might be interesting for a particular mapping or piece. At the time of performance, the device drivers for the Gestes instruments make available for mapping 25 output parameters for each Spine (Table 5.1) and 9 output parameters for each Rib or Visor (Table 5.2), leading to more than 100 parameters available simultaneously for the Gestes workshops and performances.

Table 5.1 Output Parameters for each Spine DMI

Parameter	Type	Length	Minimum	Maximum	Units
For each IMU:					
.../accel/x	float	1	-2	2	g
.../accel/y	float	1	-2	2	g
.../accel/z	float	1	-2	2	g
.../rot/x	float	1	$-\pi/2$	$\pi/2$	radians/second
.../rot/y	float	1	$-\pi/2$	$\pi/2$	radians/second
.../rot/z	float	1	$-\pi/2$	$\pi/2$	radians/second
.../orientation	float	4	-1	1	unit quaternion
.../tilt	float	1	$-\pi/2$	$\pi/2$	radians
.../roll	float	1	$-\pi$	π	radians
.../azimuth	float	1	$-\pi$	π	radians
.../motion	float	1	0	5	na
For the instrument:					
.../bend/x	float	1	$-\pi$	π	radians
.../bend/y	float	1	$-\pi$	π	radians
.../twist	float	1	$-\pi$	π	radians

During our collaborative workshops, the use of these tools also allowed us to easily add new signals describing the instruments in real-time, even while the dancers were wearing the instruments and the composers were using data from them. As each new signal was added to our device driver software, its description would simply appear as another possible source in the mapping interfaces.

Our colleague Marlon Schumacher undertook to build a “mapper” bridge for the soft-

Table 5.2 Output Parameters for each Rib and Visor DMI

Parameter	Type	Length	Minimum	Maximum	Units
/accel/x	float	1	-2	2	g
/accel/y	float	1	-2	2	g
/accel/z	float	1	-2	2	g
/tilt	float	1	$-\pi$	π	radians
/roll	float	1	$-\pi/2$	$\pi/2$	radians
/motion	float	1	0	50	na
/touch	float	1	0	1	normalized
/brush	float	1	0	5	na
/coverage	float	1	0	1	normalized

ware framework CLEF [15] so that its various synthesis and processing modules would be declared as available destinations for mapping connections. In addition, he integrated mapping management functionality (normally embedded in a GUI) into CLEF, so that mapping configurations could be saved and reloaded directly using the same mechanisms and interface used to load presets for his sound synthesis and processing system. The standard mapping GUI was still used for the initial design and modification of mappings during workshops, but during rehearsals and performances it was used only to monitor the changes instigated by CLEF.

The composers involved in the project decided that they wished to use *dynamic mapping* for their piece, in which the mapping connections change over time. As various scenes are loaded in CLEF, the resources being used for synthesis might change, for example one scene might involve granular reprocessing of incoming audio from the two string instrumentalists, while another spatializes prerecorded sound or produces sound using a different synthesis model. As proponents of the mapping-as-instrument paradigm (rather than mapping-as-piece) [30], we believe that dynamic mappings are generally confusing for the performer, and make it difficult for them to acquire expertise with a new interface. In addition, we have found that sometimes the description of a dynamic mapping scheme simply adds unnecessary temporal organization to an underlying static scheme: often the separate mappings do not even address the same gestural material, and could be folded

[15] The CIRMMT Live Electronics Framework, a modular system built in Max/MSP for composition and performance of music with both live and electronic components, developed at McGill University since 2009 [143].

into a single static mapping scheme which affords more control to the performer while also remaining consistent. Within the challenging context of live workshops, however, mapping schemes tend to be designed as "vignettes" that are appropriate for the movement and media material being explored, and are not arranged in any systemic fashion. This is a natural and necessary artifact of the workshop process, and using these mapping vignettes serially in a piece is the obvious outcome.

The Gestes project provided us with further proof of the utility of our concepts and tools for aiding collaborative mapping design: we have vastly simplified and streamlined the process of experimenting with mapping configurations. Moving forward, we are giving thought to how we can help organize and hopefully combine various mapping sketches or vignettes into more complex, consistent behaviours and qualities for our instruments. We believe that placing control in the hands of the performers is the entire point of gesture-controlled media systems, and that the "acousmatic+" paradigm – in which the composer/system has all the power but a performer is used to inject "liveness" – does a disservice to both performer and audience.

5.8 Conclusion

We have presented a new family of prosthetic digital musical instruments, developed over three years in an intensive collaboration with dancers, instrumental musicians, composers and a choreographer. Rough prototypes of the instruments were developed in early workshops where they could be mounted to the dancers' bodies and evaluated in parallel with exploration of related gestural vocabulary. The designs were subsequently developed through rapid iteration, aided when possible by the use of digital fabrication technologies which allowed us to consistently reproduce small desired variations and refinements to the previous generation. This approach aided us further when we needed to scale-up production of instrument copies and production deadlines approached, since we could easily outsource parts of the production process.

Finally, the instruments were used in live performance of the contemporary dance/music piece *Les Gestes*, which toured parts of Canada and Europe during the spring of 2013. Our approach to designing robust instruments proved sufficient, since no instruments were broken during the tour (which travelled with backups) and the technology worked seamlessly as part of the artwork.

5.9 Acknowledgments

This research was supported by the Programme d'appui la recherche-création from the Fonds de recherche sur la société et la culture (FQRSC) of the Quebec government, by the Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT), and the Natural Sciences and Engineering Research Council of Canada (NSERC). We would also like to acknowledge our collaborators from the Gestes project: composers Sean Ferguson and Marlon Schumacher, choreographer Isabelle Van Grimde, costume designer Pascale Bassani, dancers Soula Trougakos and Sophie Bréton, and musicians Marjolaine Lambert and Elinor Frey.

Part III

Conclusion

Chapter 6

Conclusions and Future Work

In this dissertation we have described the design of a system for supporting configurable inter-connections between the parts of digital musical instruments (e.g., control interfaces and digital audio synthesizers) or other interactive systems. This system is conceived as a distributed, zero-configuration network of devices with no central server, with mapping connections transported as peer-to-peer data streams. All such streams are labelled with strong semantics, and wherever possible carry metadata representing the real ranges and units of the phenomenon being sensed or the parameter to be controlled. Rather than enforcing data normalization or other system-representation standards, our system provides inter-device compatibility by automatically translating datatypes and ranges as necessary. Each connection also allows arbitrary signal processing to allow transformation of the data stream as conceived by the mapping designer. This system has been implemented as an open-source, cross-platform software library called *libmapper* written in the C programming language and accompanied by language bindings for a number of popular languages.

We have also presented work on supporting mapping connections between data streams representing properties of entities that might have multiple copies or *instances*. Such systems include multi-touch interfaces, in which instances of "touch" may exist in varying numbers over time, and polyphonic synthesizers, which might play varying numbers of "sounds" or "voices" simultaneously. The concept of instances can also be applied to temporally-segmented performer gestures, in which the "end" of a performed gesture signals the destruction of an instance of that gesture. A generalization and formalization of the multi-instance problem was presented, followed by details of the extensions to libmapper

for supporting this feature.

Finally, we have presented two case studies for the use of our mapping software, both of which culminated in successful public performances in multiple countries. The first of these are the T-Stick digital musical instruments, a family of tube-shaped gestural controllers featuring sensors for pressure, multitouch, deformation, orientation and movement. The hardware and firmware for the T-Stick has gone through three major design iterations over a period of eight years, and has been adapted for use by dancers and for controlling spatialization of sound within a concert venue. Mapping for the T-Stick is designed and implemented exclusively with the mapping tools mentioned previously, in fact our needs for the T-Stick were a driving force for both development of new mapping features and improvements to the software dependability and ease of use.

The second case study concerns the more recent development of *prosthetic* musical instruments for use by dancers and musicians, which were developed in a series of rapid iterations over the past three years. Unlike the T-Stick, mapping design for the prosthetic instruments – the "Spine", the "Rib", and the "Visor" – was carried out primarily in the context of intensive collaborative workshops involving dancers, a choreographer, and composers tasked with both musical composition and mapping design. The mapping tools were invaluable in these workshops, in that they allowed new mapping ideas to be quickly executed, modified, and archived without requiring reprogramming of the instrument drivers. As new versions of the instruments were completed, they were automatically integrated into the workflow as sources of data for controlling audio synthesis or musical processes.

6.1 Contributions

The main contribution of this work to the research community – and to industry – is to provide compatibility and "mappability" between the disparate parts of interactive systems without compromising on design flexibility or imposing any particular workflow or design process. This should allow designers of digital musical instruments and other interactive systems to freely experiment with collaborative interconnections between their creations, while encouraging them to use whatever tools are appropriate for the job or are personally preferable.

To our knowledge, the flexible support for *multi-instance* mapping described in Chapter 3 is a feature not present in other mapping systems. It will enable new types of control

interfaces and synthesis systems to be interconnected with the existing ecosystem of devices and services.

6.1.1 Impact of libmapper

As a testament to the usability of this work by a larger community, the concepts and software being produced are already being used in a variety of large and smaller projects, spanning laboratory work, department-wide and inter-university collaborations (the McGill Digital Orchestra, Sense/Stage, E[MERGE], and Gestes Projects), many on-stage performances (e.g., with the T-Stick digital musical instrument (see chapter 4), and use by external companies and institutions including Moment Factory, Concordia University, the University of Lethbridge and Janro Imaging Lab.

Although not explicitly presented here, support of the mapping tools also includes the creation and maintenance of library documentation, mailing lists for developers, and a website[1] with background information, tutorials, and links for downloading the software.

6.1.2 DMI Case-Studies

The relative success of the new DMIs presented in Part II indicates that our design decisions have merit. For the T-Stick, these included prioritizing robustness of the instrument above all other concerns, as well as hiding the sensors from view to discourage perception of the instrument as a collection of disparate sensors and encourage integral, holistic interaction with the instrument.

The development of the Spine instrument yielded several contributions: an approach to instrument construction that is lightweight, affordable, attractive, flexible and yet very strong; a novel approach to sensing the shape of a deformable object using inexpensive inertial and magnetic sensors; open-source firmware for estimating orientation by combining the IMU sensor signal; and software tools for manipulating quaternions in Max/MSP/Jitter.

Since both the T-Stick and the prostheses are intended for performance of music, we must also claim as related results the compositions and performances using the instruments.

[1] http://libmapper.org

6.2 Future Work

The software library and surrounding tools are completely functional, and are actively used for mapping and performance, however the project is by no means over and we have very ambitious goals for features and language compatibilities. Our detailed development roadmap includes a number of planned improvements and new features, some of our top priorities are:

Improved installers, integration with package managers – in order to support non-technical users, it is especially important that our tools are easy to install and run. We plan to submit packages to popular package management utilities for various operating systems, and for programming environments that include their own library management tools (e.g. Processing, Max/MSP).

Alternate connection transport – network-based tools for musical performance typically use UDP for networking since they prioritize low latency messaging over guaranteed delivery. For certain mapping scenarios, some connections might be better transported over TCP or another protocol - we are implementing a choice of protocols as a configurable connection property.

More flexible vector support – currently mapping of signals with vector datatypes is not well supported, since our expression syntax engine does not support specification of vector indexes. One of our top priorities is a revision to add this support.

Embedded devices – another avenue we are actively exploring is running our tools on microprocessors embedded in the instrument hardware instead of using dedicated driver software. We are working with adaptations of the Firmata protocol/firmware[2] for the Arduino micro-controller platform[3] as well as implementations for more powerful embedded computers.

We also plan future improvements to the DMIs discussed in this dissertation. As stated in chapter 4, we have been working towards a fourth version of the T-Stick hardware, making improvements to the resolution of touch sensing. This version is also planned

[2] http://firmata.org/wiki/Main_Page
[3] http://www.arduino.cc/

to include a wireless transceiver and run an embedded version of libmapper rather than requiring a separate driver running on a PC. A similar solution may be adapted for the Spine instruments discussed in chapter 5, along with a planned increase in the number of IMUs used for sensing orientation and shape. In planning the instrument design, we anticipated the near-future availability of affordable, IC-scale IMUs designed for use in the mobile-phone industry. Unfortunately the sensors did not become available before the final instruments were constructed for the performances of *Les Gestes*, restricting our solution to more expensive, power-hungry versions, but the new sensors will be integrated into future versions of the Spine instrument.

Part IV

Appendices

Appendix A

The T-Stick DMI: Public Appearances

Date	Type	Event	Location
2006/04	concert	seminar concert	Montreal, Canada
2006/11	recital	Fernando Rocha lecture recital	Montreal, Canada
2007/06	presentation	New Interfaces for Musical Expression	New York, USA
2007/09	demonstrations	Wired Nexfest	Los Angeles, USA
2008/02	demonstrations	Innovaction	Udine, Italy
2008/03	concert	MusiMars Festival	Montreal, Canada
2008/04	concert	Music+Technology Incubator	Montreal, Canada
2008/07	concert, workshop	Sound Symposium	St. John's, Canada
2008/09	concert	Le Vivier relaunch	Montreal, Canada
2008/10	concerts	Duo pour un violoncelle et un danseur	Montreal, Canada
2009/02	competition	Guthman competition	Atlanta, USA
2009/07	workshop	SMC Summer School, used T-Stick to control robotic gamelan	Porto, Portugal
2009/08	concert	International Computer Music Conference	Montreal, Canada

2009/10	workshop	Society for music theory, workshop on listening through time	Montreal, Canada
2009/12	concert	live@CIRMMT	Montreal, Canada
2010/04	concert	CHI conference	Atlanta, USA
2010/04	concert	Viagem (5 T-Sticks played by blind performers)	Porto, Portugal
2010/07		launch of 2010 T-Stick Composition Workshops	Montreal, Canada
2010/08	talk, performance	Expansive Spirits, Toronto Electroacoustic Symposium	Toronto, Canada
2010/12	talk	Electronic Music Foundation	New York
2011/02	concert	Ghost in the Machine conference	Montreal, Canada
2011/03	concert	Concerto for T-Stick and two laptop orchestras, Concordia Laptop Orchestra and McMaster Cybernetic Orchestra	Montreal, Canada
2011/04	concert	Open Ears festival	Kitchener, Canada
2011/05	concert	Issue project room	New York, USA
2011/05	concert	New Interfaces for Musical Expression	Oslo, Norway
2011/06	concerts	Universidade Federal de Minas Gerais	Belo Horizonte, Brazil
2011/06	concert	Sforzando Late Night Concert, Electroacoustic Music Studies Network	New York, USA
2011/11	concert	Sea of Sound festival	Edmonton, Canada
2012/03	concert	Experimental music from Brazil and beyond	Lethbridge, Canada
2012/06	concert	Concierto: Andrew Stewart y Ensamble 3	Mexico
2012/09	concert	Moving Sound, Electroacoustic works by D. Andrew Stewart (t-stick) and Rolf Boon	Lethbridge, Canada

2012/11	concert	24 Frames, Improvisation with Tim Brady (electric guitar) during Brady's cross-Canada tour	Lethbridge, Canada
2013/03	concert	From Up There and Down, New Works Calgary	Calgary, Canada
2013/05	concert	New Interfaces for Musical Expression	Daejeon, Korea Republic
2013/09	concert	Percussive Arts Society International Convention	Indianapolis, USA

Appendix B

The Digital Orchestra Toolbox for MaxMSP

The Digital Orchestra Toolbox is a collection of Max/MSP abstractions — modular functions instantiable as objects — that we have found useful in creating gesture processing patches for digital musical instruments. The toolbox currently contains more than 100 abstractions, contributed by Joseph Malloch, Stephen Sinclair, and Marlon Schumacher. Each patch is accompanied by a help patch to demonstrate its use.

Collision Detection	
`dot.alloc`	Using a shared bus, deliberate to allocate a unique identifier.
`dot.alloc2`	Using a shared bus, deliberate to allocate a unique identifier. Peer with root index suggests identifiers to newcomers.

Control	
`dot.for`	Outputs a sequence of incremented numbers when banged.
`dot.line`	Wrapper for the line object so it accepts trajectory lists like `line~`.
`dot.probgate`	A probabilistic gate with remote control.

dot.repeat	Repeats a message a specified number of times.
dot.route~	Separates signals from max-messages.
dot.swap	Like the swap object, but for symbols and lists too!

Data	
dot.atoi	Convert a symbol that starts with a decimal number to an integer.
dot.bytetobits	Converts a single decimal byte into 8 binary bits.
dot.bitstobyte	Converts 8 bits into a single decimal byte.
dot.filein	Adds dump command to the filein object.
dot.getindex	Retrieves indexes of coll data entries that match the query.
dot.index	Generates the lowest unused index (for coll storage). Argument sets maximum index.
dot.playabsolute	Play files recorded by dot.recordabsolute.
dot.properties	regexp wrapper for parsing tagged message properties expressed in the form @¡label¿ ¡data¿. Second outlet outputs number of backreferences.
dot.recordabsolute	Record an arbitrary number of datastreams with absolute timestamping.
dot.reg	Like zl reg but right outlet bangs when empty.
dot.sparkline	Draws a sparkline from a list or stream onto an lcd object.
dot.typecheck	Route input according to data type.
dot.urn	Generate random numbers without duplicates (like urn), but you can put numbers back in the pot.
dot.xmlread	A native-Max XML parser - no externals!
dot.xmlwrite	A native-Max XML parser - no externals!

Filters	
dot.asyncdemod~	Asynchrouous demodulation of baseband signal from an AM-carrier.

`dot.attackslope`	Given two thresholds, determine the slope between the points at which they are crossed in the positive direction. (i.e., attack speed)
`dot.autoscale`	Rescales signals according to auto-detected maximum and minimum values.
`dot.autoscale~`	Automatically scale incoming signal to a defined output-range.
`dot.boundary`	Mutes, clamps, wraps, or folds a stream of numbers at a predefined minimum and/or maximum value.
`dot.cartopol3`	Converts 3D Cartesian coordinates to polar representation.
`dot.centre`	Automatically offsets input to re-centre signal around zero, with user-definable delay, ramp time, and time grain.
`dot.change`	Just like the `change` object, but works for symbols and lists.
`dot.clip`	Clips a stream of numbers to a minimum, maximum, or both.
`dot.dampedenvelope`	Audio-rate envelope-generator with damping.
`dot.distance`	Finds maximum or minimum distance between a scalar and a list.
`dot.dynamicexpression`	User-definable expr-based scaling with autoscale.
`dot.extrema`	Outputs values of local maxima and minima.
`dot.fraction`	Looks at divisions of a list and outputs the highest value corresponding to nodes at 2 3 4 5 and 6 part divisions. Intended for use with `dot.harmonicfilter`.
`dot.fromsignal~`	Samples a signal triggered by change.
`dot.harmonicfilter`	For damping gains in modal synthesis based on a harmonic series. Handles "string divisio" at 0.5, 0.3, 0.25, 0.2, 0.167
`dot.history`	Outputs list of delayed samples: x[n], x[n-1], ..., x[n-m].

dot.hz↔samp~	Converts between frequency in Hz (cycles/second) and samples/cycle.
dot.interpolate4~	A sort of spectral interpolation of 4 signals via magnitude and phase vector.
dot.jab	Detect "jabbing" gestures in acceleration data.
dot.leakyintegrator	An accumulator with a hole in it. The leakiness is highly customizable.
dot.leakyintegrator2	Integrator with a leak — handles floating-point numbers, signed values, leak expressions.
dot.mass-spring	Implements a simple mass-spring-damper model.
dot.median	Outputs median value of a user-definable sample size.
dot.mix4~	mixes four signals via XY-coordinates.
dot.normalize	Normalizes a list of ints or floats, or a windowed stream.
dot.polar	Converts x/y into amplitude and angle. Change in angle with wrap-around correction is also calculated.
dot.poltocar3	Converts 3D polar coordinates to Cartesian representation.
dot.rad→norm~	Scales 2pi-radians to normalized range (0-1) with optional wraparound.
dot.region	Outputs and centre of multiple selected areas of a list (binary).
dot.scale~	MSP version of Max's scale-object (with 'proper' exponent).
dot.schmitt	A trigger with hysteresis.
dot.schmitt~	Detect triggers from envelope of a signal.
dot.signaccum	Accumulates positive vs. negative same-sign deltas.
dot.slope	Output the slope between each successive point.
dot.smooth	Simple sample-averaging filter.
dot.split	Right outlet if greater then threshold, left outlet otherwise.
dot.thresh	A little hack to allow thresh to work with symbols.

`dot.threshtrig`	Output a value only once after passing the threshold in the positive or negative directions.
`dot.timedextrema`	Outputs the minimum and maximum value received with the last n milliseconds.
`dot.timedsmooth`	Downsampled audio-rate averaging filter.
`dot.timedsmooth2`	Time-windowed averaging filter in which each input sample has the same weight.
`dot.transfer`	Table-based waveshaping with customizable transfer function.
`dot.unwrap`	Assumes input is polar and lies between arg1 and arg2. Output is the shortest polar distance from the last sample.
`dot.vscale`	Just like the `scale` object, but for vectors. Includes clipping-feature.
`dot.windowedextrema`	Outputs the minimum and maximum value received with the last n samples. Also outputs the order of the extrema in the window.
`dot.wrap`	Simple offset with wrap-around.

List Processing	
`dot.listinterpolate`	Interpolates/extrapolates between two lists of equal length.
`dot.listinterpolate4`	The same for four lists.
`dot.listpipe`	Delays a stream of input like `pipe`, but also works for lists.
`dot.matchNth`	Outputs message if the nth item matches the argument.
`dot.matcNth`	Outputs message if the nth item address pattern matches the argument.
`dot.nth`	Works like `zl nth`, but can match multiple indexes.

MIDI

dot.combinote	Combines 8 input signals into a single value such that any given combination will be a unique number.
dot.MIDIout	Helper for quickly configuring MIDI output.
dot.MIDIpedal~	Detect triggers from a MIDI-pedal through an audio-input.

OSC	
dot.appendaddr	Append text to the end of the first item of a list.
dot.OSCalias	Shortens long OSC addresses by giving them aliases.
dot.OSCcompress	Simply removes spaces in OSC address strings.
dot.OSCexpand	Simply expands OSC address strings so that the route object can parse them.
dot.OSCroute	Native-max OSC parser allowing multiple OSC addresses to be dynamically added and removed. Arguments or right inlet set addresses to route. Left outlet outputs matches with index corresponding to argument order.
dot.OSCunalias	Restores OSC addresses that have been aliased using dot.OSCalias.
dot.prependaddr	Prepend text to the beginning of the first item of a list.

Quaternions	
dot.jit.quaternion.conjugate	Calculate the conjugate of a quaternion stored in the planes of a jitter matrix.
dot.jit.quaternion.inverse	Calculate the inverse of a quaternion stored in the planes of a jitter matrix.
dot.jit.quaternion.multiply	Multiply two quaternions stored in the planes of a jitter matrix.
dot.quaternion.conjugate	Calculate the conjugate of a quaternion.
dot.quaternion.inverse	Calculate the inverse of a quaternion.

`dot.quaternion.multiply`	Multiply two quaternions.
`dot.quaternion.normalize`	Normalize a quaternions.
`dot.quaternion.SLERP`	Performs Spherical Linear Interpolation between two quaternions.
`dot.quaternion2axis`	Convert quaternion to axis/angle representation.

Sensors	
`dot.fqa`	Factored Quaternion Algorithm for calculating orientation from magnetometer and accelerometer data.
`dot.orient`	Calculates absolute orientation from gravity and magnetic field vectors.
`dot.wmp`	Process the data from a Wii Motion Plus and remove gyro bias.

Serial	
`dot.doubleSLIPdecode`	Parses double-ended SLIP-encoded data with user-defined start, end, and escape characters.
`dot.doubleSLIPencode`	Encodes data using double-ended slip-coding with user-defined start, end, and escape characters.
`dot.serial`	An abstraction containing the serial object, with menu generation and built-in polling and repeated-reading functionality.
`dot.SLIPdecode`	Parses slip-encoded data with user-defined delimiter and escape character.
`dot.SLIPencode`	Encodes data using slip-coding with user-defined delimiter and escape character.

Statistics	
`dot.aggregate`	Calculates aggregate of a list or a windowed stream.

dot.covariance	Calculates covariance of two windowed streams of numbers.
dot.ema	Calculates the Exponential Moving Average of a stream of numbers.
dot.emd	Calculates the Exponential Moving Deviation of a stream of numbers.
dot.exemplarcovariance	Calculates covariance of a windowed stream of numbers and an example.
dot.phase→freq~	Instantaneous frequency estimation from phase deltas.
dot.sintrack~	Track magnitude, instantaneous phase, and instantaneous frequency of a sinusoid.
dot.std	Calculates the Standard Deviation of a windowed stream of numbers.

Timing	
dot.channelthresh	Combines ID-tagged channels into lists using a delay threshold. Like `thresh`, but keeps channel information.
dot.debounce	Filters multiple messages.
dot.randometro	Like the `metro` object but outputs randomly within a range.
dot.squeuedlim	Limits the speed of messages passing through like `speedlim`, but queued like `zl queue`.
dot.wait	bangs when values are over-threshold for wait time.

XBee	
dot.decodeXBeeAPI	Communicate withan XBee radio in "API mode"
dot.encodeXBeeAPI	Communicate withan XBee radio in "API mode"

References

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

[2] S. Sinclair and M. M. Wanderley, "A run-time programmable simulator to enable multi-modal interaction with rigid-body systems," *Interacting with Computers*, vol. 21, no. 1–2, pp. 54–63, 2009.

[3] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza, "TUIO: A protocol for table-top tangible user interfaces," in *Proc. of the The 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, 2005.

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

[5] S. Jordà, "Sonigraphical instruments: from FMOL to the reacTable," in *Proceedings of the Conference on New Interfaces for Musical Expression*, (Montreal, Canada), pp. 70–76, 2003.

[6] M. Zadel, *Graphical Performance Software in Contexts: Explorations with Different Strokes*. PhD thesis, McGill University, November 2012.

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

[8] E. M. v. Hornbostel and C. Sachs, "Classification of musical instruments: Translated from the original german by anthony baines and klaus p. wachsmann," *The Galpin Society Journal*, vol. 14, pp. 3–29, 1961.

[9] S. Mann, "Natural interfaces for musical expression: Physiphones and a physics-based organology," in *Proceedings of the 2007 Conference on New Interfaces for Musical Expression (NIME-07)*, (New York, USA), pp. 118–123, 2007.

[10] M. M. Wanderley and P. Depalle, "Gestural control of sound synthesis," *Proceedings of the IEEE*, vol. 92, pp. 632–644, April 2004. Special Issue on Engineering and Music - Supervisory Control and Auditory Communication.

[11] A. Mulder, "Towards a choice of gestural constraints for instrumental performers," in *Trends in Gestural Control of Music* (M. Wanderley and M. Battier, eds.), pp. 321–352, Paris, France: IRCAM, 2000.

[12] A. Tanaka and B. Bongers, "Global string: A musical instrument for hybrid space," in *Proceedings: Cast01//Living in Mixed Realities, Fraunhofer Institut fur Medienkommunikation*, pp. 177–181, Citeseer, 2001.

[13] J. Bowers and P. Archer, "Not hyper, not meta, not cyber but infra-instruments," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, (Vancouver, Canada), pp. 5–10, May 2005.

[14] R. J. K. Jacob, L. E. Sibert, D. C. McFarlane, and J. M. Preston Mullen, "Integrality and separability of input devices," *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 1, pp. 3–26, 1994.

[15] D. A. Norman, *The design of everyday things*. Doubleday, 1990.

[16] A. Schloss, "Recent Advances in the Coupling of the Language MAX with the Mathews/Boie Radio Drum," in *Proc. of the 1990 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, pp. 398–400, 1990.

[17] J. Rasmussen, *Information Processing and Human-Machine Interaction: an Approach to Cognitive Engineering*. New York, NY, USA: Elsevier Science Inc., 1986.

[18] B. Cariou, "Design of an Alternate Controller from an Industrial Design Perspective," in *Proc. of the 1992 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, pp. 366–367, 1992.

[19] B. Cariou, "The aXiO MIDI Controller," in *Proc. of the 1994 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, pp. 163–166, 1994.

[20] J. Malloch, D. Birnbaum, E. Sinyor, and M. M. Wanderley, "Towards a new conceptual framework for digital musical instruments," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-06)*, (Montreal, Quebec, Canada), pp. 49–52, Sept. 18–20 2006.

[21] E. F. Clarke, *Generative processes in music : the psychology of performance, improvisation, and composition*, ch. 1, pp. 1–26. Oxford: Clarendon Press, 1988.

[22] M. M. Wanderley, Ed., "Mapping strategies in real-time computer music," *Organised Sound*, vol. 7, 2002.

[23] M. R. Masliah and P. Milgram, "Measuring the allocation of control in a 6 degree-of-freedom docking experiment," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, vol. 2:1, (The Hague, The Netherlands), pp. 25–31, April 2000.

[24] R. Vertegaal, "An Evaluation of Input Devices for Timbre Space Navigation," Master's thesis, Department of Computing - University of Bradford, 1994.

[25] R. Vertegaal and B. Eaglestone, "Comparison of Input Devices in an ISEE Direct Timbre Manipulation Task," *Interacting with Computers*, vol. 8, no. 1, pp. 13–30, 1996.

[26] A. Hunt, *Radical User Interfaces for Real-time Musical Control*. PhD thesis, University of York, UK, 1999.

[27] D. Stowell, M. D. Plumbley, and N. Bryan-Kinns, "Discourse analysis evaluation method for expressive musical interfaces," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, (Genova, Italy), pp. 81–86, June 2008.

[28] D. Stowell, A. Robertson, N. Bryan-Kinns, and M. D. Plumbley, "Evaluation of live human-computer music-making: Quantitative and qualitative approaches," *International journal of Human-Computer Studies*, vol. 67, no. 11, pp. 960–975, 2009.

[29] S. Harrison, D. Tatar, and P. Sengers, "The three paradigms of HCI," in *Alt. CHI Session at the SIGCHI conference on Human factors in computing systems*, (San Jose, CA), 2007.

[30] M. M. Wanderley, N. Orio, and N. Schnell, "Towards an analysis of interaction in sound generating systems," in *Proceedings of ISEA2000 Conference*, 2000.

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

[32] R. Fiebrink, *Real-time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance*. PhD thesis, Princeton University, Princeton, NJ, USA, January 2011.

[33] M. M. Wanderley, N. Schnell, and J. Rovan, "Escher - Modeling and Performing "Composed Instruments" in Real-time," in *Proc. IEEE International Conference on Systems, Man and Cybernetics (SMC'98)*, 1998.

[34] A. R. Jensenius, T. Kvifte, and R. I. Godøy, "Towards a gesture description interchange format," in *Proceedings of the International Conference on New interfaces for Musical Expression*, (Paris, France), pp. 176–179, IRCAM – Centre Pompidou, 2006.

[35] F. Bevilacqua, R. Müller, and N. Schnell, "MnM: a Max/MSP mapping toolbox," in *Proceedings of the conference on New Interfaces for Musical Expression*, (Vancouver, Canada), pp. 85–88, 2005.

[36] D. Van Nort and M. M. Wanderley, "The LoM mapping toolbox for Max/MSP/Jitter," in *Proceedings of the International Computer Music Conference*, (New Orleans, USA), 2006.

[37] H.-C. Steiner, "[hid] toolkit: a unified framework for instrument design," in *Proceedings of the 2005 International Conference on New Interfaces for Musical Expression*, (Vancouver, Canada), pp. 140–143, 2005.

[38] H.-C. Steiner, "Towards a catalog and software library of mapping methods," in *Proceedings of the conference on New Interfaces for Musical Expression*, (Paris, France), pp. 106–109, IRCAM – Centre Pompidou, 2006.

[39] H.-C. Steiner and C. Henry, "Progress report on the mapping library for pd," in *Proceedings of the PureData Convention*, (Montreal, Canada), 2007.

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

[41] J. Malloch, S. Sinclair, and M. M. Wanderley, "A network-based framework for collaborative development and performance of digital musical instruments," in *Computer Music Modeling and Retrieval - Sense of Sounds* (R. Kronland-Martinet, S. Ystad, and K. Jensen, eds.), vol. 4969 of *LNCS*, (Berlin / Heidelberg), pp. 401–425, Springer-Verlag, 2008.

[42] The Khronos Group, "Streaminput - cross-platform advanced sensor processing and user interaction." Online. Available: http://www.khronos.org/streaminput/. Accessed August 8, 2013.

[43] R. M. Taylor, II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, "VRPN: a device-independent, network-transparent VR peripheral system," in *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '01, (New York, NY, USA), pp. 55–61, ACM, 2001.

[44] M. A. Baalman, H. C. Smoak, C. L. Salter, J. Malloch, and M. M. Wanderley, "Sharing data in collaborative, interactive performances: the senseworld datanetwork," in

Proceedings of the International Conference on New Interfaces for Musical Expression, (Pittsburgh, USA), pp. 131–134, June 2009.

[45] "Sense/stage: Low cost, open source wireless sensor infrastructure for live performance and interactive, real-time environments," 2009. Available: http://sensestage.hexagram.ca/. Accessed August 8, 2013.

[46] "tapemovie - environment for realtime audio and video instruments." Online. Available: http://tapemovie.org/. Accessed June 10, 2012.

[47] Steim Foundation, "junxion." Online. Available: http://steim.org/product/junxion/. Accessed August 8, 2013.

[48] Steim Foundation, "junXion v4 manual." Online, 2008. Available: http://www.steim.org/steim/support/manuals/jXv4_Manual.pdf. Accessed October 16, 2009.

[49] J. Bullock and H. Frisk, "libIntegra: a system for software-independent multimedia module description and storage," in *Proc. of the 2007 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, (Copenhagen, Denmark), 2007.

[50] "Integra Live," 2007. http://integralive.org.

[51] T. Place and T. Lossius, "Jamoma: A modular standard for structuring patches in max," in *Proceedings of the International Computer Music Conference*, (New Orleans, USA), 2006.

[52] S. de Laubier and V. Goudard, "Puce muse-la méta-mallette," *Proceedings of Journées d'Informatique Musicale (JIM2007)*, 2007.

[53] Émilien Ghomi, *Designing Expressive Interaction Techniques for Novices Inspired by Expert Activities: the Case of Musical Practice*. PhD thesis, l'Université Paris-Sud, Paris, France, December 2012.

[54] C. Roberts, *Towards a Dynamic Framework for Interactivity*. PhD thesis, Masters thesis, University of California, Santa Barbara, 2010.

[55] "Interactive spaces." Online, 2012. Available: http://code.google.com/p/interactive-spaces/. Accessed August 8, 2013.

[56] "Robot operating system (ROS): Documentation." Online. Available: http://www.ros.org/wiki/. Accessed August 8, 2013.

[57] J.-Y. L. Lawson, A.-A. Al-Akkad, J. Vanderdonckt, and B. Macq, "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, NY, USA), pp. 245–254, ACM, 2009.

[58] M. Lee, A. Freed, and D. Wessel, "Real-time neural network processing of Gestural and Acoustic Signals," in *Proc. of the 1991 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, pp. 277–280, 1991.

[59] M. Lee and D. Wessel, "Connectionist Models for Real-Time Control of Synthesis and Compositional Algorithms," in *Proc. of the 1992 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, pp. 277–280, 1992.

[60] S. Fels, *Glove Talk II: Mapping Hand Gestures to Speech Using Neural Networks*. PhD thesis, University of Toronto, 1994.

[61] R. Fiebrink, P. R. Cook, and D. Trueman, "Play-along mapping of musical controllers," in *Proc. of the 2009 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, (Montreal, Canada), pp. 61–64, August 2009.

[62] E. Eagen and L. Haken, *EagenMatrix User Guide*, 2013.

[63] S. deLaubier, "The meta-instrument," *Computer Music J.*, vol. 22, no. 1, pp. 25–29, 1998.

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

[65] A. Hunt, *Radical User Interfaces for Real-time Musical Control*. PhD thesis, University of York, UK, 1999.

[66] MIDI Manufacturers Association, "The complete MIDI 1.0 detailed specification: Incorporating all recommended practices," 1996.

[67] F. R. Moore, "The dysfunctions of MIDI," *Computer Music Journal*, vol. 12, no. 1, pp. 19–28, 1988.

[68] M. Wright, "A comparison of MIDI and ZIPI," *Computer Music Journal*, vol. 18, no. 4, pp. 86–91, 1994.

[69] K. McMillen, "ZIPI: Origins and motivations," *Computer Music Journal*, vol. 18, no. 4, pp. 47–51, 1994.

[70] P. R. Cook and G. Scavone, "The synthesis toolkit (stk)," in *Proceedings of the International Computer Music Conference*, pp. 164–166, 1999.

[71] M. Wright, A. Freed, and A. Momeni, "OpenSound Control: State of the art 2003," in *Proceedings of the Conference on New Interfaces for Musical Expression*, 2003.

[72] J. Bullock and H. Frisk, "The Integra framework for rapid modular audio application development," in *Proc. International Computer Music Conference*, (Huddersfield, UK), University of Huddersfield, 2011.

[73] S. Schiesser, "midOSC: a Gumstix-based MIDI-to-OSC converter," in *Proc. New Interfaces for Musical Expression*, pp. 165–168, 2009.

[74] Entertainment Services and Technology Association, "ANSI E1.11-2004 Entertainment technology USITT DMX512-A asynchronous serial digital data transmission standard for controlling lighting equipment and accessories." Online, 2004.

[75] Entertainment Services and Technology Association, "ANSI E1.17-2010 Architecture for control networks device description language (DDL)." Online, 2011. Available: http://tsp.plasa.org/tsp/documents/docs/ACN-ddl_2009-1024r1_free.pdf. Accessed June 10, 2012.

[76] IEEE, "IEEE Standard for a smart transducer interface for sensors and actuators wireless communication protocols and transducer electronic data sheet (TEDS) formats," *IEEE Std 1451.5-2007*, 5 2007.

[77] "Transducer markup language." Online. Available: http://www.transducerml.org/. Accessed September 12, 2009.

[78] R. M. Taylor II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, "VRPN: a device-independent, network-transparent VR peripheral system," in *Proc. of the ACM symposium on Virtual reality software and technology*, pp. 55–61, ACM, 2001.

[79] J. Malloch, S. Sinclair, and M. M. Wanderley, "Libmapper (a library for connecting things)," in *Proc. of the International Conference on Human Factors in Computing Systems (CHI2013)*, (New York, NY, USA), pp. 3087–3090, ACM, 2013. extended abstract.

[80] B. F. E. Cronin, A. R. Kurc, "An efficient synchronization mechanism for mirrored game architectures," *Multimedia tools and applications*, vol. 23, pp. 7—30, May 2004.

[81] Apple, "Bonjour printing specifications," 2005. Available: https://developer.apple.com/bonjour/printing-specification/bonjourprinting-1.0.2.pdf. Accessed March 2013.

[82] R. P. Soucy, "IP multicast explained," 2012. Available: http://www.soucy.org/network/multicast.php. Accessed March 2013.

[83] R. Muller, "OSCBonjour," 2006. Available: http://recherche.ircam.fr/equipes/temps-reel/movement/muller/index.php?entry=entry060616-173626. Accessed March 2013.

[84] R. Steinmetz and K. Wehrle, *Peer-to-peer systems and applications*, vol. 3485. Springer, 2005.

[85] H. Karl and A. Willig, *Protocols and architectures for wireless sensor networks*. Wiley-Interscience, 2007.

[86] A. Schmeder, "Efficient gesture storage and retrieval for multiple applications using a relational data model of open sound control," in *Proc. International Computer Music Conference*, 2009.

[87] M. Ressel and R. Gunzenhäuser, "Reducing the problems of group undo," in *Proc. of the international ACM SIGGROUP conference on Supporting group work*, GROUP '99, (New York, NY, USA), pp. 131–139, ACM, 1999.

[88] F. Bevilacqua, R. Müller, and N. Schnell, "MnM: a Max/MSP mapping toolbox," in *Proc. New Interfaces for Musical Expression*, (Vancouver, Canada), pp. 85–88, University of British Columbia, 2005.

[89] H.-C. Steiner, "Firmata: Towards making microcontrollers act like extensions of the computer," in *Proc. New Interfaces for Musical Expression*, (Pittsburgh, PA), pp. 125–130, Carnegie Mellon University, 2009.

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

[91] A. Hunt and R. Kirk, *Trends in Gestural Control of Music*, ch. Mapping Strategies for Musical Performance. Ircam - Centre Pompidou, 2000.

[92] R. Jacob, L. Sibert, D. Mcfarlane, and M. P. Mullen, "Integrality and Separability of Input Devices," *ACM Transactions in Human-Computer Interaction*, vol. 1, no. 1, pp. 3–26, 1994.

[93] A. Hunt, M. M. Wanderley, and R. Kirk, "Towards a model for instrumental mapping in expert musical interaction," in *Proceedings of the International Computer Music Conference*, (San Francisco), pp. 209–212, International Computer Music Association, 2000.

[94] J. Lazzaro and J. Wawrzynek, "An RTP payload format for MIDI," in *Audio Engineering Society Convention 117*, Audio Engineering Society, 2004.

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

[96] R. Diestel, *Graph Theory*. Springer-Verlag, 2005.

[97] M. Zadel and G. Scavone, "Different strokes: a prototype software system for laptop performance and improvisation," in *Proceedings of the 2006 Conference on New Interfaces for Musical Expression*, pp. 168–171, IRCAM, Centre Pompidou, 2006.

[98] M. Zadel and G. Scavone, "Recent developments in the different strokes environment," in *Proc. of the 2008 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, 2008.

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

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

[101] J. Malloch, S. Sinclair, A. Hollinger, and M. M. Wanderley, "Input Devices and Music Interaction," in *Musical Robots and Interactive Music Systems* (J. Solis and K. Ng, eds.), Springer Verlag, 2011.

[102] D. A. Stewart, "Vigorous music-making: The inherent "liveliness" of a T-Stick instrumentalist," in *Proc. of the 2010 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, (Ann Arbour, USA), Ann Arbor, MI: MPublishing, University of Michigan Library, 2010.

[103] X. Pestova, E. Donald, H. Hindman, J. Malloch, M. T. Marshall, F. Rocha, S. Sinclair, D. A. Stewart, M. M. Wanderley, and S. Ferguson, "The CIRMMT/McGill digital orchestra project," in *Proc. of the 2009 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, (Montreal), 2009.

[104] S. Ferguson and M. M. Wanderley, "The McGill digital orchestra: An interdisciplinary project on digital musical instruments," *Journal of Interdisciplinary Music Studies*, vol. 4, no. 2, pp. 17–35, 2010.

[105] R. Koehly, D. Curtil, and M. M. Wanderley, "Paper FSRs and latex/fabric traction sensors: Methods for the development of home-made touch sensors," in *NIME '06: Proceedings of the 2006 conference on New interfaces for musical expression*, (Paris, France), pp. 230–233, IRCAM—Centre Pompidou, 2006.

[106] M. T. Marshall, "Building a USB sensor interface." Online. Available: `http://www.sensorwiki.org/`. Accessed October 24, 2009.

[107] C. Chafe, "Tactile audio feedback.," in *Proceedings of the International Computer Music Conference*, pp. 76–79, ICMA, 1993.

[108] C. Chafe and S. O'Modhrain, "Musical muscle memory and the haptic display of performance nuance," in *Proceedings of the International Computer Music Conference*, (Hong Kong), pp. 428–431, ICMA, 1996.

[109] S. M. O'Modhrain, *Playing by Feel: Incorporating Haptic Feedback into Computer-Based musical Instruments*. PhD thesis, Stanford University, November 2000.

[110] B. Bongers, "The use of Active Tactile and Force Feedback in Timbre Controlling Electronic Instruments," in *Proc. of the 1994 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, pp. 171–174, 1994.

[111] B. Bongers, "Tactile display in electronic musical instruments," in *IEE Colloquium on Developments in Tactile Displays (Digest No. 1997/012)*, (London, UK), pp. 7/1–7/3, January 1997.

[112] B. Bongers, G. van der Veer, and M. Simon, "Improving gestural articulation through active tactual feedback in musical instruments," in *Symposium on Gesture Interfaces for Multimedia Systems*, (Leeds UK), March 2004.

[113] J. Rovan and V. Hayward, "Typology of tactile sounds and their synthesis in gesture-driven computer music performance," in *Trends in Gestural Control of Music* (M. M. Wanderley and M. Battier, eds.), pp. 297–320, Paris, France: IRCAM, 2000.

[114] M. T. Marshall, *Physical Interface Design for Digital Musical Instruments*. PhD thesis, McGill University, Montreal, Canada, March 2009.

[115] D. Birnbaum and M. M. Wanderley, "A systematic approach to musical vibrotactile feedback," in *Proceedings of the Internation Computer Music Conference*, (Copenhagen, Denmark), pp. 397–401, August 2007.

[116] M. Giordano and M. M. Wanderley, "A learning interface for novice guitar players using vibrotactile stimulation," in *Proceedings of the 8th Sound and Music Computing (SMC) Conference*, 2011.

[117] M. T. Marshall and M. M. Wanderley, "Vibrotactile feedback in digital musical instruments," in *Proceedings of the Conference on New Interfaces for Musical Expression*, (Paris, France), pp. 226–229, IRCAM—Centre Pompidou, June 2006.

[118] H.-Y. Yoa, "Touch magnifying instrument applied to minimally invasive surgery," Master's thesis, McGill University, Montreal, Canada, 2004.

[119] H.-Y. Yao and V. Hayward, "An experiment on length perception with a virtual rolling stone," in *Proceedings of Eurohaptics*, (Paris, France), pp. 325–330, July 2006.

[120] C. Konvalin, "Compensating for tilt, hard iron and soft iron effects," tech. rep., Memsense, 2008.

[121] G. F. Welch, "History: The use of the Kalman filter for human motion tracking in virtual reality," *Presence*, vol. 18, pp. 72–91, February 2009.

[122] E. R. Bachman, *Inertial and Magnetic Tracking of Limb Segment Orientation for Inserting Humans into Synthetic Environments*. PhD thesis, Naval Postgraduate School, United States Navy, December 2000.

[123] A. Young, "Comparison of orientation filter algorithms for realtime wireless inertial posture tracking," in *Sixth International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2009)*, pp. 59–64, IEEE, 2009.

[124] J. M. Cooke, *NPSNET: Flight simulation dynamic modeling using quaternions*. PhD thesis, Monterey, California. Naval Postgraduate School, 1992.

[125] D. Wessel and M. Wright, "Problems and prospects for intimate control of computers," *Computer Music Journal*, vol. 26:3, pp. 11–22, Fall 2002.

[126] M. Waiswisz, "The Hands, a Set of Remote MIDI-Controllers," in *Proc. of the 1985 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association.*, pp. 313–318, 1985.

[127] B. Bongers, "Physical interfaces in the electronic arts," in *Trends in Gestural Control* (M. Wanderley and M. Battier, eds.), Centre Pompidou: Paris: IRCAM, 2000.

[128] J. Piringer, "Elektronische musik und interaktivität: Prinzipien, konzepte, anwendungen," diplom arbeit thesis, Institut für Gestaltungs und Wirkungsforschung der Technischen Universität Wien, 2001.

[129] T. Blaine and S. Fels, "Contexts of collaborative musical experiences," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, (Montreal, Canada), pp. 129–134, May 2003.

[130] D. Levitin, S. McAdams, and R. L. Adams, "Control parameters for musical instruments: a foundation for new mappings of gesture to sound," *Organised Sound*, vol. 7:2, pp. 171–189, 2002.

[131] L. E. Miller, "Cage, cunningham, and collaborators: The odyssey of variations v," *Musical Quarterly*, vol. 85, no. 3, pp. 545–567, 2001.

[132] A. Camurri, B. Mazzarino, and G. Volpe, "Analysis of expressive gesture: The eyesweb expressive gesture processing library," in *Gesture-based communication in human-computer interaction*, pp. 460–467, Springer, 2004.

[133] T. Winkler, "Creating interactive dance with the very nervous system," in *Proceedings of the Connecticut College Symposium on Arts and Technology*, (New London, CT.), pp. 212–217, 1997.

[134] S. A. v. D. Skogstad, K. Nymoen, Y. de Quay, and A. R. Jensenius, "OSC implementation and evaluation of the Xsens MVN suit," in *Proceedings of the 2011 Conference on New Interfaces for Musical Expression*, pp. 300–303, 2011.

[135] W. Siegel and J. Jacobsen, "The challenges of interactive dance: An overview and case study," *Computer Music Journal*, vol. 22, no. 4, pp. 29–43, 1998.

[136] A. Mulder, *Design of Virtual Three-dimensional Instruments for Sound Control*. PhD thesis, Simon Fraser University, 1998.

[137] B. Bongers, *Trends in Gestural Control of Music*, ch. Physical Interfaces in the Electronic Arts. Interaction Theory and Interfacing Techniques for Real-time Performance. Ircam - Centre Pompidou, 2000.

[138] M. Mori, "The uncanny valley," *Energy*, vol. 7, no. 4, pp. 33–35, 1970.

[139] S. Mann, "Wearable computing: A first step toward personal imaging," *Computer*, vol. 30, no. 2, pp. 25–32, 1997.

[140] S. Jordà, "Digital instruments and players: part i—efficiency and apprenticeship," in *Proceedings of the 2004 Conference on New Interfaces for Musical Expression*, (Hamamatsu, Japan), pp. 59–63, 2004.

[141] P. Cook, "Principles for designing computer music controllers," in *Proceedings of the 2001 conference on New Interfaces for Musical Expression*, (Seattle, USA), pp. 1–4, 2001.

[142] ZigBee Alliance, "Zigbee specification," *Document 053474r06, Version*, vol. 1, 2006.

[143] "CIRMMT live electronics framework," 2013. http://sourceforge.net/projects/clef/.