FROM CONTROLLER TO SOUND: TOOLS FOR COLLABORATIVE DEVELOPMENT OF DIGITAL MUSICAL INSTRUMENTS

Joseph Malloch, Stephen Sinclair, and Marcelo M. Wanderley
Input Devices and Music Interaction Laboratory
Centre for Interdisciplinary Research in Music Media and Technology
McGill University – Montreal, QC, Canada
joseph.malloch@mcgill.ca, sinclair@music.mcgill.ca,
marcelo.wanderley@mcgill.ca

1. ABSTRACT

This paper describes the design and implementation of a system for collaborative development of a digital musical instrument mapping layer. System development included the design of a decentralized network for the management of peer-to-peer data connections using Open Sound Control. A graphical user interface for dynamically creating, modifying, and destroying mappings between control data streams and synthesis parameters is presented.

2. INTRODUCTION

Although designers of Digital Musical Instruments (DMI) are interested in creating useful, flexible, and creatively-inspiring interfaces and sounds, this process often depends on the vision and insight of a single individual. The McGill Digital Orchestra project instead brings together research-creators and researchers in performance, composition and music technology to work collaboratively in creating tools for live performance with digital technology [1]. A large part of this research focuses on developing new musical interfaces. [1]

In the process of creating instruments for this project, we have found ourselves faced with the unique challenge of mapping new instruments in collaboration with experienced performers, as well as with composers tasked with writing pieces for these instruments. Because this ambitious project has taken on these three main challenges of the digital performance medium simultaneously, we have found ourselves in need of tools to help optimize the process. Specifically, mapping the various streams of controller output to the input parameters of synthesis engines has presented us with situations where both ease of use and flexibility were both of the utmost importance. We needed to be able to modify connections between data streams during precious engineer-composer-performer meeting time, while minimizing wasted minutes "reprogramming" our signal processing routines. Although arguably both powerful and intuitive, even the graphical environment provided by Max/MSP did not seem appropriate for these purposes, because non-programmers who had limited familiarity with such tools were expected to help in experimentation and design.

In consideration of several ongoing projects, including GDIF [7], Jamoma [10], Integra [2], and OpenSound Control (OSC) [13], we have created a "plug and play" network environment. Controllers and synthesizers are able to announce their presence and make their input and output parameters available for abitrary connections, negotiated using a graphical mapping tool implemented in Max/MSP. Any controller is able to connect to any synthesizer "on the fly," while performing data scaling, clipping, and other operations.

In the course of developing an adequate working environment for this project, we have made developments in three main areas: the design of a network architecture which lends itself to a distributed "orchestral neighbourhood", in which controllers and synthesizers can interface with each other over the Max/MSP or UDP/IP buses by means of an OSC-controlled arbitrator; the creation of a "toolbox" containing many useful functions which we found ourselves using repeatedly, coded as Max/MSP abstractions; and lastly a graphical mapping tool with which gestural data streams can be dynamically connected and modified.

We have tried to create a GUI that is intuitive and transparent: relationships between parameters are visible at a glance, and changing mappings and scaling requires only a few mouse clicks. We have used all of these tools in a real collaborative context, allowing us to present not only implementations, but also observations of their effect on our group dynamic and workflow. We have tried to create an interface that is useful not only for technical users, but also as a creative tool for composers and performers.

3. GESTURAL MAPPING

The digital instrument builder is faced with several tasks: after considering what sensors should be used, how the

[1] The McGill Digital Orchestra is a research/creation project supported by the *Appui à la recherche-création* program of the *Fonds de recherche sur la société et la culture* (FQRSC) of the Quebec government, and will culminate with concert performances of new works during the 2008 MusiMars/MusiMarch Festival in Montréal.

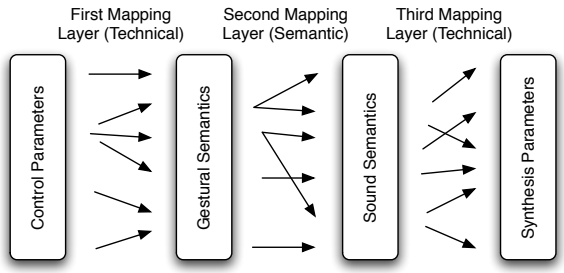

Figure 1. A diagram of the 3-layer framework used for Digital Orchestra development, adapted from [4].

musician will likely interface with them, and what sounds the instrument will make, there is still the decision of which sensors should control which aspects of the sound. This task, known as *mapping* , is an integral part of the process of creating a new musical instrument [6].

3.1. The Semantic Layer

An important result of previous discussions on mapping has been the acknowledgement of the need for a multi-layered topology. Specifically, Hunt and Wanderley [4] suggested the need for 3 layers of mapping, in which the first and last layers are device-specific mappings between technical control parameters and gestures (in the case of the first) or aesthetically meaningful "sound parameters", such as *brightness* or *position* (in the case of the third). This leaves the middle layer for mapping between parameter names that carry proper gesture and sound semantics. We shall refer to this layer as the "semantic layer", as described in Figure 1.

The tools presented here adhere to this idea. However, since the first and last mapping layers are device-specific, the mapping between technical and semantic parameters (layers 1 and 3) are considered to be part of the controller and synthesizer interfaces. Using an appropriate OSC addressing namespace, controllers present all available parameters (gestural and technical) to the mapping tool. The tool is used to create and modify the semantic layer, with the option of using technical parameters if needed.

As a simple example, the T-Stick interface [8] presents the controller's accelerometer data for mapping, but also offers an event-based "jabbing" gesture which is extracted from the accelerometers. The former is an example of layer 1 data which can be mapped directly to a synthesizer parameter. The latter is gestural parameter presented by layer 2, which can be mapped, for example, to a sound envelope trigger. The mapping between layer 1 and layer 2 for the "jabbing" gesture, (what we call *gesture extraction*), occurs in the T-Stick's interface patch.

We have also applied this system to gesture control of sound spatialization parameters [9], in which case a technical mapping layer exposes abstract spatialization parameters (such as sound source trajectories) to the semantic layer, rather than synthesis parameters.

3.2. Connection Processing

Gestural data and sound parameters will necessarily carry different units of measurement. On the gestural side, we have tried, whenever possible, to use units related to physical measurements: distance in meters, angles in degrees. In sound synthesis, units tend to be more arbitrary, but some standard ones such as Hertz and MIDI note number are obvious. In any case, data ranges will differ significantly between controller outputs and synthesis inputs. The mapping tool attempts to handle this by providing several features related to data processing.

One interesting data processing tool that we are exploring is a filter system for performing integration and differentiation. We have often found during sessions that a particular gesture might be more interesting if we could map its energy or its rate of change instead of the value directly [3]. Currently the data processing is limited to first-order FIR and IIR filtering operations, and anything more complex must be added as needed to the "gesture" mapping layer and included in the mappable namespace.

3.3. Divergent and Convergent Mapping

It has been found in previous research that for expert interaction, complex mappings are more satisfying than simple mappings. In other words, connecting a single sensor or gestural parameter to a single sound parameter will result in a less interesting feel for the performer [11, 6].

Of course, since our goal is to use abstracted gesture-level parameters in mapping as much as possible, simple mappings in the semantic layer are in fact already complex and multi-dimensional [5]. Still, we found it would be useful to be able to create one-to-many mappings, and so the mapping tool we present here supports this. Each connection may have different scaling or clipping applied.

We also considered the use of allowing the tool to create many-to-one mappings. The implication is that there must be some *combining* function which is able to arbitrate between the various inputs. Should they be summed, or perhaps multiplied, or should some sort of comparison be made between each of the inputs?

A combining function implies some relationship between gestural parameters; in some cases, the combination of gestural data may itself imply the extraction of a distinct gesture, and should be calculated on the first mapping layer and presented to the mapping tool as a single parameter. In other cases the combination may imply a complex relationship between synthesis parameters that could be better coded as part of the abstracted synthesis layer. In yet other cases the picture is ambiguous, but the prospect of needing to create on-the-fly many-to-one mappings during a working session seemed to be unlikely. We did not implement any methods for selecting combining functions, and for the moment we have left many-to-one mappings for future work.

4. THE ORCHESTRAL NETWORK NEIGHBOURHOOD

In our system, there are several entities on the network that must communicate with each other in a common language. These include controllers and synthesizers, as well as the software devices used for address translation and data processing, called *routers*, and finally the GUI used to create and destroy connections. This protocol must allow them to perform some basic administration tasks, including: announcing presence on the network; deciding what name and port to use; and finally describing what messages it can send and receive.

The system can be thought of as a higher-level protocol running on top of an OSC layer. While it has been decided that all entities on the network will speak OSC to each other, OSC itself dictates nothing about what lower-level transport protocols and ports to use, nor what kinds of messages should be exchanged. We needed to devise a common set of OSC messages to allow the use of a standard interface to control all devices in question.

4.1. Topology and Protocol

Because OSC addressing is designed to uniquely identify any particular value, it is possible to broadcast messages on a common bus and have them be received by the intended recipient. This makes it mostly trivial to switch between various network topologies. While a common bus is necessary for locating devices, it is not necessary nor is it optimal to have gestural data streams sharing the same bus.

We have decided to use a multicast UDP/IP port for administrative tasks such as device announcement and resource allocation. This common port is needed to resolve conflicting device identifiers and to allow new devices to negotiate for a unique private port on which to receiving messages. We shall refer to it as the "admin bus".

For routing mapped data streams, several topologies can be used. Though it simplifies programming, sharing a bus for high-traffic gestural streams wastes communication as well as processing resources. Messages must be received and addresses must be parsed before being rejected. If several devices are present on the network, a high percentage of traffic may be rejected, making a common bus inefficient. In our system, each device reserves a UDP/IP port for receiving data streams. Thus the OSC traffic is quickly routed and filtered on the transport layer and address parsing is only necessary for properly targeted messages.

Another factor affecting the network topology is the role of the router in mapping. Currently, controllers send their data streams to a router which performs address mapping and scaling before re-transmitting to a synthesizer. This implies a centralized topology as seen in Figure 2. However, with the protocols described in this section, it is perfectly feasible to have multiple router instances on the network. This can help reduce traffic loads and distribute

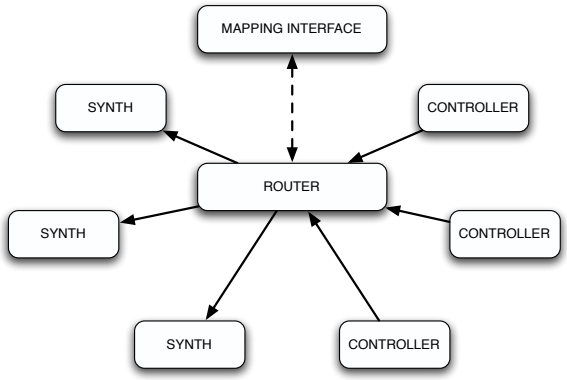

Figure 2. A centralized topology in which all traffic is routed through a central router service.

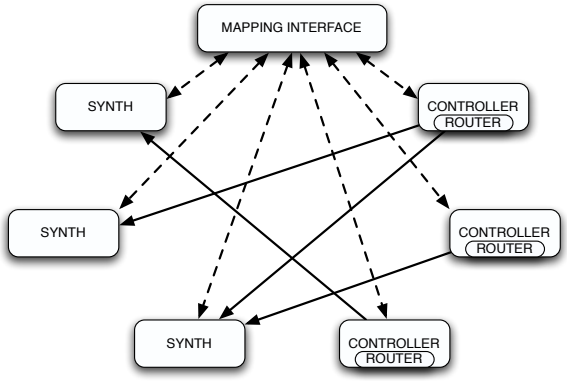

Figure 3. Equivalently, a router can be embedded in each controller to create a true peer-to-peer network.

processing. Taking it to the extreme, we propose appending a router to each controller, in order to create a truly peer-to-peer topology, as described by Figure 3.

4.2. Name and port allocation

When an entity first appears on the network, it must choose a port to be used for listening to incoming data streams. It must also give itself a unique name by which it can be referred to. A simple solution would be to assign each device a static name and port. However, we are not interested in maintaining a public database of "claimed" ports, and, (being a digital *orchestra*), we expect multiple instances of a particular device to be availble for use.

In an attempt to be more dynamic and decentralized, we have developed a collision algorithm for port and name allocation: when a new entity announces itself, it posts to the admin bus a message stating which port it tentatively intends to use. If this port is reserved or is also being asked for by another device, a random number is added and the device tries again. If multiple devices are attempting to reserve the same numbers, several iterations may occur, but eventually each device ends with a unique port number to use. (Strictly speaking, this is only necessary for devices hosted on the same computer.) The same algorithm is used for determining a device name composed

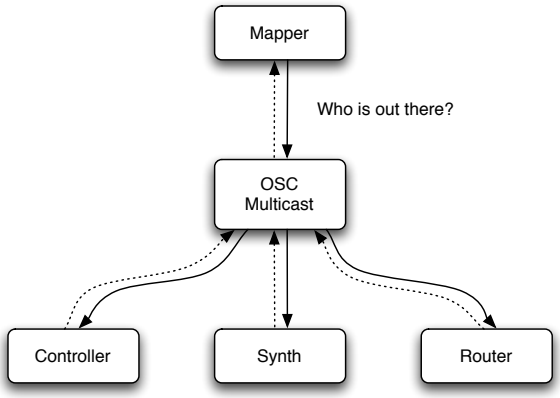

Figure 4. The mapper GUI requests that devices on the network identify themselves.

of the device class and a unique number.

4.3. Discovery

Device discovery is an important step toward a truly "plug and play" environment. When a device appears on the network, and after it successfully receives a unique name and port number, it announces its presence with a simple message stating its function, name, and IP address and port:

Router `/device/router /router/1 192.168.0.3 8003`

Controller `/device/output /tstick/1 192.168.0.3 8001`

Synth `/device/input /granul8/1 192.168.0.4 8000`

When the mapping interface is launched, it needs to be able to query the network for devices. It does so by submitting a simple request:

Mapper `/device/who`

All entities will respond by repeating their announce message. The mapping interface listens for the announce messages and lists the available devices in its drop-down menus.

An alternative method of performing device discovery, using the ZeroConf protocol, has been proposed at the 2006 OSC developers' meeting. Since then, a Max/MSP implementation of the idea, called *OSCBonjour*, has been created by Rémy Müller, which we have explored for the above tasks. The idea is promising, but does not yet handle more than device discovery. We decided that, for the moment, a pure OSC solution is adequate for our purposes, but this does not exclude the possiblity of using *OSCBonjour* in the future.

4.4. Namespace queries

Lastly, each orchestra member must be able to tell others what it can do. In other words, it must be able to say what messages it can receive and what messages it can send. Wright et al. [13] proposed the use of the `/namespace` message for causing a device to enumerate its available namespace. We have implemented this for each transmitter and receiver on the network. In addition to listing the namespace itself, each available parameter optionally can include information about data type, data range and units used. These come into play when the mapper must set up automatic scaling between data streams, as described below.

In order to make this metadata optional, we have used a tagged argument scheme, similar to the Jamoma syntax. In the example below, the mapper interface communicates with a controlled named "/tstick/1" and a granular synthesizer named "/granul8/1".

Mapper `/tstick/1/namespace`

Controller `/tstick/1/namespace/output /tstick/1/raw/piezo @type i @min 0 @max 255 @units na`

Controller `/tstick/1/namespace/output /tstick/1/...`

Mapper `/granul8/1/namespace`

Synth `/granul8/1/namespace/input /granul8/1/reverb/mix @type f @units normalized @min 0 @max 1`

Synth `/granul8/1/namespace/input /granul8/1/...`

5. THE DIGITAL ORCHESTRA TOOLBOX

In the process of creating controller, synthesizer, and mapping patches, we have made an effort to modularize any commonly used subroutines. These have been organized, with help patches, into a toolbox that we find ourselves re-using quite often. This toolbox is will be available on the Digital Orchestra website [1].

The following is a short description of some of the available abstractions:

dot.admin Uses other toolbox abstractions to handle communication on the Orchestral Network Neighbourhood multicast bus.

dot.alloc The abstracted algorithm used by **dot.admin** for allocating a unique port and device name.

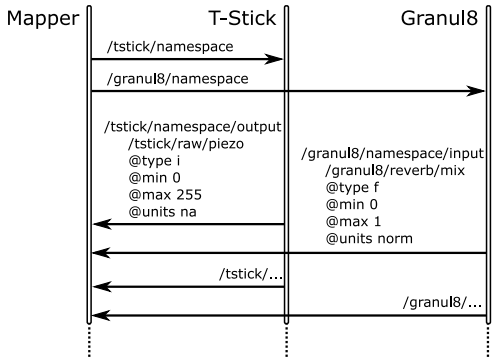

Figure 5. A namespace request causes devices to return their parameter space with information on ranges and units.

dot.autoexpr Given a destination maximum and minimum, **dot.autoexpr** will automatically adjust the linear scaling coefficients for a stream of incoming data. Calibration can be turned on and off. It can also handle arbitrary mathematical expressions and dynamically instantiate objects necessary for performing specified transformations, including first-order FIR and IIR filters.

dot.dampedenvelope An envelope signal generator which emulates the behaviour of a resonating object which can be damped.

dot.extrema Automatically outputs local maxima and minima as peaks and troughs are detected in the incoming data.

dot.index Keeps track of used indexes and can output the lowest unused slot.

dot.leakyintegrator A configurable integrator which leaks over time. The integration can either be linear, exponential, or an arbitrary transfer function can be specified as a table.

dot.play/dot.record These objects can be used to record up to 254 incoming data channels into a `coll` object, and later played back at the same rate.

dot.prependaddr Prepends the given symbol onto the first symbol of a list, without affecting the remaining list members. This is intended for constructing OSC addresses.

dot.region Outputs widths and centres of multiple selected areas in a list of binary numbers.

dot.timedsmooth A down-sampled audio-rate averaging filter.

dot.transfer Performs table-based waveshaping on control data streams with customizable transfer function.

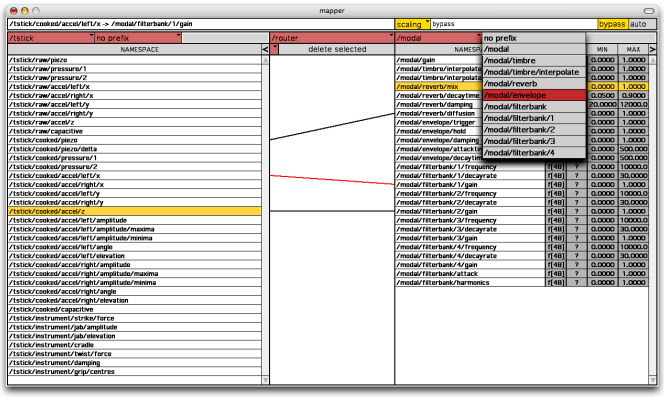

Figure 6. The mapping graphical user interface can be used to explore the available namespace, make connections, and specify scaling and other data processing.

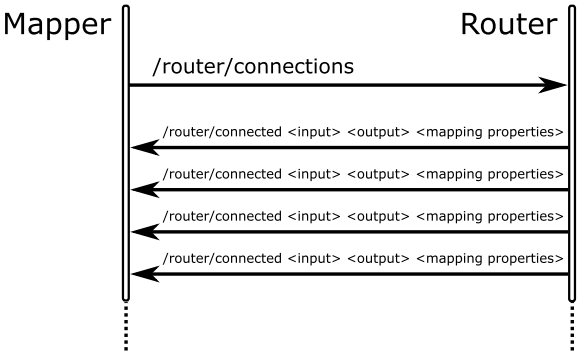

Figure 7. A router device can report its current set of connections. The mapper GUI requests it when the router is first selected.

6. THE MAPPING INTERFACE

A mapping interface has been developed to act as a GUI for mapping tasks. It forms a separate program from the Router, and communicates with controllers, synths, and routers using OSC. In addition to allowing the negotiation of mapping connections from another location on the network, this approach allows for the simultaneous use of multiple mapping interfaces, with multiple users collaborating to map the parameters of a common set of controllers and synths. The Mapping interface has several main functions:

When a mapper is launched, it asks discovered routers for their connection status. Connections are displayed on the screen for selected devices.

6.1. Browsing the Network Neighbourhood

The first use of the mapping interface is naturally choosing the devices which you wish to work with, both for ges-

ture and for sound synthesis. The interface queries devices on the network, to discover mappable inputs and outputs, and displays this information in an easily understandable format. Drop-down menus are used to select a device or devices.

6.2. Browsing and searching namespaces

In this capacity the mapping interface communicates directly with the various devices present on the network, requesting them to report their namespaces when necessary and displaying this information to the user. In addition to the namespace itself, some other information about each parameter is requested, including whether it is an *input* or an *output*, the data type (*i*, *f*, *s*, etc.), the unit type associated with the parameter (Hz, cm, etc.), and the minimum and maximum possible values. OSC address patterns for controller outputs are displayed on the left side of the mapping interface, and synthesizer inputs are displayed on the right.

In order to manage the browsing and mapping of very large namespaces, the mapping interface also allows for powerful filtering and searching using pattern-matching. Two stages of namespace filtering are available, which may be used together. One stage allows filtering by OSC address-pattern prefix, chosen from a drop down menu, so that the user may view the set of parameters which are children of a particular node in the address hierarchy. The other stage allows filtering by regular expression, so that only parameters matching a particular pattern are displayed.

On occasions where the namespace can change, such as for entities that have a configurable interface, addition or removal of addresses is announced on the multicast bus so that routers can destroy any connections appropriately, and mappers can add or remove entries.

6.3. Negotiating mapping connections and properties

In this capacity the mapping interface communicates only with a specific router, acting as a GUI for controlling the router's function, and supplying important visual feedback regarding the router's internal state to the user. Simple and intuitive methods are provided for creating and destroying mapping connections, and for editing the properties of existing connections (i.e.: scaling, clipping). Connections are created by selecting displayed namespaces on each side of the interface (inputs and outputs), and lines are drawn connecting mapped parameters. Mapping connections can be selected (in which case they are displayed in red) for editing or deletion. By selecting multiple namespaces or connections, many mappings can be created, edited, or destroyed together.

When a connection is made, the mapping tool defaults to performing no operation on the data ("bypass"). A button labeled "linear" instructs the tool to perform basic linear scaling between the provided data ranges. A button labeled "auto" turns on and off calibration of the scaling using the detected minima and maxima of the input data stream. The user can also manually type "linear" in an expression textbox with arguments defining a specific input and output range. Options are also available for defining a clipping range.

The expression box is quite versatile. For more advanced users, it accepts any string which can be understood by Max/MSP's "expr" object. Expressions can refer to the current value, or a single previous input or output sample. It may be used to specify non-linear and logarithmic mappings, for example. There is currently no support for table-based transfer functions.

6.4. Message Examples

6.4.1. The user selects controller namespace "/tstick/instrument/damping", synth namespace "/granul8/1/grain/1/filter/frequency":

Mapper `/router/1/connect <controller parameter> <synth parameter>`

Example `/router/1/connect /tstick/1/instrument/damping /granul8/1/grain/1/filter/frequency`

6.4.2. The router receives the message and creates mapping with default parameters:

Router `/router/1/connected <controller parameter> <synth parameter> <properties>`

Example `/router/1/connected /tstick/1/instrument/damping /granul8/1/grain/1/filter/frequency @scaling bypass @clipping none`

6.4.3. The user applies auto scaling to the mapping:

Mapper `/router/1/modify <controller parameter> <synth parameter> <properties>`

Example `/router/1/modify /tstick/1/instrument/damping /granul8/1/grain/1/filter/frequency @scaling auto 20 1000`

Router `/router/1/modified <controller parameter> <synth parameter> <properties>`

Example `/router/1/modified /tstick/1/instrument/damping /granul8/1/grain/1/filter/frequency @scaling auto 20 1000`

Example `/router/1/modified /tstick/1/instrument/damping /granul8/1/grain/1/filter/frequency @scaling (x-32)*0.00345+100`

6.4.4. The user ends calibration:

Mapper `/router/1/modify` `<controller parameter>` `<synth parameter>` `@calibration 0`

Example `/router/1/modify /tstick/1/instrument/damping /granul8/1/grain/1/filter/frequency @calibration 0`

6.4.5. The user deletes the mapping:

Mapper `/router/1/disconnect` `<controller parameter>` `<synth parameter>`

Example `/router/1/disconnect /tstick/1/instrument/damping /granul8/1/grain/1/filter/frequency`

Router `/router/1/disconnected` `<controller parameter>` `<synth parameter>`

Example `/router/1/disconnected /tstick/1/instrument/damping /granul8/1/grain/1/filter/frequency`

6.5. Saving and Loading Mapping-sets

Mappings can be saved to disk and loaded for later use. This task is taken care of by the router, which is currently implemented as a Max/MSP *coll* text file. We consider that this information may be more useful as XML data, since it is language agnostic and could be more easily imported into other implementations. We will define an XML format for mapping data in the near future.

7. DISCUSSION

From their earliest use, the solutions we have developed have allowed us to streamline the process of mapping in collaboration with performers and composers. The ability to quickly experiment with a variety of mapping connections democratizes the mapping process, since it is easier to try everyone's ideas during a mapping session. Showing the performers that the connections are malleable allows them to contribute to the development of a comfortable gestural vocabulary for the instrument, rather than accepting the mappings provided. Composers are able to explore control of sounds that interest them without supervision or assistance of a technical member. Using common tools for the group means that the work of others is easily viewed and understood.

Controllers and synths that are still in development are also easily supported: as the supported parameter-space increases, the device simply presents more namespaces to the GUI.

Naturally this system does not solve all of the problems encountered in a collaborative effort of this type. The technical knowledge of the group members varies widely, and some technical knowledge of the individual controllers and synths is still necessary, not least because they are still in development and may not always respond predictably. As much as possible, however, we have made the connection, processing, and communication of data between devices easy to both comprehend and perform.

One area of frustration in our work has been dealing with devices (specifically software synths) which communicate solely using MIDI. Since the norm in this case is to use MIDI control-change messages, many software environments allow flexible mapping between MIDI input values and their internal semantically labeled synth parameters. This means that although the synth parameters are easily understood from within a sequencing environment for adjustment or automation, external access to these parameters is provided only through an arbitrary set of MIDI control change identifiers. One solution is to create a static set of MIDI mappings for our use, and provide a translation layer outside the environment to expose semantic parameters identical to those used internally. In practise, however, convincing users to abandon the MIDI mapping layer has proven more difficult than we anticipated.

In namespace design we have tried throughout to conform to the hierarchy proposed for GDIF [7], since we are also involved in its development, and this also raises some implementation questions. An important part of the GDIF hierarchy concerns representing gesture information in terms of the body of the performer, using the `/body` OSC prefix, and indeed several of our controllers already use this namespace. However, distinguishing performers using OSC address patterns proves much more complex when considering the various possible permutations of multiple performers and controllers.

8. FUTURE WORK

In addition to incremental improvements in function and usability, we have planned the addition of several new features:

Many-to-one mapping: As discussed above, we would like to implement the ability to negotiate many-to-one mapping relationships explicitly within the mapping interface, with simple GUI control over the desired combining function.

Vectors: Many OSC devices currently send or receive data in vectors or lists. The ability to split, combine, individually scale, and reorder vector elements will be added.

OSC pattern-matching: Pattern-matching annd wildcard functionality is defined in the OSC specification [12] but generally has not been fully implemented in OSC systems. It is easy to imagine scenarios in which using wildcards in mapped OSC address patterns would be a powerful addition to our system.

Editing namespaces: In the context of GDIF, the ability to alter parts of mapped namespaces without destroying the existing connections is also desirable.

Data rates: Rather than sending controller information as quickly as possible, we would like to make the data

rate a property of the mapping connection. A data stream might be used to control very slowly-evolving synthesis parameters, in which case very high data rates may be unnecessary and wasteful.

9. ACKNOWLEDGEMENTS

The authors would like to thank Alexander Refsum Jensenius for important discussion related to this effort, as well as the members of the Digital Orchestra Project group, especially Heather Hindman, Xenia Pestova, Chloé Dominguez, Fernando Rocha, D. Andrew Stewart, and Sean Ferguson. This project was supported by funds from the *Fonds de recherche sur la société et la culture* (FQRSC) of the Quebec government and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] The mcgill digital orchestra, 2007. Available: `http://www.music.mcgill.ca/musictech/DigitalOrchestra/`.

[2] Integra live, 2007. `http://integralive.org`.

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

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

[5] A. Hunt, M. Wanderley, and R. Kirk. Towards a model for instrumental mapping in expert musical interaction. In *Proceedings of the International Computer Music Conference*, San Fransisco, 2000. International Computer Music Association.

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

[7] T. Kvifte and A. R. Jensenius. Towards a coherent terminology and model of instrument description and design. In *Proceedings of the conference on New interfaces for musical expression*, pages 220–225, Paris, France, 2006. IRCAM – Centre Pompidou.

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

[9] M. Marshall, J. Malloch, and M. M. Wanderley. A framework for gesture control of spatialization. In *Paper to be presented at the 2007 International Gesture Workshop*, Lisbon, Portugal, 2007.

[10] 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.

[11] 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.

[12] M. Wright. Opensound control specification, 2002. Available: `http://www.cnmat.berkeley.edu/OSC/OSC-spec.html`.

[13] 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.