Libmapper
(A Library for Connecting Things)

Joseph Malloch
IDMIL, CIRMMT,
McGill University
Montreal, Canada
joseph.malloch@mail.mcgill.ca

Stephen Sinclair
IDMIL, CIRMMT,
McGill University
Montreal, Canada
stephen.sinclair@mail.mcgill.ca

Marcelo M. Wanderley
IDMIL, CIRMMT,
McGill University
Montreal, Canada
marcelo.wanderley@mail.mcgill.ca

Abstract
We present libmapper, a software library and protocol for providing network-enabled discovery and connectivity of real-time control signals. Today there is a trade-off present in the state of the art for music-related networking. At one extreme, we have many systems still using MIDI, an old and insufficient standard for specifying keyboard-oriented commands embedded in short, coded 3-byte messages, limiting modulation controls to a 7-bit range. At the other extreme we have Open Sound Control (OSC) [5], a flexible packet format that supports named data and a wide number of binary numerical representations, but lacks built-in semantic standards.

The present work proposes a semantic layer built on OSC over multicast UDP/IP used to carry metadata about signals, which can specify peer-to-peer connectivity between nodes along with instructions for associated translation of data representations. The translation layer avoids the need for normalization or standardization of data representation while maintaining ease of use and providing a distributed, flexible approach to music networking. The goal is to provide a system for fast and dynamic experimentation during the mapping phase of instrument design.
Author Keywords
network; control; communications; mapping; music

ACM Classification Keywords
H.5.5 [Information interfaces and presentation]: Sound and music computing; H.5.1 [Information interfaces and presentation]: Multimedia Information Systems.

General Terms
Music Mapping, Networking

Introduction
During the design of a digital musical instrument (DMI), it is often the case that the design of the mapping—the relationship between gesture and sound, i.e. between sensors and audio synthesis parameters—is left to the final stages of the project.

Yet, experience has shown that developing a satisfying mapping that is musically and gesturally interesting is a non-trivial exercise. Therefore, it is common to find that mapping is a highly iterative process, and is often also a collaborative one between the designer and a performer, composer, or other intended user of the device.

System design
In this work we present a software library, called libmapper. Our system takes the form of a metadata protocol based on Open Sound Control (OSC), which can be used to moderate connectivity between real-time OSC streams such as sensor outputs and synthesizer inputs.

We see in Figure 3 that a network node, called a device, is composed of several components which support the libmapper infrastructure. Two of these are internal: an admin handles the common metadata protocol, while a router maintains the list of links and connections, and handles translation of data representation from the sending device’s to the destination device’s expected format. In practice, user code need only deal with devices and signals.

A device is named by a string concatenated to a unique ordinal negotiated by the admin on a multicast UDP/IP bus. A signal identifies a data stream, and has information about the data’s type as well as range information and its measurement unit. Extra arbitrary user-specified metadata can also be provided for signals, connections, links and devices, for example to represent a device’s location in the room, or the name of the device owner.

Data transformation is specified by means of an arbitrary vector or scalar expression which can include logarithms, trigonometric functions, unit conversion, truncation, and type coercion between integer and floating point values. Past samples of input or output can be referenced, allowing specification of FIR and IIR filters.

Advantages of data translation
With libmapper, we emphasize the advantages of data translation over normalization or representation standards. These represent three approaches to the mapping problem.
The normalization approach applies scaling to all values such that they lie within a predetermined range. This makes it simple to connect “anything to anything.” The representation standards approach defines a set of agreed-upon semantics for control codes or message identifiers such that receivers can respond as expected to a given instruction packet.

With the translation approach, libmapper recognizes that both of these solutions depend on implicit metadata, information about data ranges and semantics, which can be more flexibly included in an explicit manner. That is to say, information about data ranges and message semantics is provided by libmapper nodes, and thus automatic translation from the sender’s representation to that expected by the destination is possible.

The advantage is that instrument designers are free to express outputs in meaningful terms, such as position in meters, or sensor readings in voltage, temperature, pressure, etc. Encouraging the use of physically meaningful units maintains information that is useful during the mapping stage of instrument design, instead of throwing it away a priori by shoehorning data into a predetermined format for compatibility reasons.

When range information is provided, the automatic translation approach allows single-click connections, just as in the normalization approach; additionally, however, mapping designers can further modify the generated expressions with specific tweaks such as changing the scale and offset, application of logarithmic or other non-linear relations, or even basic filtering.

Uses of libmapper
Several contexts in which libmapper has been used have inspired development of specific features to make it more generally useful for experimentation with a variety of mapping techniques.

Explicit mapping
The normal scenario for libmapper is explicit—manually making connections between device sensors and synthesizer parameters. Mappings may then be saved to a file for later modification or use. This procedure has been used for creating mappings on the fly while working with musicians, composers, and dancers in projects such as the McGill Digital Orchestra [2] and Gestes [1]. Both of these projects required extensive experimentation with mapping configurations during workshops, and libmapper allowed composers to take control over the mapping task while working directly with musicians, a role where a dedicated programmer has been needed in past projects.

Implicit mapping
We have extensively explored the use of machine learning algorithms to allow for implicit mapping. In this case a designer or musician puts the input device into a certain state and also configures the synthesizer to produce an accompanying sound. Several such configurations are saved in a “snapshot” database, and subsequently used to train a function approximator. For this purpose, libmapper supports querying the state of destination signals.

Another method for efficiently building an example database is to have some external process modify the synthesizer settings, and have the musician “play along” as the sound changes [3]. This kind of “continuous supervision” is supported in libmapper as a reversed connection from output to input.

Such methods have been used to map difficult-to-understand and possibly redundant sensor configurations such as found in the Ballagumi, a
More information is available at libmapper.org

- Source code
- API documentation
- Tutorials
- Pre-compiled objects for Max/MSP, Pure Data
- Development roadmap
- Mailing lists

Languages and environments currently supported:

- C, C++
- Python
- Java
- SuperCollider
- Max/MSP
- Pure Data

transparent and flexible instrument containing an array of specially-treated optical fibers to implement bend and pressure sensors [4].

Polyphonic mapping
There exist cases where one would ostensibly need to create and destroy signals on the fly. An example is polyphonic sound: one might wish to map the position of several fingers on a multitouch tablet to several voices, and free resources for re-use when a voice stops or a finger is lifted.

Multiple identical short-lived “connections” are better expressed as a single specification replicated over many possible instances. In libmapper, an optional hash embedded in each message allows the destination device to maintain a mapping to individual source instances. This feature has been used for mapping multitouch interaction and polyphony, as well as supporting multi-agent interactive systems that are distributed between network nodes.

Hardware and Software Support
Although we advocate the use of libmapper for providing compatibility between new software applications, we do not wish to enforce the use of particular software or hardware tools over others. It is essential to our concept of mapping that designers and users/performers are able to use whatever tools they prefer or are appropriate for the task at hand. For this reason, libmapper has been implemented as a low-level, cross-platform C library, and bindings are provided for popular programming languages and environments, including Python, Java, Pure Data, Max/MSP, and SuperCollider.

We do not consider our mapping tools to constitute a platform (implying a common software design strategy), but rather an ecosystem in which idiosyncratic approaches can communicate. Currently this ecosystem includes software bridges for the MIDI, HID, and Arduino/Firmata protocols, graphical user interfaces for supporting the mapping process, and libmapper-enabled software drivers for various popular input devices such as the Nintendo Wii Remote, joysticks, and Microsoft Kinect. Wherever possible we try to provide compatibility with existing tools.

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.

References