BRIDGES FOR NETWORKED MUSICAL ENSEMBLES

Lonce Wyse

Communications and New Media Programme
National University of Singapore

Norikazu Mitani

Arts and Creativity Lab, IDMI
National University of Singapore

ABSTRACT

Bridges is a software system designed to support composition and performance for networked musical ensembles. Bridges facilitates dynamic connections between a variety of devices, applications, and individual musical objects within multiple applications. The focus is on colocated performers, and on control data and communication structures rather than on audio delivery. It manages addressing and mapping between components making instrument design and networked compositions more object oriented, reconfigurable, and portable. Bridges is designed to support research in to new kinds of musical activity, particularly communication between performing musicians, mappings between interfaces and sound synthesis, and most importantly, dynamic network and control structures.

1. INTRODUCTION

Music is generally a social activity in which people come together to play at the same place and/or time. Ensemble musicians traditionally communicate with each other by sound, sight, and touch. Today, computer and communications technologies intervene at many stages between musicians, instruments, sound, and audience. On one hand, the "great acousmatic dislocations" of time, space, and mechanical causality [5] disrupt the traditional modes of communications between ensemble members, and are not necessarily overcome by live performance. On the other hand, access to these mediating chains of communications and control structures by composers and performers opens tremendous opportunities for new kinds of musical activity. At the center of all of these musical communication issues is the network.

1.1. The Network

In the late 1970's, the musicians who would form the League of Automatic Music Composers connected computers together in a network for live performance [2]. Today, networks are more user friendly, and the collection of tools for making music over networks continues to expand. Max/MSP [4], PD[7], ChucK[9], Supercollider[6], and others all have the basic capability to communicate through sockets and ports to other local or remote applications typically using the Open Sound Control (OSC)[11] messaging protocol.

However, the potential of networks could still be made more accessible for musical exploration. For example, IP addresses and port numbers of specific instruments or processes must be still be known by the composers and/or performers in order for communications to be established. One problem with this is that IP addresses change from machine to machine and from performance to performance. It would save composers time and provide new performance capabilities if musical objects could interact "directly" with each other obviating the need for musicians to worry about the network infrastructure details.

Another problem that arises at both composition and performance time is that different devices use different messaging protocols and conventions for parameter ranges. For example, MIDI uses 127 integers, a phone accelerometer may have an individual manufacturer-defined range, while software instruments might use different units such as kilometres-per-hour, pressure in dynes, etc. Composers must know prior to performance time what controls will be mapped to what receivers and spend time preparing the maps. This issue is a serious impediment to the ideal of permitting controllers or instruments to simply plug in to the network at performance time and begin exchanging musical information.

Here we describe Bridges - an application that hides unnecessary network infrastructure information from composers and performers, allows performers to dynamically make connections between objects on the network, performs automatic mapping to match sender/receiver parameter operating ranges, and provides access to information about, and interaction with, the whole network architecture. Our goal is to give the ensemble communications network itself "first class object" status so it that can be easily manipulated for musical purposes, or played like other instruments, while at the same time preserving direct communication between musical objects.

1.2. Related Work

There have been several applications designed to connect musical objects over the net that have some of the same

functionality as Bridges. NRCI [3] is an open-source suite of PD tools with common objects defined for interfaces, timing generators, controllers, synthesizers, and processors. It uses an OSC-based all-client (no central server) broadcast messaging protocol. Clients can request or offer specific types (pitch, amplitude, rhythmic) of data, but for sending data, message formats need to be agreed upon in advance or negotiated at performance time.

Although much broader in scope, the Diamouses [1] project has some similarities with Bridges, particularly in that it uses a hybrid of coordinated information and peer-to-peer strategies.

Osculator [8] contains many of the mapping features that Bridges contains, but it is OSX-specific, and does not have the flexibility for real-time network reconfiguration that the Bridges is designed for.

2. COMPONENTS OF BRIDGES

A representational hierarchy defines relationships between important components in the system (Figure 1).

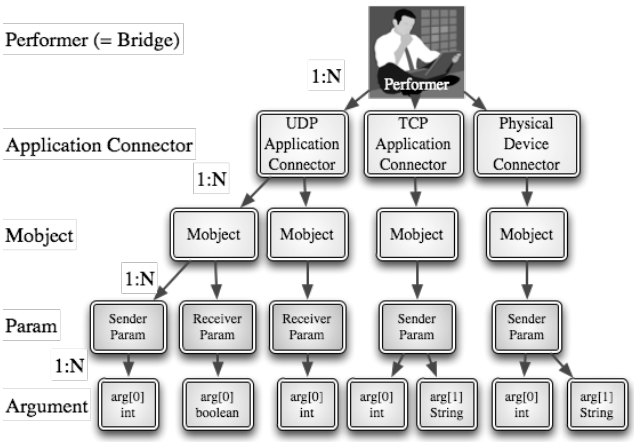

Figure 1. Logical hierarchical structure of components relevant to the Bridge.

A "performer" is a musical human being, and there can be zero or more performers per machine on the network, each interacting with some subset of audio/video applications and physical devices connected on a machine.

The next layer is occupied by Application Connectors which communicate information with the Bridge about application objects, and define the port and connection type (e.g. UDP,TCP, or MIDI) that will be used for communication. There can be many applications, and thus application connections, used by a single performer.

At the next level we have the "mobject" (musical object). A "mobject" represents a synthesis algorithm, instrument, a virtual or physical controller, a sound transformer, or even a graphical structure – anything capable of sending or receiving control signals as part of a musical environment. Every application connector can have multiple mobjects.

At the forth level is the "parameter". Parameters represent the inputs and output that mobjects make available for interaction. Every mobject can have multiple input or output parameters.

At the fifth level are parameter "arguments". Some audio/video application or devices will send multiple data at one time. A single "parameter" can be multidimensional, and thus have multiple arguments for sending or receiving multidimensional musical control data.

A Bridge itself is run one per machine. Mobjects, via parameter arguments, exchange control signals with each other both locally and on networked machines (Figure 2).

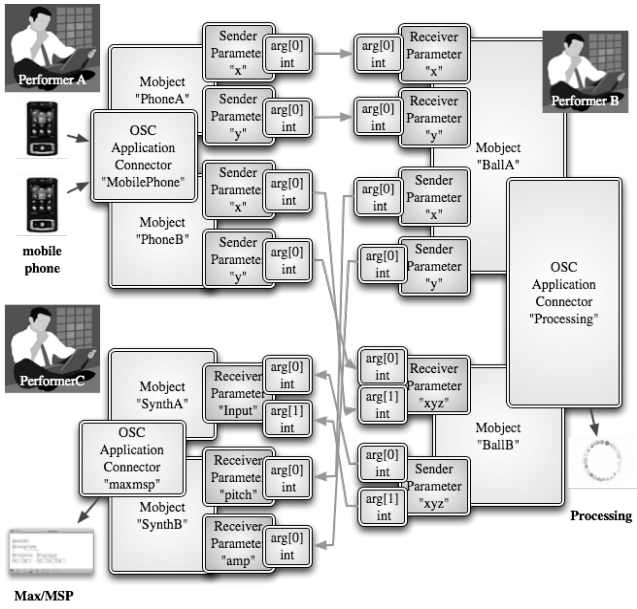

Figure 2. Example network consisting of three Bridges.

A Bridge engages in two types of communications with the clients on its local machine. One mode is for configuration where the client informs a Bridge (and thus the networked world) what objects it has available for sending and/or receiving musical controller data. The other mode is for sending streams of musical controller data which get mapped and routed in a peer-to-peer fashion with other clients on the network (via their own local Bridges). The two modes are engaged simultaneously during performance, but they contain different kinds of information, and use different ports for communication:

1) Configuration commands (via GUI or messaging)
   a. register mobjects and mapping info,
   b. make connections between mobjects.
2) Music controller data exchange (via messaging)
   a. exchange data between mobjects.

2.1. Configuration Commands

The configuration command mode is where the Bridges application demonstrates its unique capabilities since this is the mode used to modify the communications architecture. Application connector configurations, mobject names, parameter attributes, argument data ranges, and connections between sender and receiver arguments are all specified using the configuration mode.

Bridges provides two equivalent ways for specifying configurations. One is through a graphical user interface (GUI) and the other is through a messaging protocol. Both methods provide the same configuration functionality. The GUI is necessary primarily for dumb object providers such as MIDI devices that are unable to specify metadata about themselves or network architectures. The messaging protocol is most useful with programmable applications and when the network structure is dynamic during performance. Configuration messages can be controlled programmatically so that the network architecture can be manipulated as fluidly as any other musical components.

Two messaging schemes are supported for the configuration mode: OSC (via UDP) and TCP (under development for communication with Adobe Flash applications).

2.2. Music Controller Data Exchange

The second mode of communication is for the exchange of controller data to and from mobjects via parameter arguments. The role of a Bridge is to route and map the messages between mobjects according to the sender and receiver connections that have been configured via the GUI or configuration messages. For music controller data, three protocols are supported: OSC (via UDP), TCP sockets, and MIDI. When connections are made between mobjects in applications that use different protocols, Bridges automatically translates from one to the other. It also automatically maps between the different numerical ranges that the various senders and receivers have registered.

Each application that uses Bridges to manage communications only has to know ports, protocols, and parameter ranges of its own components that it registers with a Bridge. Connections to networked components are then done only by name, while Bridges manages the infrastructure details. In this way, a mobile phone accelerometer registered as an OSC-formatted data sender using the range of [-64,63] can connect to an receiver object registered with a Bridge as a MIDI device by name only, and not worry about ports, message protocols or parameter ranges.

2.2.1. MIDI messages

The legacy structure of MIDI messages requires a little special attention so that it fits into the GUI and mapping structures of Bridges. This is because the messages contain several fields that can each be interpreted either as controller data to be routed, or as information that should determine how other fields in the message should be routed. The composer or performer can thus ether chose to filter the messages based on data in a field, or get the data in a field for routing. If they chose to filter it, they chose the range of data for that field that will permit the other parts of the message to be routed. If they want to rout the data in a particular field, the field is connected to am mobject receiver parameter argument like any other. (Figure 3). In this way incoming MIDI data can be routed depending on channel or controller number, and even different note numbers could be filtered so that the velocity data in the same message could be sent to different receiver objects.

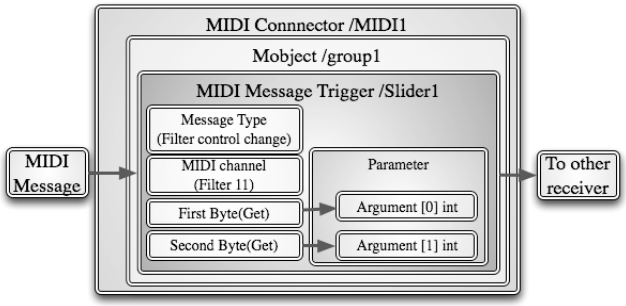

Figure 3. MIDI message contain several fields of data that can be used to filter the message, or can be used as data to be routed.

Because of the GUI for creating the architecture, and the automatic mapping between ranges and messaging protocols, Bridges is a useful stand-alone tool for connecting arbitrary devices and applications running or connected to a single machine. However, other tools (e.g. Osculator) already provide some of these capabilities. The most important musical potential of Bridges lies in its ability to work in a network of Bridges.

2.3. Bridges Networks

In the local subnet version of Bridges, changes made to the architecture at any of the individual Bridges are shared with all the others via a UDP broadcast message. In this way, all Bridges have the same information about the network architecture. This information includes host, port, object and parameter names and ranges for all available objects - everything necessary for the Bridges to establish and execute controller data exchange which is carried out peer-to-peer with specifically addressed UDP messages. This hybrid broadcast plus peer-to-peer strategy is well suited for the two kinds of information that needs to be communicated – configuration data that is coordinated across machines, and control data between mobject parameter arguments.

2.4. Playing the Network

We designed Bridges to support research and creative explorations in the musical use of networks. We were inspired by the early Hub experiments with various architectures, connection patterns, and social/control relationships between ensemble members. All performers have GUI and programmatic access to change any part of the network configuration at all times, not just the connections that involve their own local mobjects. This unrestricted reconfiguration capability allows for the widest variety of strategies and network topologies [10] to be implemented.

Connection messages can also be grouped as "scenes" which represent the entire network connection structure. Any Bridge can capture or send scenes at any time. This lends itself to interesting improvisational possibilities as well as compositional structuring. For example, the ensemble could navigate through a sequence of musical sections each embedded in a different network architecture.

Because of the management of infrastructure details, new ensemble members with their own client machine, and previously unknown applications or devices, can also join and leave the network at any time.

2.4.1. Visualization

Network visualization is an extremely important role Bridges plays beyond the interface it provides. If the object names and connections displayed in the GUI are inadequate for a given musical goal, a Bridge is also capable of responding to requests from local applications to provide them with a snapshot of the network. This is provided so that further research can be conducted into representations and interfaces for fluid musical networks where new kinds of communication between ensemble members are needed.

3. SUMMARY

We have developed an application that facilitates dynamic performance-time connections between devices and applications whether connected locally or on a network. Bridges manage infrastructure details, automatically map across messaging protocols and numerical ranges, and provide both a graphical and a programmatic means for getting information about the network as well as for manipulating its structure. The tool makes it easy for composers and ensemble performers, programmers and nonprogrammers, to explore the potential of the networks for new kinds of music making.

The primary reason we developed Bridges is to enable access to the "web of mutual influence [2], so that all kinds of communications and control strategies between and among instruments, algorithms, and musicians, can by explored. Thus, visualization and representation for ensemble performance are key areas for further exploration. We are particularly interested in these issues given the temporal, spatial, and causal dislocations that can result from arbitrary chains of mediation between musical gesture, an issue compounded by the dynamic network structures Bridges supports.

We view this as an early-stage development in a dialog with composers and performers. Bridges is available at artsandcreativitylab.org.

4. REFERENCES

[1] Alexandraki,C, Koutlemanis, P, Gasteratos, P, Valsamakis, N., Akoumianakis, D., Milolidakis G. "Towards the implementation of a generic platform for networked music performance: The DIAMOUSES approach", in *Proceedings,, International Computer Music Conference* Belfast, Northern Ireland, 2008.

[2] Brown, C., and Bischoff, J. "Indigenous to the Net: Early Network Music Bands in the San Francisco Bay Area", 2002. http://crossfade.walkerart.org/brownbischoff/

[3] Burns, C., and Surges, G. "NRCI: Software Tools for Laptop Ensemble." *Proceedings, International Computer Music Conference* Belfast, Northern Ireland, 2008.

[4] Cycling '74, *Max/MSP*. 379A Clementina Street, San Francisco, CA 94103.

[5] Emmerson, S., "'Live' versus 'real-time'", *Contemporary Music Review*, 1994, 10(2), 95-101.

[6] McCartney, J. "Rethinking the computer music language: SuperCollider", *Computer Music Journal*, 26, 2002, pp. 61–68.

[7] Puckett, M. *Pure Data*. http://puredata.info

[8] Troillard, C. *Osculator*. http://www.osculator.net

[9] Wang, G. and and Cook. P. "ChucK: A Concurrent, On-the-fly, Audio Programming Language", in *Proceedings, International Computer Music Conference*, Singapore, 2003.

[10] Weinberg, Gil. "Interconnected Musical Networkes: Toward a Theoretical Framework", *Computer Music Journal*, 29, 2005, pp. 23-29.

[11] Wright, M., and Freed, A. "Open Sound Control: a new protocol for communicating with sound synthesizers", *Proceedings of the International Computer Music Conference*, Thessaloniki, Greece, 1997.