http://cnmat.berkeley.edu/publications/open-sound-control-state-art-2003

OpenSound Control: State of the Art 2003

Matthew Wright
Center for New Music and Audio Technology (CNMAT)
Univ. California, Berkeley
1750 Arch St., Berkeley, CA 94709
(1) 510 643-9990
matt@cnmat.berkeley.edu

Adrian Freed
CNMAT
Univ. California, Berkeley
1750 Arch St., Berkeley, CA 94709
(1) 510 643-9990
adrian@cnmat.berkeley.edu

Ali Momeni
CNMAT
Univ. California, Berkeley
1750 Arch St., Berkeley, CA 94709
(1) 510 643-9990
ali@cnmat.berkeley.edu

ABSTRACT

OpenSound Control ("OSC") is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology. OSC has achieved wide use in the field of computer-based new interfaces for musical expression for wide-area and local-area networked distributed music systems, inter-process communication, and even within a single application.

Keywords

OpenSound Control, Networking, client/server communication

1. TUTORIAL OVERVIEW of OSC

This is a user-level overview of OSC. For more technical details such as exact semantics and the binary format of OSC messages, please see the OSC specification [33].

1.1 Client/Server

OSC is designed to support a client/server architecture. OSC data is transmitted in data units called *packets*. Anything that sends OSC packets (e.g., an application, physical device, subprogram, etc.) is a *client* and anything that receives OSC packets is a *server*.

OSC is a transport-independent, high-level application protocol; in other words, OSC does not specify what low-level networking mechanism will be used to move OSC packets from the client to the server.

1.2 Messages

The basic unit of OSC data is a *message*, consisting of an *address pattern,* a *type tag string,* and *arguments*. The address pattern is a string that specifies the entity or entities within the OSC server to which the message is directed (within the "Addressing Scheme" described below) as well as what kind of message it is. The type tag string gives the data type of each argument. The arguments are the data contained in the message.

For example, a message's address pattern might be /voice/3/freq, its type string might indicate that there is a single floating-point argument, and the argument might be 261.62558.

1.3 Argument Data Types

Each message contains a sequence of zero or more arguments. The official OSC data types are ASCII strings, 32-bit floating point and integer numbers, and "blobs," chunks of arbitrary binary data. OSC's type mechanism allows for many other types, including 64-bit numbers, RGBA color, "True," and "False." Only a few implementations support these other types, but they all represent them in a standard way.

1.4 Addressing Scheme

All of the points of control of an OSC server are organized into a tree-structured hierarchy called the server's *address space*. Each node of the address space has a symbolic name and is a potential destination for OSC messages.

Each OSC server defines its own address space according to the features it provides and the implementor's idea of how these features should be organized. This is in contrast to protocols such as ZIPI [22] and MIDI that attempt to define in advance what the architecture of a synthesizer should be and what kinds of messages can be sent to it.

An OSC *address* is simply the full path from the root of the address space tree to a particular node, with a slash-delimited format like a URL or file system pathname. For example, the address "/voices/3/freq" refers to a node named "freq" that is a child of a node named "3" that is a child of a top-level node named "voices."

An OSC server's address space may change dynamically, therefore OSC's query system (described below) includes a mechanism for discovering the current address space.

1.5 Address Pattern Matching

Remember that each OSC message contains an OSC address *pattern*, not an OSC address. An OSC address pattern is exactly like an OSC address, except that it may contain special characters for regular expression [10] pattern matching. When an message's address pattern matches more than one of the addresses in the server's address space, the effect is the same as if there were individual messages (all with the same arguments) sent to each of the matched addresses.

The special characters are '?', '*', a string of characters inside '[brackets]', and a comma-delimited list of strings inside '{curly,braces}'. They work like Unix shell filename globbing.

1.6 Bundles and Temporal Atomicity

A *bundle* is a sequence of messages and/or bundles. This recursive definition allows for arbitrary nesting of sub-bundles. All of the messages in the same bundle must be processed by the OSC server atomically; in other words it should be as if all of the messages in the bundle are processed in a single instant. An OSC packet may be a bundle or a (single) message.

1.7 Time Tags

Each bundle has a *time tag* that specifies the desired absolute time at which the messages in the bundle should take effect. The format is that used by the Internet Network Time Protocol [23] and provides sub-nanosecond accuracy over a range of over 100 years. OSC currently relies on an outside mechanism to synchronize clocks on different machines to the same absolute time, for example, NTP [23] or SNTP [24].

1.8 Queries

Queries are OSC messages that request the server to send information back to the client. Example queries include "what is the current list of nodes under this given node?", "what argument types are expected for messages sent to this given node?", "what is the value of the parameter that can be set by sending messages to this node?", and "please send me some documentation for the object or feature specified by this address."

2. IMPLEMENTATIONS OF OSC

CNMAT created OSC and maintains a web site, downloadable code, and developers' email list. As the public face of OSC we often hear about other people's use of OSC; this section lists the implementations and uses of OSC of which we are aware. Since the standard is open and our code is freely downloadable, we assume that there are other implementations of which we are not aware; we look forward to hearing about them. No doubt some of the details described in this section will be obsolete by the time this paper goes to press, especially, we hope, some of the limitations of certain systems.

All of the implementations mentioned in this section are linked from the main OSC home page at CNMAT (http://www.cnmat.berkeley.edu/OSC).

2.1 CNMAT's Open-Source OSC Software

All of the software mentioned in this section is available for download from CNMAT (http://www.cnmat.berkeley.edu/OSC).

When we introduced OSC in 1997, we released *OSC-client.c*, a C library for constructing OSC packets through a procedure call interface. There is nothing more to implementing an OSC client than constructing proper OSC packets and sending them over the network.

We also released a pair of text-based Unix command-line utilities: *sendOSC* allows the user to type in message addresses and arguments via a no-frills text interface, and formats and sends these messages via UDP to the desired IP address and port number; *dumpOSC* listens for OSC messages on a given UDP port and prints them out in a simple ASCII format.

As part of CNMAT's early efforts to promote OSC, we released the *OSC Kit* [32] in source code form in 1998. Our reasoning was that although the community as a whole was in favor of OSC and its features (as they had been of ZIPI), people would be reluctant to implement OSC (as they had been of ZIPI) unless we did a lot of the work for them (which we did not do for ZIPI). Thus, the OSC Kit implements most of the features needed for an OSC server: (dynamic) construction of an address space, parsing OSC packets, pattern-matching address patterns within the address space, associating a user-defined callback procedure with each node of the address space and invoking that procedure in response to messages sent to that node, and even a scheduler for implementing time tags. The OSC Kit is completely neutral to architecture and operating system and was designed not to degrade reactive real-time performance.

2.2 Music Programming Environments

All of the current OSC implementations known to the authors send and receive OSC packets only as UDP packets.

2.2.1 Max/MSP

The first programming environment to implement OSC was *Max/MSP* [28, 36], in the form of Max "externals" written by Matt Wright. The *OpenSoundControl* external translates in both directions between native Max data lists and OSC-formatted binary data. The *otudp* external (as well as the now-obsolete *udp* external) sends and receives arbitrary UDP packets and can be used in conjunction with the *OpenSoundControl* object. These are implemented as separate objects to allow for transmission of OSC packets other than by UDP packets and to allow for transmission of UDP packets other than OSC packets. Finally, the *OSC-route* external enables the parsing of OSC address patterns by Max programmers and implements OSC's pattern matching features. All of these objects have been ported to the OSX version of Max/MSP.

The Max/MSP implementation has full support for sending and receiving messages and bundles, but there is currently no integration between OSC time tags and Max's scheduler and no support for queries. There is backwards-compatible support for both sending and receiving non-type-tagged messages. Temporal atomicity of bundles is handled by the fact that *OpenSoundControl* outputs a "bang" after outputting all of the messages in a bundle; it is the responsibility of the Max/MSP programmer to ensure that all of the messages take effect atomically when the "bang" is output.

2.2.2 SuperCollider

James McCartney added OSC support to the object-oriented *SuperCollider* ("SC") language and environment [21] in 1998. The *OSCNode* object represents a node of the OSC address space and contains a symbolic name, a list of the children of the node, and a function to be called when the node receives a message. The *OSCOutPort* and *OSCInPort* objects represent UDP ports that can send or receive (respectively) OSC packets. Every *OSCInPort* has an *OSCNode* that is the root of the address space associated with that port.

There is a large sub-tree of OSC messages that can be sent to the SC environment itself, including "run the main patch," "stop synthesis," "play this sound file from the local disk," and even "compile and execute the code in the string argument to this message." There is also an OSC representation for all of the important MIDI messages (note on/off, continuous controllers, pitch bend, program change, channel and key pressure, and all-notes-off); when SC receives one of these OSC messages it's exactly as if SC had received the corresponding MIDI message via MIDI input.

Version 3 of SC, only for OSX, has a completely new architecture and is called *SuperCollider Server*. In this version, the synthesis engine is a separate application from the SC language and programming environment; the two communicate exclusively with OSC messages via UDP or TCP. This allows the SC synthesis engine to be controlled by applications other than the SC language.

2.2.3 Open Sound World

Open Sound World (OSW) [5] is a scalable, extensible object-oriented language that allows sound designers and musicians to process sound in response to expressive real-time control. OSW has the same graphical dataflow model and nested subpatch structure as the Max family of languages; one important difference is that every OSW object has a symbolic name. Thus, the objects in an OSW patch automatically form an OSC-style hierarchical address space and can thus easily be addressed with OSC messages; the OSW kernel handles this automatically. OSW also provides an object called *OSCListen* that can be used to process incoming OSC messages manually; this allows OSW programmers to construct an OSC address space that does not necessarily reflect the tree structure of the OSW program that is the OSC server.

OSW has the best support of OSC queries of any implementation known to the authors, thanks in large part to recent work by Andy Schmeder at CNMAT. The dynamic

address space of an OSW program can be discovered by a querying client. Any message that can be understood by any of the objects in an OSW patch can be sent to that object via OSC. An OSC client can get the current value of any variable of any OSW object.

OSW fully supports type tags. There is currently no connection between OSC time tags and OSW's notion of the current time. A careful programmer can use OSW's "synchronous outlets" mechanism to ensure that all elements within a bundle will be processed atomically.

2.2.4 Pd

OSC support in the *Pd* programming language and environment [29] is in the form of third-party objects. The *sendOSC* and *dumpOSC* objects are for sending and receiving OSC packets and are derived from CNMAT's text-based Unix command-line utilities of the same name.

The *sendOSC* object must be set to write to a given IP address and UDP port. Then it translates Pd lists to properly-formatted OSC messages and sends them. There is also support for creating bundles, but not for specifying bundles' time tags.

The *dumpOSC* object creates a UDP socket, parses incoming OSC packets on that port, converts each OSC message to a Pd list, and outputs the lists sequentially. Time tags are ignored. There is no mechanism to assist with temporal atomicity of bundles; in fact, no representation of the bundle structure of incoming OSC packets is available to the Pd programmer—consecutive lists output by *dumpOSC* might be from the same bundle or from different OSC packets entirely.

The *routeOSC* object is derived from and practically identical to the Max/MSP *OSC-route* object; it supports the parsing of address patterns with pattern matching.

Pd does not currently support queries.

2.2.5 Virtual Sound Server

NCSA's *Virtual Sound Server* ("VSS") [2] is an environment for real-time interactive sound synthesis; it is designed to be used in conjunction with graphics rendering software and includes mechanisms for synchronization of its audio with other applications' video. VSS can be controlled with a limited form of OSC utilizing a flat address space. Type tag strings, pattern matching, bundles, time tags and queries are not supported.

2.2.6 Csound

Csound support for OSC currently exists only as part of the "unofficial" release (http://web.tiscali.it/mupuxeddu/csound). This implementation is based on the OSC Kit and allows users to define Csound orchestras that can be controlled by OSC. The Csound programmer can name elements of the OSC address space, but the overall tree structure of the OSC address space is constrained by the fact that all Csound instruments are at the same level in a flat namespace. A procedure called at the K-rate checks for and processes newly-received OSC messages.

2.3 Software Synthesizers

Grainwave [3] is a software granular synthesizer with very limited OSC support. It accepts MIDI messages formatted as OSC messages; the use of OSC is solely as a mechanism to transmit MIDI-style data over the Internet.

Native Instruments' *Reaktor* (www.native-instruments.com) is a general-purpose environment for building software synthesizers. Reaktor's OSC support in version 3 is similar to Grainwave's, essentially just MIDI over the Internet, but version 4, currently still in beta, is said to have much more integrated OSC support.

2.4 General Purpose Programming Languages

All of the implementations described in this section are available in source code form via CNMAT's OSC home page.

Chandrasekhar Ramakrishnan has implemented Java classes that can create OSC packets via a procedural interface and send them in UDP packets [30]. It supports type tags but not time tags. Future plans include the ability to receive OSC.

C. Ramakrishnan has also built an Objective-C wrapper around OSC-Client.c, designed primarily to allow Cocoa applications to send OSC messages.

There is an implementation of OSC in Perl (http://barely.a.live.fm/pd/OSC/perl). The sending half is implemented by a Perl wrapper around the *sendOSC* program that was created automatically with the *SWIG* interface compiler (http://www.swig.org). The receiving half is a port of the *dumpOSC* program to Perl; it provides a function called *ParseOSC* that takes in a binary OSC packet (such as data received via Perl's built-in UDP implementation) and returns the address and arguments of an OSC message.

There are two OSC implementations for Python; unfortunately both are Python source files with the name "OSC.py". Daniel Holth's and Clinton McChesney's *pyOSC*, part of the *ProctoLogic* project (http://galatea.stetson.edu/~ProctoLogic), translates bidirectionally between the binary OSC format and Python data types. Bundles are read correctly but cannot be constructed. It also includes a *CallbackManager* that allows a Python programmer to associate Python callbacks with OSC addresses and then dispatches incoming OSC messages. Unfortunately pattern matching is not yet implemented. ProctoLogic is covered by the LGPL.

Stefan Kersten's *OSC.py* is a Python module for OSC clients (http://user.cs.tu-berlin.de/~kerstens/pub/OSC.py). It can construct arbitrary OSC packets and send them in UDP packets, and can even produce OSC time tags based on Python's built-in time procedures.

Smalltalk also has two implementations of OSC. *VWOSC* (http://www.mat.ucsb.edu/~c.ramakr/illposed/vwosc.html) was written by C. Ramakrishnan and Stephen Pope and currently only send OSC. The *Siren* Music and Sound Package for Squeak Smalltalk (http://www.create.ucsb.edu/Siren) includes an experimental OSC implementation.

2.5 Web Graphics Systems

Macromedia's *Flash* displays web application front-ends, interactive web site user interfaces, and short-form to long-form animation. It contains a scripting language called *ActionScript* that has good support for manipulating XML documents as well as a mechanism for sending and receiving streamed XML documents via a TCP/IP socket. Ben Chun has defined an XML document type to represent OSC packets in XML and created a bidirectional gateway between Flash and OSC with a program called *flosc* [6] that translates between OSC packets over UDP and XML documents over TCP.

As a multimedia authoring tool designed to create rich interactive content for both fixed media and the Internet, Macromedia's *Director* can incorporate photo-quality images, full-screen or long-form digital video, sounds, animation, 3D models, text, hypertext, bitmaps, and Macromedia Flash content. Garry Kling at UCSB has written an extension ("xtra") to Director called OSCar [18] that can send OSC packets from

the *Lingo* scripting language. Future plans include the ability for Lingo to receive OSC.

2.6 Gesture-to-OSC Hardware

The *Kroonde* [19] is a system for receiving data from wireless sensors, for example, pressure, flexion, acceleration, magnetic field, and light sensors. The Kroonde receiver takes in data from up to 16 of these sensors and converts them either to MIDI or to OSC over UDP over 10 BaseT Ethernet.

Dan Overholt has built an interface called the *MATRIX* ("Multipurpose Array of Tactile Rods for Interactive eXpression") that consists of a 12x12 array of spring-mounted rods each able to move vertically. An FPGA samples the 144 rod positions at 30 Hz and transmits them serially to a PC that converts the sensor data to OSC messages [25, 26].

Newton Armstrong at Princeton has built prototype hardware (http://music.princeton.edu/~newton/controller.html) with 11 continuous and 40 switch analog inputs, which are digitized, converted to OSC messages, and sent as UDP packets via a built-in 10BaseT Ethernet port.

There are plans for the next version of IRCAM's *AtoMIC Pro* gesture-acquisition hardware [9] to output OSC rather than MIDI as it does now.

3. OSC-based Networking Applications

Here is a somewhat chronological survey of networked music applications that have been built with OSC. It is certainly not comprehensive; we encourage all users of OSC to inform us about their projects.

At ICMC 2000 in Berlin (http://www.audiosynth.com/icmc2k), a network of about 12 Macintoshes running SuperCollider synthesized sound and changed each others' parameters via OSC, inspired by David Tudor's composition "Rainforest."

The *Meta-Orchestra* project [16] is a large local-area network that uses OSC.

In Randall Packer's, Steve Bradley's, and John Young's "collaborative intermedia work" *Telemusic #1* [35], visitors to a web site interact with Flash controls that affect sound synthesis in a single physical location. The resulting sound is streamed back to the web users via RealAudio. This system was implemented before *flosc* and before Flash's *XMLSockets* feature existed, so data goes from Flash to JavaScript to Java to OSC.

In a project [17] at the MIT Media Lab, the analyzed pitch, loudness, and timbre of a real-time input signal control sinusoids+noise additive synthesis. The mapping is based on Cluster-Weighted Modeling and requires extensive offline analysis and modeling of a collection of sounds. In one implementation, one machine performs the real-time analysis and sends the control parameters over OSC to a second machine performing the synthesis.

Three projects at UIUC are based on systems consisting of real-time 3D spatial tracking of a physical object, processed by one processor that sends OSC to a Macintosh running Max/MSP for sound synthesis and processing.. In the *eviolin* project [13], a Linux machine tracks the spatial position of an electric violin and maps the spatial parameters to control a resonance model in real-time. The sound output of the violin is processed through this resonance model. In the *Interactive Virtual Ensemble* project [12], a conductor wears wireless magnetic sensors that send 3D position and orientation data at 100 Hz to a wireless receiver connected to an SGI Onyx that processes the sensor data. In this system, the Max/MSP software polls the SGI via OSC to get the current sensor values. *VirtualScore* is an immersive audiovisual environment for creating 3D graphical representations of musical material over time [11]. It uses a CAVE to render 3D graphics and to receive orientation and location information from a wand and a head tracker. Both real-time gestures from the wand and stored gestures from the "score" go via OSC to the synthesis server.

Stanford's CCRMA's *Circular Optical Object Locator* [14] is based on a rotating platter upon which users place opaque objects. A digital video camera observes the platter and custom image-processing software outputs data based on the speed of rotation, the positions of the objects, etc. A separate computer running Max/MSP receives this information via OSC and synthesizes sound.

Listening Post [15] is a networked multimedia art installation based on representing conversations in Internet chat rooms on a large number of video monitors and also with sonification via a 10-channel speaker system. A local network of 4 computers handle text display, text-to-speech, sound synthesis, and coordination of all these elements; all of the components of the system communicate with OSC. *Listening Post* is currently on display at the Whitney museum of American Art.

A research group at UCSB's CREATE has been developing "high-performance distributed multimedia" and "distributed sensing, computation, and presentation" systems [27]. These large-scale networks typically consist of multiple sensors such as VR head-trackers and hand-trackers, dozens of computers interpreting input, running simulations, and rendering audio and video, and multichannel audio and video output, all connected with CORBA and OSC. Another UCSB project [8] combines CORBA and OSC to allow a VR system with data gloves and motion trackers to send control messages to synthesis software written in SuperCollider.

In the *Tgarden* project [31], wireless accelerometers are sensed by a Linux machine and mapped via OSC to control sound and video synthesis in Max/MSP, SuperCollider, and NATO.

4. OSC Pedagogy

University courses teaching OSC include the following:

- Iowa State Music 448 ("Computer Music Synthesis")

- Princeton COS436 ("Human Computer Interface Technology")

- Stanford Music 250a ("HCI Seminar")

- UC Berkeley Music 158 ("Musical Applications of Computers and Related Technologies") and 209 ("Advanced Topics in Computer Music") and CNMAT's summer Max/MSP Night School.

- UC Santa Barbara Music 106 ("Interactive Electronic Performance and Synthesizer Design Using Max/MSP")

5. BENEFITS OF ORGANIZING REAL-TIME MUSIC SOFTWARE WITH OSC

This section describes some programming techniques that make use of OSC as the primary organizational scheme for building real-time performance instruments. Although our examples concentrate on the Max/MSP programming environment, the described techniques can be generalized to other platforms and aim in general to improve modularity and interconnectivity of software components.

5.1 A Module's OSC Namespace Is Its Entire Functionality

We propose a style of programming in which the entire functionality of each software module is addressable through OSC messages. Advantages of this style include the following:

1. The OSC namespace for each module explicitly names all of that module's features. This can enable software to be self-documenting and transparent in its functionality.
2. The entire functionality of each module is accessible via a single control mechanism: incoming OSC messages. In graphical languages such as Max/MSP and Pd, this allows even the most complex objects to have a single control inlet, reducing the clutter and confusion of connecting to multiple inlets (Figure 1). As a module's functionality grows, no structural changes (such as adding more inlets) are required; the programmer simply expands the module's OSC namespace
3. If the components of a complex system already communicate among themselves exclusively with OSC messages, then it becomes very easy to move some of the components to other computers to form a distributed local area network system.
4. Certain OSC messages can be standardized across different modules. For example, the message "/gain" with a floating point argument can be used in many different synthesis and processing components to change gain; the message "/namespace" can trigger any module to display its OSC namespace; the message "/go" followed by the argument 1 or 0 can be used to turn on and off the processing in the module; and the message "/init" can initialize a module.

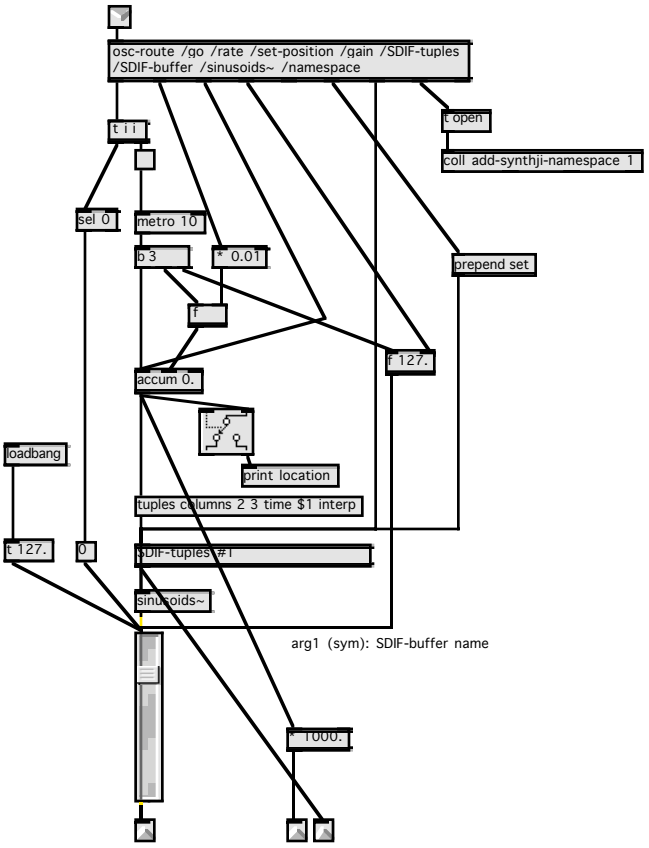

Figure 1: A Max/MSP patch that performs additive synthesis in real-time using Sinusoidal Track data stored in an SDIF-buffer. Everything the patch does is accessible through a single inlet and is described by the list of OSC messages that the patch understands.

By sending the "/namespace" message to the patch in figure 1 the user is presented with this list of OSC messages and can quickly learn how to control the patch:

1, /go _int_ turn processing on and off;
2, /rate _float_ play rate;
3, /set-position _float_ between 0 and 1 sets position in buffer from start to end;
4, /gain _float_ sets gain;
5, /SDIF-tuples _anything_ talks to SDIF-tuples;
6, /SDIF-buffer _symbol_ sets SDIF-buffer for playback;
7, /sinusoids~ _anything_ talks to sinusoids~;
8, /namespace _bang_ opens this collection;

5.2 Storing and Recalling Global Snapshots of Complex Software Components with OSC

In developing complex software instruments that perform many processes with many possible arrangements of parameters, the task of storing and retrieving complete snapshots of the system's state can be quite challenging. We propose a system of performing this storage and recalling that is based entirely on the usage of OSC messages as the communication scheme between the instrument's snapshot mechanism and its constituent modular components.

Once an instrument comprised of a set of components—all of whose functionality is addressable with OSC messages—is developed, it is possible to store and recall global settings of the entire system by collecting and dispensing OSC messages from and to the individual components. We propose a model where each module keeps track of its current state by remembering what OSC messages have been sent to it most recently. Note that since the OSC name for each function of the module is unique, this can be achieved by using the OSC message as an index whose value is replaced each time a new value is received. In order to collect a snapshot, we query each component for its current state. Each component answers the query in the form of a list of OSC message that will bring it back to its current state if sent to it at a later time. OSC messages from each component are then collected and stored in one central location. In order to recall a stored global snapshot of the system, one simply has to send out the list of OSC messages that each component submitted earlier.

This method was successfully employed in developing Ron Smith's work for orchestra and live electronics titled *Constellation* [20], as well as the collaborative dance piece of Carol Murota, Edmund Campion and Ali Momeni entitled *Persistent Vision* [4], a work for 16 dancers and live interactive sound installation.

5.3 Managing Polyphony With OSC

Many of the platforms for developing specialized real-time audio/video software include some tools for building polyphony, e.g., Max/MSP's *poly~* object and Pd's *nqpoly~* object. By using a simple abstraction that routes OSC messages to specific voices of a polyphonic component (figure 2), OSC's pattern-matching features can be used to address specific voices or sets of voices with great ease.

5.4 Dynamic Routing of Controller Data with OSC

We continue to advocate the use of OSC as the bridge between input data streams from gestural controllers and signal processing engines. This style of programming involves describing a complete OSC namespace for all output streams from a controller [34]. Intermediary patches dynamically map the control data to the OSC namespace for a

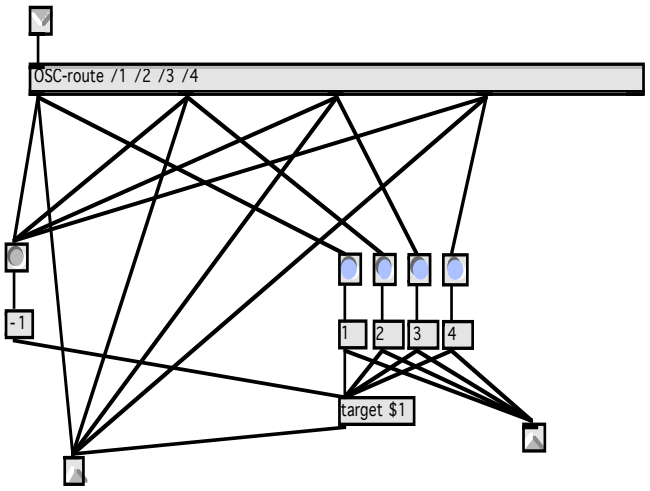

Figure 2: A Max/MSP patch that routes OSC messages starting with the desired voice number to that voice of a 'poly~' object (which uses the 'target' message to address specific voices). The patch was created with Max/MSP's scripting mechanism and could have been made with any number of voices.

specific signal processing engine. This allows the performer to change instruments by switching from one intermediary-mapping-patch to another, thereby directing his controller's OSC output to a different signal processing module.

In the past year, we have further developed our implementations of this approach to gesture mapping for a number of controllers including the Buchla Thunder, Wacom drawing tablets (via a new interface using Cycling 74's Jitter), and the game controller Cyborg 3D made by Saitek.

Finally, we promote the use of OSC for designing controller data streams that are *modal*. For example, the Saitek Cyborg 3D joystick provides an extremely flexible controller due to the number of buttons it has accessible to the performer in conjunction with its 4 continuous controllers (figure 3).

Figure 3: The Saitek Cyborg 3D joystick has 13 buttons and 4 continuous controllers. The three buttons on the top, labeled 1 to 3, can be used to route the continuous controller values to different destinations.

It is often desirable to control a number of processes with one controller, for instance multiple voices of a polyphonic engine. In the case of the Cyborg 3D, OSC messages in the form of '/button-number/continuous-controller value' can be constructed that will render the continuous contoller values *modally* addressable to different voices. For example, holding down button 1 and moving the joystick up and down would produce messages like, '/1/vertical value', headed to the first voice of our processing engine. Holding down buttons 2 and 3 while moving the vertical axis of the controller would produce both '/2/vertical value' and /3/vertical value' messages, thereby controlling the second and third voices of the processing engine. A similar technique can be applied to any combination of held buttons and manipulated continuous controllers to effectively turn the 4 available continuous controllers into a much larger number of control data sources.

6. FUTURE OF OSC

Here are some ideas for the future of OSC. Obviously, all implementations of OSC should be completed and made consistent, able to both send and receive the full OSC spec including type tags, bundles, time tags, etc. Full use of time tags requires solving the time synchronization problem; experiments must be done to see if NTP and SNTP will be adequate.

There is no reason that OSC should be so tied to UDP; more systems should support OSC via TCP, especially in situations where guaranteed delivery is more important than low latency.

OSC's query system is still more or less in an experimental stage; the community should standardize the syntax and semantics of a collection of useful queries.

Of course we would like to see more systems using OSC. On the day this paper was submitted we heard that Carlos Agon had completed an initial implementation of OSC in both OpenMusic [1] and Macintosh Common Lisp. The jMax [7] team is also planning to implement OSC.

The translation between OSC and XML used by *flosc* could be generally useful to the OSC community; we would like to see it become standardized.

7. ACKNOWLEDGMENTS

Amar Chaudhary, John Ffitch, Guenter Geiger, Camille Goudeseune, Peter Kassakian, Stefan Kersten, James McCartney, Marcelo Wanderley, David Wessel.

8. REFERENCES

[1] Agon, C., Assayag, G., Laurson, M. and Rueda, C. Computer Assisted Composition at Ircam : PatchWork & OpenMusic. (1999) Computer Music Journal. (http://www.ircam.fr/equipes/repmus/RMPapers/CMJ98/index.html)

[2] Bargar, R., Choi, I., Das, S. and Goudeseune, C., Model-based interactive sound for an immersive virtual environment, in International Computer Music Conference, (Aarhus, Denmark, 1994), ICMA, 471-474. (Current documentation and software: http://www.isl.uiuc.edu/software/software.html)

[3] Berry, M., GrainWave 3, 2002. (http://www.7cities.net/users/mikeb/GRAINW.HTM).

[4] Campion, E., Momeni, A. and Murota, C., Persistent Vision, 2002. (http://cnmat.CNMAT.Berkeley.EDU/~ali/documents/persisten-vision.html).

[5] Chaudhary, A., Freed, A. and Wright, M. An Open Architecture for Real-Time Audio Processing Software, Audio Engineering Society, Audio Engineering Society 107th Convention, preprint #5031, 1999. (http://cnmat.CNMAT.Berkeley.EDU/AES99/docs/osw.pdf)

[6] Chun, B., flosc : Flash OpenSound Control, 2002. (http://www.benchun.net/flosc).

[7] Dechelle, F., Borghesi, R., De Cecco, M., Maggi, E., Rovan, B. and Schnell, N. jMax: An environment for real-time musical applications. (1999) Computer Music Journal, 23 (3). 50-58.

[8] Durand, H. and Brown, B.D., A Distributed Audio Interface using CORBA and OSC, in Symposium on Sensing and Input for Media-Centric Systems (SIMS), (Santa Barbara, CA, 2002), 44-48. (http://www.create.ucsb.edu/sims/PDFs/Durand_Brown_SIMS.pdf)

[9] Fléty, E., AtoMIC Pro: a Multiple Sensor Acquisition Device, in Conference on New Interfaces for Musical Expression (NIME-02), (Dublin, Ireland, 2002), 96-101.

[10] Friedl, J. Mastering Regular Expressions: Powerful Techniques for Perl and Other Tools. O'Reilly & Associates, Sebastopol, CA, 1997.

[11] Garnett, G.E., Choi, K., Johnson, T. and Subramanian, V., VirtualScore: Exploring Music in an Immersive Virtual Environment, in Symposium on Sensing and Input for Media-Centric Systems (SIMS), (Santa Barbara, CA, 2002), 19-23. (http://www.create.ucsb.edu/sims/PDFs/Garnett_SIMS.pdf)

[12] Garnett, G.E., Jonnalagadda, M., Elezovic, I., Johnson, T. and Small, K., Technological Advances for Conducting a Virtual Ensemble, in International Computer Music Conference, (Habana, Cuba, 2001), 167-169.

[13] Goudeseune, C., Garnett, G. and Johnson, T., Resonant Processing of Instrumental Sound Controlled by Spatial Position, in CHI '01 Workshop on New Interfaces for Musical Expression (NIME'01), (Seattle, WA, 2001), ACM SIGCHI. (http://www.csl.sony.co.jp/~poup/research/chi2000wshp/papers/goudeseune.pdf)

[14] Hankins, T., Merrill, D. and Robert, J., Circular Optical Object Locator, in Conference on New Interfaces for Musical Expression (NIME-02), (Dublin, Ireland, 2002), 163-164.

[15] Hansen, M. and Rubin, B., LISTENING POST: GIVING VOICE TO ONLINE COMMUNICATION, in International Conference on Auditory Display, (Kyoto, Japan, 2002). (cm.bell-labs.com/cm/ms/who/cocteau/papers/pdf/icad2002.pdf)

[16] Impett, J. and Bongers, B., Hypermusic and the Sighting of Sound - A Nomadic Studio Report, in International Computer Music Conference, (Habana, Cuba, 2001), ICMA, 459-462. (91 page report: http://www.meta-orchestra.net/downloads/report.pdf)

[17] Jehan, T. and Schoner, B., An Audio-Driven Perceptually Meaningful Timbre Synthesizer, in International Computer Music Conference, (Habana, Cuba, 2001), 381-388. (http://www.media.mit.edu/hyperins/papers/Tristan_ICMC_2001.pdf)

[18] Kling, G. and Schlegel, A., OSCar and OSC: Implementation and use of Distributed Multimedia in the Media Arts, in Symposium on Sensing and Input for Media-Centric Systems (SIMS), (Santa Barbara, CA, 2002), 55-57. (http://www.create.ucsb.edu/sims/PDFs/OSCar_SIMS.pdf)

[19] La Kitchen Hardware, KROONDE: 16 sensors wireless UDP interface, 2002. (http://www.la-kitchen.fr/hardw/presentation_en.pdf).

[20] Madden, T., Smith, R.B., Wright, M. and Wessel, D., Preparation for Interactive Live Computer Performance in Collaboration with a Symphony Orchestra, in International Computer Music Conference, (Habana, Cuba, 2001), ICMA.

[21] McCartney, J., A New, Flexible Framework for Audio and Image Synthesis, in International Computer Music Conference, (Berlin, Germany, 2000), ICMA, 258-261. (www.audiosynth.com)

[22] McMillen, K., Wessel, D. and Wright, M. The ZIPI music parameter description language. (1994) Computer Music Journal, 18 (4). 52-73. (http://cnmat.CNMAT.Berkeley.EDU/ZIPI/mpdl.html)

[23] Mills, D.L., Network Time Protocol (Version 3) Specification, Implementation and Analysis, 1992. (http://www.faqs.org/rfcs/rfc1305.html).

[24] Mills, D.L., Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI, 1996. (http://www.faqs.org/rfcs/rfc2030.html).

[25] Overholt, D., The MATRIX: A Novel Controller for Musical Expression, in CHI '01 Workshop on New Interfaces for Musical Expression (NIME'01), (Seattle, WA, 2001), ACM SIGCHI. (http://www.csl.sony.co.jp/person/poup/research/chi2000wshp/papers/overholt.pdf)

[26] Overholt, D., Musical Mapping and Synthesis for the MATRIX Interface, in Symposium on Sensing and Input for Media-Centric Systems (SIMS), (Santa Barbara, CA, 2002), 7-10. (http://www.create.ucsb.edu/sims/PDFs/DanO_SIMS.pdf)

[27] Pope, S.T. and Engberg, A., Distributed Control and Computation in the HPDM and DSCP Projects, in Symposium on Sensing and Input for Media-Centric Systems (SIMS), (Santa Barbara, CA, 2002), 38-43. (http://www.create.ucsb.edu/sims/PDFs/Pope_SIMS.pdf)

[28] Puckette, M. Combining event and signal processing in the MAX graphical programming environment. (1991) Computer Music Journal, 15 (3). 68-77.

[29] Puckette, M.S., Pure Data, in International Computer Music Conference., (Hong Kong, 1996), ICMA, 269-272. (http://www-crca.ucsd.edu/~msp/Publications/icmc96.ps)

[30] Ramakrishnan, C., Java OSC, 2003. (http://www.mat.ucsb.edu/~c.ramakr/illposed/javaosc.html).

[31] Wei, S.X., Visell, Y. and MacIntyre, B. TGarden Media Choreography System, Technical Report, Graphics, Visualization and Usability Center, Georgia Institute of Technology, Atlanta, GA, 2003. (http://www.gvu.gatech.edu/people/sha.xinwei/topologicalmedia/papers/Sha-MacIntyre02.pdf)