Differentiating Values and Properties in an Open Sound Control Namespace

Authors...
institutions...

ABSTRACT

The paper suggests an approach for creating structured Open Sound Control (OSC) messages by separating the addressing of node values and node properties. This includes a mechanism for querying values and properties. As a result, it is possible to address complex nodes inside of more complex tree structures using an OSC namespace. This is particularly useful for creating flexible communication in modular systems. A prototype implementation is presented and discussed.

Keywords

Jamoma, OSC, Open Sound Control, standardization

1. INTRODUCTION

Open Sound Control [1] is a protocol for transmitting messages that address nodes using a familiar "slash" notation. It has been adopted by the computer music community as a de facto standard for communication both within a single process and between separate processes. OSC does not prescribe any particular namespace standardization, only a message formatting standardization.

The authors are involved in developing OSC namespaces for the Jamoma[2] project. Jamoma is a modular system for developing high-level modules in the Max/MSP/Jitter environment [8]. Communication within and between Jamoma modules is handled using the Open Sound Control protocol.

As Jamoma's modular structures grew more complex, the authors found the flat namespace conventions of OSC to be inadequate for addressing our constructs. Specifically, we found OSC to be ideal for specifying an address to a node. However, it became increasing unclear what to do once the address reached this said node. The problem is exacerbated when the node itself implements its own OSC namespace.

[1] `http://www.opensoundcontrol.org`
[2] `http://www.jamoma.org`

NIME08, Genova, Italy

2. REPRESENTING COMPLEX STRUCTURES IN AN OSC NAMESPACE

2.1 Open Sound Control

Open Sound Control was developed in the late 1990s at CNMAT and has emerged as the de facto standard for communication among computers, sound synthesizers, and other multimedia devices in the computer music research community [12]. The original idea of OSC is that it is tree-structured into a hierarchy called the *address space* where each of the nodes has a symbolic name and is a potential destination of OSC messages [12]. Oppositely to the well-defined structure of MIDI, the *open* nature of OSC means that the address space is defined and created by the "implementors idea of how these features should be organized." For example:

```
/voices/3/freq 440.0
```

This open approach has made OSC useful in a number of different situations and adaptable to situations not foreseen by developers [10]. However, this lack of standardization in namespace schemas is also probably a major reason that OSC has not gained a more widespread in commercial software applications.

2.2 OSC Namespace Standardization

There are a number of projects ongoing which attempt to standardize namespaces for various uses of Open Sound Control. A partial listing of these follow.

Occam [3] takes OSC messages and converts them to MIDI. It exports a MIDI source to CoreMIDI which can be used in any Mac OS X application that accepts MIDI. It broadcats the existence of this OSC-to-MIDI service using Rendezvous (Zero-Conf).

LIBOSCQS[4] is a library to provide a Query System and Service Discovery for applications using the Open Sound Control (OSC) protocol[9]. This page contains an introduction to LIBOSCQS.

Integra, an EU Culture 2000 project[5] aiming at developing "a new software environment to make music with live electronics," and to modernize works that use old technology [1]. Part of the development has been concerning creating a client/server model. The Integra OSC address space suggests using the : (colon) to only set messages without out-

[3] `www.mat.ucsb.edu/$\sim$c.ramakr/illposed/occam.html`
[4] `http://liboscqs.sourceforge.net`
[5] `http://www.integralive.org`

putting, to avoid feedback. They also include a .get message to query for values.
SuperCollider, a text based audio synthesis programming language is using OSC for internal communication in its client/server model.
Iannix [6] is a graphical editor of multidimensional and multiformal scores, a kind of poly-temporal meta-sequencer, based on the former UPIC created by Iannis Xenakis." The current implementation is built around a client/server model where all communication is handled using OSC. [2]
Open Sound Control 2.0 proposal – Developers of the Lemur multitouch interface presented a draft for a 2.0 version of Open Sound Control during an OSC developer meeting at NIME 2006 [4], and has later changed the suggestion slightly [5].
OSCQS [3]
TUIO, an OSC address space for handling data from the Reactable [6] was been developed at MTG at Pompeu Fabra.
Copperlan TODO:WHAT IS THAT?
McGill Mapper [7]

2.3 Syntactic Definitions

In actual practice, these various independent efforts at standardizing namespaces incorporate syntactic elements with conflicting meanings as compared to each other. There are some commonalities to these efforts and the problems that they try to address. Specifically:

- How to get the value of a node (vs. setting the value of a node)
- How to access properties of a node (as opposed to the value of the node)

It is clear from a review of these sundry efforts that additional syntax is needed for clarifying function, address, or both when developing a complex OSC namespace.

In Jamoma, there are a wide variety of cases where a value may be set for a given node in the OSC namespace. However, the node also has additional properties which define the behavior of that node. For example, a node may represent the temperature such as:

```
/path/to/the/temperature 32
```

However, we may wish to set a property of the node so that it knows how to interpret the temperature. Is it specified in Celsius or Fahrenheit? Is this a shift from an existing value, or is setting the value directly? Before we send the value, perhaps we should query the node to find out what an acceptable range is for this node.

This problem of accessing properties of a node, as opposed to the value of a node, is particularly troublesome using the standard OSC addressing conventions (using only slashes to navigate the tree). One example of a system parallel to OSC for representing and sending data over a network is XML.

2.4 XML

Extensible Markup Language (XML)[7] is a particularly relevant analogue to Open Sound Control. XML defines a means for formatting data, but not the data or the anything specific to the dataspace itself CITATION OF THE XML SPEC http://www.w3.org/TR/2006/REC-xml-20060816/ .

A number of standardized namespaces using XML have gained wide adoption, including Scalable Vector Graphics (SVG)[8], XHTML[9] and SOAP[10]. SOAP is of particular interest because it is designed as a protocol for exchanging structured information.

Using XML, information is encapsulated into elements. These elements could be structured such that they are analogous to an OSC message. For the purpose of this discussion we will treat them as such. Using XML elements, one way to represent the above temperature example is thus:

```
<path>
  <to>
    <the>
      <temperature>
        32
      </temperature>
    </the>
  </to>
</path>
```

This is clearly more cumbersome to manually type than the Open Sound Control message, it is more work for the receiving processor to parse, and it uses more bandwidth as well. However, XML elements are able not only to express a value (content in xml parlance) between the tags, but also they can provide properties (attributes) to the node. For example, we may provide the type of temperature we are specifying:

```
<path>
  <to>
    <the>
      <temperature unit='Kelvin'>
        32
      </temperature>
    </the>
  </to>
</path>
```

We suggest a model where it is possible to fork an OSC address to access not only the value of the node, but also the properties of that node, much like what is possible in other existing models such as XML. Different than XML, we will propose that they can be addressed independently rather than simultaneously.

2.5 The colon separator

In addition to the ASCII symbols already reserved for specific purposes within the OSC protocol [11], we introduce the colon ":" as a separator between the OSC address of a node and the namespace for accessing the properties of the node:

```
<parameter address> <value>
<parameter address>:<property address> <value>
```

The former message sets the value of the node just as it would using the existing OSC conventions. The latter sets a property of the value. Again using temperature as an example, we can send two messages: one for setting the unit property, and one for setting the value.

```
/path/to/the/temperature:/unit Fahrenheit
/path/to/the/temperature 212
```

This usage of the colon has precedent in POSIX paths when addressing remote filesystems. For example, scp uses the following format to locate a file on a remote server:

[6] http://sourceforge.net/projects/iannix
[7] http://www.w3.org/XML/
[8] http://www.w3.org/Graphics/SVG/
[9] http://www.w3.org/TR/xhtml1/
[10] http://www.w3.org/TR/soap/

`user@host.com:/path/to/file`

In our case, we are indeed using the colon to separate OSC namespaces, one of which is the address to a node and one of which is the address within that remote node.

Section 3 provides an illustration of the ideas suggested here. In the following discussion the address of the value will be omitted for the sake of brevity; e.g. `/computer/module/parameter:property` will be abbreviated as `:property`.

3. JAMOMA AS A PROTOTYPE IMPLEMENTATION

The general concepts introduced in the previous section has formed the basis for the implementation of a standardized namespace addressing properties of values in the Jamoma framework for Max/MSP [8]. The following description of the implementation is meant to serve as an illustration of the concept, as well as indicating how it might provide the user with extended and structured control of available values.

Jamoma distinguish between module *parameters* and *messages*. Parameters alter the state of the module. When querying or setting the state of the module, parameter values need to be retrieved or set. In contrast messages are stateless. One example of a message would be a request to open a reference file for the module. Apart from the difference in terms of state, messages and parameters are implemented and behave the same way, and the following discussion is equally valid for both. In the following discussion a *value* can be a module parameter or message.

3.1 Value type

The type of the value can be specified. Possible types are *none*, *boolean*, *integer*, *float*, *symbols* and *list*. If one do not want to restrict the type of the value, it can be set to *generic*. The *none* type is only valid for messages. Some of the properties below will only be valid for certain types of values. The type property is accessed thus:

```
:type
:type/get
```

3.2 Controlling the value itself

In addition to setting the value directly, it can be set and retrieved as a property. If the value is of integer, float or list type it can also be stepwise increased or decreased. If so the size of the steps is itself a property:

```
:value
:value/get
:inc
:dec
:value/stepsize
:value/stepsize/get
```

3.3 Controlling the range

For integer, float and list values a range can be specified. This can be useful for setting up autoscaling mappings from one value to another, or for clipping the output range. The clipping property can be *none*, *low*, *high* or *both*. The range properties are accessed thus:

```
:range
:range/get
:range/clipmode
:range/clipmode/get
```

3.4 Filtering of repetitions

It is sometimes useful to filter repetitions to avoid redundant processing. The *boolean* repetitions property is accessed thus:

```
:repetitions
:repetitions/get
```

3.5 Ramping to new values

The ability to smoothly move from one value to another is fundamental to any kind of transition ands transformation of musical or artistic material. Jamoma offers the possibilty of ramping from the current to a new value in a set amount of time. While the OSC message

`/myComputer/myModule/myParameter 1.0`

will set the parameter value to 1.0 immediately, the message

`/myComputer/myModule/myParameter 1.0 ramp 2000`

will cause the value to ramp to 1.0 over 2000 milliseconds. Ramping in Jamoma works with messages and parameters of type integer, float and list.

Jamoma offers vastly extended possibilities in how ramping can be done as compared to Max. In Jamoma the process of ramping is made up from the combination of two components: A driving mechanism cause calculations of new values at desired intervals during the ramp, while a set of functions offers a set of curves for the ramping. Both components are implemented as C++ APIs, and can easily be extended with new ramp or function *units*, expanding the range of possible ramping modes.

3.5.1 The Jamoma RampLib API

The Jamoma RampLib API provides a means by which to create and use *ramp units* in Jamoma. A ramp unit is a self-contained algorithm that can slide from an existing value to a new value over a specified amount of time according to different timing mechanisms. Each ramp unit is implemented in the form of two C++ files: a source file and a header file that provides an interface for the source file. Currently four such ramp units are implemented:

- *none* - jumps immediately to the new value. Typically used for values where ramping do not make sense.
- *scheduler* - use the Max internal clock to generate new values at fixed time intervals.
- *queue* - ramping using the Max queue, updating values whenever the processor has free capacity to do so.
- *async* - only calculate new values when requested to do so. This might be used in video processing modules to calculate fresh values immediately before processing the next video image or matrix.

When a new ramp is started, the ramp unit internally use a normalized ramping value, increasing linearly from 0.0 to 1.0 over the duration of the ramp. Whenever the ramp unit is to provide a new value, it updates the normalized ramping value, and pass it to a Function Unit as described in Section 3.5.2. The normalized value returned is then scaled to the range defined by the start and end values for the ramp, and passed on to the module.

3.5.2 The Jamoma FunctionLib API

The Jamoma FunctionLib API provides normalized mappings of values $x \in [0, 1]$ to $y \in [0, 1]$ according to functions $y = f(x)$. The FunctionLib can easily be expanded by introducing new functions in the form of two C++ files: a source file and a header file that provides an interface for the source file.

Currently five functions are implemented:

- Linear: $y = x$.
- Cosine: $y = -\frac{1}{2} \cdot cos(x \cdot \pi) + \frac{1}{2}$.
- Lowpass series: $y[n] = y[n-1] \cdot k + x[n] \cdot (1-k)$, where k is a feedback coefficient.
- Power function: $y = x^k$, where the parameter k can be set.
- Hyperbolic tangent: $y = c \cdot (tanh(a \cdot (x-b)) - d)$, where coefficients a, b, c, d depends on the width and offset of the curve.

There are plans to introduce exponential functions.

3.5.3 OSC namespace for ramping properties

Ramping properties are addressed using `:ramp/drive` and `:ramp/function` OSC nameclasses. In addition to the ablity of setting or getting current ramp driving mechanism, it might be useful to have the module return a list of all available ramp units. This can be done by means of a `/dump` message:

```
:ramp/drive
:ramp/drive/get
:ramp/drive/dump
```

Some ramp units might have additional parameters that can be controlled by the user. For instance the user can control how often the *scheduler* ramp unit is to update; the granularity of the ramp. In general a ramp unit parameter `/foo` might be accessed thus:

```
:ramp/drive/parameter/foo
:ramp/drive/parameter/foo/get
```

If the current ramp unit do not have a `/foo` parameter, the message will be ignored by the ramp unit. It is possible to query what parameters are available for the current ramp unit with the `/dump` message:

```
:ramp/drive/parameter/dump
```

The function used for the ramp can be set or queried in much the same way. It is also possible to request information on available function units with the `/dump` message:

```
:ramp/function
:ramp/function/get
:ramp/function/dump
```

Some function units might have additional coefficients influencing the shape of the curve, e.g. the exponent of the *power* function unit can be set. In general a function unit parameter `/bar` might be accessed thus:

```
:ramp/function/parameter/bar
:ramp/function/parameter/bar/get
```

If the current function unit do not have a `/bar` parameter, the message will be ignored by the function unit. It is possible to query what coefficients are available for the current function unit with the `/dump` message:

```
:ramp/function/parameter/dump
```

3.6 Controlling the user interface

In certain applications the CPU overhead of continuously updating the graphical user interface whenever parameter or message values change might become a burden, competing for CPU with e.g. video processing algorithms. If the user do not need continuous visual feedback on updated values of parameters or messages, the GUI for the parameter or message can be frozen, freeing up the processor and GPU for tasks considered more important:

```
:ui/freeze
:ui/freeze/get
```

A parameter or message that has its GUI frozen can be forced to update and refresh the displayed value once by means of the message:

```
:ui/refresh
```

3.7 Description

The description property is a string providing a text description of the parameter. In Jamoma this is used for autogenerating online documentation of the modules. It can also be used for building modules that retrieve the total namespace of all Jamoma modules used, and provide interactive documentation of available parameters. The descriptions property is accessed as:

```
:description
:description/get
```

3.8 Retrieving all properties of a value

The discussion so far has illustrated what properties are available for values in Jamoma, and how the can be accessed one by one. Jamoma also offers the possibility of retrieving the total namespace for one value with the message:

```
:namespace/dump
```

In addition current settings for all properties can be retrieved by the message:

```
:properties/dump
```

4. DISCUSSION AND FURTHER WORK

4 ramp units times 5 function units = 20 ramping modes

ramp units can be used for other scheduled processes as well

Possibility of expanding ramp units as low frequency oscillators

function units can be used elsewhere, e.g. for mapping

Audio rate ramp unit.

DataspaceLib

Querying - we propose a different system to the Lemur OSC2 draft

Ramp Lib and Function Lib can be used outside the context of jcom.parameter and jcom.value: jcom.map and jcom.ramp

5. ACKNOWLEDGMENTS

All Jamoma developers and users for valuable contributions. iMAL

6. REFERENCES

[1] J. Bullock and H. Frisk. Libintegra: a system for software-independent multimedia module description and storage. In *Proceedings of the International Computer Music Conference*, pages 43–46, Copenhagen, DK, 2007.

[2] T. Coduys and G. Ferry. IanniX aesthetical/symbolic visualisations for hypermedia composition. In *Proceedings of the International Conference Sound and Music Computing (SMC'04)*, 2004.

[3] M. Habets. Oscqs - schema for opensound control query. System version 0.0.1, 2005.

[4] Jazzmutant. Extension and enchancement of the OSC protocol. Draft Presented at the OSC-meeting at NIME 2006, IRCAM, Paris, 2006.

[5] Jazzmutant. Extension and enchancement of the OSC protocol. Draft 25 July, 2007.

[6] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza. Tuio - a protocol for table based tangible user interfaces. In *Proceedings of the 6th International Gesture Workshop, Vannes, France, 18-21 May*, 2005.

[7] J. Malloch, S. Sinclair, and M. M. Wanderley. From controller to sound: Tools for collaborative development of digital musical instruments. In *Proceedings of the 2007 International Computer Music Conference*, Copenhagen, Denmark, 2007. San Francisco: ICMA.

[8] T. Place and T. Lossius. Jamoma: A modular standard for structuring patches in max. In *Proceedings of the 2006 International Computer Music Conference*, pages 143–146, New Orleans, LA, 2006. San Francisco: ICMA.

[9] A. W. Schmeder and M. Wright. A query system for Open Sound Control. Draft Proposal, July 2004.

[10] M. Wright. Open sound control: an enabling technology for musical networking. 10(3):193–200, 2005.

[11] M. Wright and A. Freed. Open Sound Control: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 International Computer Music Conference*, pages 101–104, Thessaloniki, Greece, 1997. San Francisco: ICMA.

[12] M. Wright, A. Freed, and A. Momeni. Opensound control: State of the art 2003. In *NIME '03: Proceedings of the 2003 International Conference on New Interfaces for Musical Expression*, 2003.