Composability for Musical Gesture Signal Processing using new OSC-based Object and Functional Programming Extensions to Max/MSP

Adrian Freed
CNMAT
Dept. of Music
UC Berkeley
adrian@cnmat.berkeley.edu

John MacCallum
CNMAT
1750 Arch Street
Berkeley, CA 94709
johnmac@berkeley.edu

Andy Schmeder
CNMAT
1750 Arch Street
Berkeley, CA 94709
schmeder@berkeley.edu

ABSTRACT
An effective programming style for gesture signal processing is described using a new library that brings efficient run-time polymorphism, functional and instance-based object-oriented programming to Max/MSP. By introducing better support for generic programming and composability Max/MSP becomes a more productive environment for managing the growing scale and complexity of gesture sensing systems for musical instruments and interactive installations.

Keywords
Composability, object, Open Sound Control, Gesture Signal Processing, Max/MSP, Functional Programming, Object-Oriented Programming, Delegation

1. INTRODUCTION
Open Sound Control (OSC) was originally designed as a message-passing format to facilitate exchange of control parameters between programs on different computers across a network. Since its release in 1997 [16] OSC has proven to be useful for message exchanges between processes on the same computer system and more recently within processing modules in the same program [17]. This paper shows how OSC messages can be used to provide composable, dynamic data types, to support generic, object-oriented and functional programming styles in dynamic, visual dataflow programming languages such as Max/MSP and PD.

2. Composable Aggregate Types
Max/MSP and PD are among the most popular programming languages for media computing and gesture signal processing for musical applications [6, 7]. An unfortunate legacy of the early success of these programs is their spartan support for data types and the lack of objects and a composable and extensible type system. These limitations are particularly problematic for the NIME community as projects increasingly involve complex gestural signal processing flows for large numbers of heterogeneous sensors and actuator types.

The solution advanced in this paper is to use Open Sound Control messages and native Max/MSP patches and externals to implement objects for dynamic, instance-based object-oriented programming (sometimes referred to as prototype-based programming). Self [15], ECMAScript [4], Javascript and NewtonScript [8, 12] are examples of languages using this programming style [1, 9, 11].

The ideas introduced here are embodied in a freely available collection of Max/MSP externals and patches known as the "o." library (pronounced "Oh dot"). We demonstrate applications of this library and new, productive programming techniques that leverage the high degree of composability [2] that emerges when delegation, aggregation and mapping techniques of object-oriented programming are melded to dataflow execution models.

NIME'11, 30 May–1 June 2011, Oslo, Norway.

3. OSC Object Construction and Dispatch
3.1 Introduction
In prototype-based object-oriented programming objects are created from scratch (ex nihilo) or by cloning [14] and in some languages modifying an existing prototype object. The "o." library uses the cloning approach, allocating new memory, copying the contents of an inbound OSC message and modifying and adding to the copy as required (in the spirit of Kevo [9]). This approach follows the convention of Max primitive types, is easy to understand, avoids atomicity issues and allows programs to be easily distributed to multiple processors without the cost of managing references. It also invites a pure functional programming style with the well-known advantages of minimizing hidden state or stored values.

The conceptual steps from class-based object-oriented programming (OOP) to what we are doing with OSC here are small: concatenation of objects is sufficient for inheritance [9] and objects can serve as their own type definitions [5].

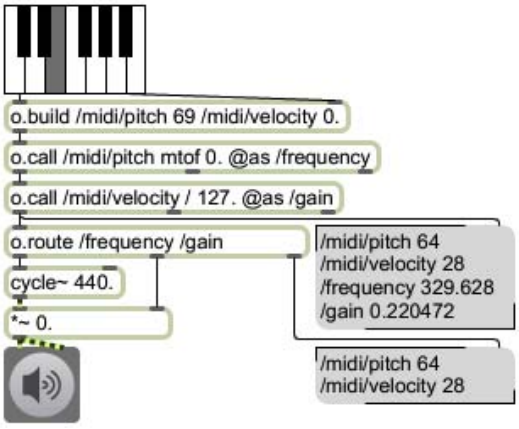


Figure 1. Aggregating values into OSC bundles.

3.2 Example
Figure 1 shows how o.build is used to create an interface to the Max keyboard object (kslider) that captures both legacy representations of the depression of a key on a musical keyboard as well as more contemporary ones. Bundling the

data from each outlet of the kslider better reflects the atomic binding of the two values implied by the gesture that created them than sending them as separate data at different times out of separate outlets. This common use of the o.build method is analogous to the "named associations" style of Ada function call arguments [13]. Instead of having to direct the right parameters at the right time to the appropriate inlet, parameters are named and bound together into a single bundle. This alternative to the positional association style of Max/MSP is also exploited in Jamoma [10].

The o.route method complements o.build to bring values out of the OSC bundle into the Max/MSP message world. Notice that the last outlet of o.route outputs a new bundle containing the unmatched elements of the original bundle. The "remainder" bundle can be further processed as the patch evolves–the essence of this delegation style of object-oriented inheritance. It is useful to contrast this approach with static class-based inheritance. The key difference is that in the delegation style new object types are created dynamically by simply adding new address/value pairs to existing objects. Programmers do not need to consult object definitions or API's to understand objects, their derivatives and promises: they simply look at the data in the objects themselves as they are formed and reformed using, for example, the gray UI Max object o.message which is analogous to the Max message box.

Just as Max wiring simulates physical wiring, OSC building and routing simulates scalable strategies used for wiring complex physical world systems, i.e. the labelling, color coding, bundling and bussing of wires.

4. Making OSC methods from Max patches

Using function-mapping approaches that are analogous to "map" and "apply" from Lisp, existing Max/MSP externals and patches can operate on data in OSC bundles. This eliminates the need for a large number of new speciality operators to be introduced and learned.

The most common scheme for this is implemented in the o.call method. This Max object instantiates a max patch internally according to its arguments and then routes named messages from incoming OSC packets to the internal max patch. Finally it gathers the output into an OSC bundle. Figures 1 and 2 illustrate this for various common scaling operations with floating-point division and the mtof (midi to frequency) function.

The o.call method uses prefix and suffix operators so it is syntactically closer to Lisp and other functional languages than to C. To clarify subsequent examples we note that the argument list comes first followed by the function description (a Max patch which o.call dynamically instantiates). Finally there may be closing attributes following an @ symbol. These are used to describe what to do with the results of the function call mapping. By default the result is bound to the same name as the first argument pattern. @as is followed by the names to be assigned to new elements that will be added to the incoming bundle. @prepending specifies a prefix to be added to the address of the first argument. These various conventions will be liberally used in the following examples.

5. Delegation-style Inheritance

In Figure 2 the example of Figure 1 has been augmented with the feature of pitch-dependent panning to illustrate how delegation can be used to add functionality to programs in a way that promotes reuse (the core benefit of object-oriented programming).

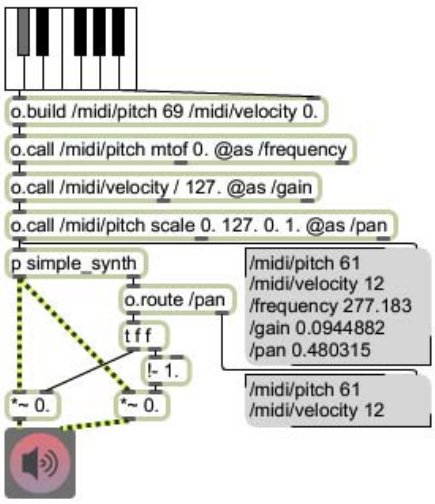


Figure 2. Delegation-style inheritance.

The objects in the top most grey box create a new bundle that includes the contents of the incoming bundle adding a new /pan value computed from the pitch in the incoming bundle.

The key to reuse is that neither the original patch assembling the description of the keyboard gesture nor the synthesizer need any changes to support the new /pan parameter. The additional sound functionality is added by using the delegation outlet (conventionally the rightmost) of the synthesizer patch. These new functionalities can of course be encapsulated as required. Note that the /pan route operation delegates its unmatched bundle inviting future inheritance.

6. External Sources/Sinks of OSC Bundles

External sources and sinks of OSC data include the venerable udpsend and udpreceive objects and slipOSC for serial-wrapped OSC (typically from USB serial devices).

The new o.io externals replace these Max functions by enumerating (o.io.discover) and wrapping (o.io) data from all the core I/O subsystems of OS/X computers as OSC messages. This sort of wrapping functionality is already partially addressed by programs such as Osculator (http:/osculator.net/) and Glovepie (http://glovepie.org). Unfortunately there is measurable and potentially troublesome variance in the delay of messages via these programs. The o.io object minimizes these by time-tagging the data using the lowest-level APIs to get as close as possible to the actual time the data was acquired. The o.io method already supports core popular protocols HID, UDP, TCP, MIDI, serial and proprietary API's such as the one provided for the built-in laptop motion sensors and multitouch trackpads.

The o.io method was carefully designed for extensibility so that new API's and device types can be easily added. Bundles from o.io typically contain the raw data from the API and then one or more overlays of higher-level interpreted data according to the device. For example some HID protocol devices provide entries in a table with useful names to substitute for the parameter numbers of the core data stream.

6.1 OSC Bundle Methods

Certain operations on bundles are clumsy to do by breaking them up into Max/MSP native types and reassembling them with o.build. These include merging, unions, intersections and accumulation for which the o.var method is provided. The o.if method is unusual in that it only inspects the contents of a bundle thereby avoiding a copy operation as it directs the bundle out of the "true" or "false" outlet according to the evaluation of a conditional expression.

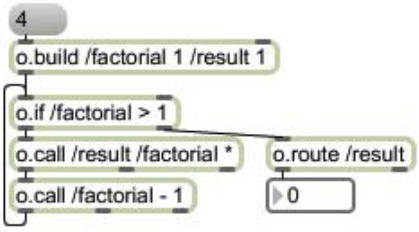


Figure 3. Recursion and o.if

The factorial calculation of Figure 3 illustrates o.if in action and how recursion is compactly done with "o." methods. Note that the evolving state of this computation is traceable by simply collecting the bundles recursively passed back. The concentration of observable state into bundles turns out to be a very productive programming technique, minimizing bugs that are hard to find because of state hidden within Max externals and patches.

7. Gesture Signal Processing with "o."

This section elaborates a complete gesture signal processing application by analysing the patch of figure 4 from top to bottom:

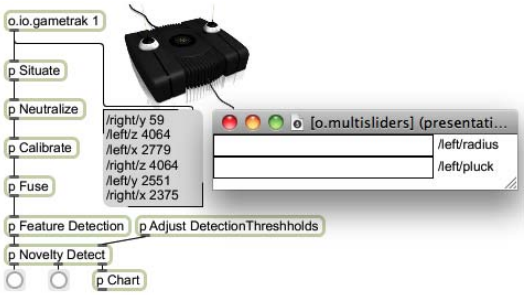


Figure 4: Feature Extraction

The source of analysed gesture data is a popular controller for experimental music called the Gametrak [3]. It provides data from the unwinding of retracting cords passing through the centers of a pair of joysticks. In the following sections we will trace the series abstractions encountered as packets move from top to bottom in this patch.

7.1 Situate

The HID encoding of values from the Gametrak is designed according to the viewpoint of the inventors of HID and their imagined uses for the Gametrak. We use the term "situate" to refer to the process of complementing this deferred agency of the hardware builders with the meaning the user of the Gametrak and Max patch can attribute according to their immediate situation. In the example shown this involves renaming.

```
o.rename /hid/Game-Trak-V1.3/1/15 /left/x
o.rename /hid/Game-Trak-V1.3/1/16 /left/y
o.rename /hid/Game-Trak-V1.3/1/17 /left/z
```

The appearance of x,y,z suggests the user's comfort with cartesian coordinate conventions. Another user might prefer the terms NS, WE, and Extension.

7.2 Neutralize

The value stream from this device (as with MIDI) confronts us with particular implementation choices: integers and the domain 0-4095. We neutralize this using the unit intervals [0-1] or [-1 1], the latter being useful in this case to represent directional deviations from the center of the joystick. These intervals are easy to scale by multiplication.

Regular expressions are used to match both the left and right addresses and to precisely call out a different range for x or y, or z.

```
o.call /*/z scale 4095 0 0. 1.
o.call /*/[xy] scale 0 4095 -1. 1.
```

This demonstrates the value of dynamic method routing and a surprising conciseness. The combination of wiring and patterns takes care of what is typically done more verbosely in lexical programming languages using terms such as lambda, self, this,or with.

7.3 Display

The named sliders displaying some of the values in the neutralized packet (in Figure 4) were built using o.multislider implemented using the same functional programming strategies of o.call while in addition tiling out the user interface.

7.4 Calibrate

Here we "taint" the domain of the neutralized gesture measurements by mapping them to a calibrated frame with extension in meters and positions as angles. We use the "prepending" attribute to add this interpreted value to the neutralized one rather than replace it.

```
o.call /*/[xy] * 30. @prepending /degrees
o.call /*/z * 2. @prepending /meters
```

This is an example of designing for reuse–a key aspect of composability. This calibration increases the potential for future reuse of the OSC packet (i.e. the object) without requiring knowledge of this future and without imposition of a complex interface.

7.5 Fuse

A simple sensor fusion is performed by combining the x,y axis data to create a radius and rotation. These are added to the copy of the incoming bundle.

```
o.call /left/x /left/y expr "sqrt($f1 * $f1 + $f2 * $f2)" @as /left/radius
o.call /left/x /left/y atan2 @as /left/orientation
```

7.6 Feature Detector

This feature detector computes the direction of a "stirring" gesture on the left string of the Gametrak. The algorithm is simply to look at the sign of the derivative of the rotation of the gesture. The first step is to use o.remember to build a packet containing the incoming packet and its predecessor. The elements of these two packets are distinct because the name "/was" is prepended to all the data in the old bundle. This avoids the complexity of the classical alternative: pointers or references. The difference operator is simply composed from Max/MSP's subtraction primitive:

```
o.remember unit delay
o.call /left/orientation /was/left/orientation - 0. @as /delta/left/orientation
```

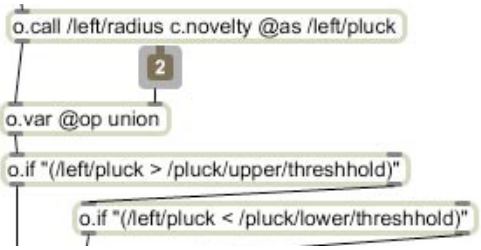
To complete the account here is the window function at the end of the feature detector:

```
p Difference
o.call /left/orientation > 0.001 @as /left/anticlockwise
o.call /left/orientation < -0.001 @as /left/clockwise
```

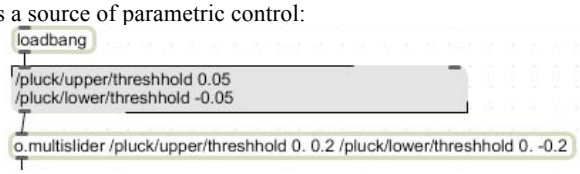
So far none of state of this computation is hidden. It is all available and traceable in the OSC bundles themselves. Although o.remember has to store a bundle internally, the stored contents are added to every outgoing bundle and flagged with the "/was" prefix. In the novelty detector of the next section we will stray slightly from this purely, functional approach but in a way that is still manageable.

7.7 Novelty Detector

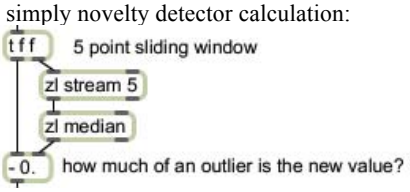
The basic algorithm is to clip the difference between the radial position of the left string and its median value inside a short sliding window.



As is typical of these simple detectors the threshholds of detection need to be adjusted using, for example, o.multisliders as a source of parametric control:



Here is the simply novelty detector calculation:



8. BlackBoxing

Hiding implementation details in modular, black boxes is a very effective technique but of little use unless the interfaces are documented. Our concatenating approach is interesting in that the name spaces can be designed so that the data itself emerging from the boxes describes the interface The packet emerging from the bottom of the example patch illustrates this.

```
/left/radius 1.194855
/was/right/y -0.967277
/was/left/radius 1.196872
/left/y -0.943346
/degrees/right/x -10.586081
/was/degrees/left/y -28.285715
/was/left/orientation 2.477973
/left/anticlockwise 1
/left/pluck -0.001591
/was/left/x 0.737241
/degrees/left/y -28.300365
/left/z 0.004640
/was/meters/left/z 0.008303
/left/orientation 2.480800
/delta/left/orientation 0.002828
/pluck/upper/threshhold 0.050000
/was/degrees/right/x -10.630037
/was/degrees/right/y -29.018314
/degrees/right/y -28.959707
/right/z 0.005372
/was/left/z 0.004151
/pluck/lower/threshhold -0.050000
/was/left/y -0.942857
/was/meters/right/z 0.009280
/was/degrees/left/x 22.117216
/right/x -0.352869
/was/right/z 0.004640
/right/y -0.965324
/degrees/left/x 22.000000
/left/clockwise 0
/meters/right/z 0.010745
/left/x 0.733333
/meters/left/z 0.009280
/was/right/x -0.354335
```

9. Conclusion

With the "o." library OSC messages are more than simply a new aggregate type for Max/MSP. They represent the glue necessary to integrate modern functional and object-oriented programming styles into a visual, dataflow language. Furthermore the time tags, atomicity and ordering semantics of OSC bundles promote productive development necessary for gesture signal processing and other reactive media programming applications.

10. Future Work

The "o." library has the foundational components to bring most modern programming paradigms to Max/MSP. Notable exceptions to this are reflectivity and parallelism. These both require significant changes in the Max kernel.

11. Dedication to Max Mathews

We dedicate this paper to the memory of Max Mathews who started us all out on computer languages for music and who mentored and inspired three generations of exciting work.

12. Acknowledgements

We gratefully acknowledge the support and encouragement of the McEnerney Endowment, Meyer Sound Laboratories, Pixar/Disney, the Concordia University Faculty of Fine Arts.

13. Bibliography

[1] Dony, C., Malenfant, J. and Bardou, D. Classifying Prototype-based Programming Languages. *Prototype-based Programming: Concepts, Languages and Applications*, 1998.

[2] Elliott, C. An embedded modeling language approach to interactive 3D and multimedia animation. *Software Engineering, IEEE Transactions on*, *25* (3). 291-308, 1999.

[3] Freed, A., McCutchen, D., Schmeder, A., Skriver Hansen, A.-M., Overholt, D., Burleson, W., Norgaard Jensen, C. and Mesker, A. Musical Applications and Design Techniques for the Gametrak Tethered Spatial Position Controller *SMC 2009*, 2009.

[4] Hansen, A.-M.S., Overholt, D., Burleson, W. and Jensen, C.N. Pendaphonics: a tangible pendulum-based sonic interaction experience *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*, ACM, Cambridge, United Kingdom, 2009.

[5] Lieberman, H. Using prototypical objects to implement shared behavior in object-oriented systems. *ACM SIGPLAN Notices*, *21* (11). 214-223, 1986.

[6] Magnusson, T. and Hurtado, E. The Phenomenology of Musical Instruments: A Survey. *eContact!* , *10.4*, 2008.

[7] Magnusson, T. and Mendieta, E., The acoustic, the digital and the body: A survey on musical instruments. in, (2007), ACM, 94-99.

[8] McKeehan, J. and Rhodes, N. *Programming for the Newton: software development with NewtonScript*. Academic Press Professional, Inc. San Diego, CA, USA, 1995.

[9] Noble, J., Taivalsaari, A. and Moore, I. *Prototype-Based Programming: Concepts, Languages and Applications*. Springer, 1999.

[10] Place, T. and Lossius, T., Jamoma: A modular standard for structuring patches in Max. in *Proceedings of the 2006 International Computer Music Conference*, (2006).

[11] Smith, W. Class-based NewtonScript programming. *PIE Developers*, 1994.

[12] Smith, W. SELF and the Origins of NewtonScript. *PIE Developers magazine, July*, 1994.

[13] Taft, S. and Duff, R. *Ada 95 reference manual: language and standard libraries: international standard ISO/IEC 8652: 1995 (E)*. Springer Verlag, 1997.

[14] Taivalsaari, A. Delegation versus concatenation or cloning is inheritance too. *SIGPLAN OOPS Mess.*, *6* (3). 20-49, 1995.

[15] Ungar, D. and Smith, R., Self: The power of simplicity. in, (1987), ACM, 227-242.

[16] Wright, M. and Freed, A., Open Sound Control: A New Protocol for Communicating with Sound Synthesizers. in *International Computer Music Conference*, (Thessaloniki, Hellas, 1997), International Computer Music Association, 101-104.

[17] Wright, M., Freed, A., Lee, A., Madden, T. and Momeni, A., Managing Complexity with Explicit Mapping of Gestures to Sound Control with OSC. in *International Computer Music Conference*, (Habana, Cuba, 2001), International Computer Music Association, 314-317.