USING DYNAMICALLY-BOUND STRUCTURES TO MANAGE PARAMETER PROPERTIES IN JAMOMA

First author
School
Department

Second author
Company
Address

Third author
Company
Address

ABSTRACT

Fundamental to the development of musical or artistic material is the ability to transform raw materials. This ability implies the facility to master many facets of the material, and to manipulate it with plasticity. Computer music environments typically provide points of control to manipulate material by providing parameters with controllable values. It is the experience of the authors that this capability to control the values of parameters is inadequate for many artistic endeavors and does not reflect the analogous tools and methods of artists working with physical materials.

A possible partial solution to this problem is to treat a parameter not as a single-value representing entity, but to treat a parameter as a multi-dimensional tool or object. Thus the parameter, as a tool or object, has many facets itself in addition to the value it renders. These many *properties* of the parameter define its behavior. The authors have explored a prototype of these ideas in Jamoma, a modular framework for Max/MSP/Jitter. In addition to defining behaviors, the behaviors are themselves interdependent upon each other requiring a flexible and dynamically bound code base for the implementation.

1. INTRODUCTION

Jamoma is a Model-View-Controller-based framework that provides a modular structure for the Max environment [**?**]. To be effective, it must use a messaging model that is dynamic, flexible, and efficient. The messaging model in Jamoma addresses modules using parameters (nodes with state) and messages (nodes which are stateless). In addition to providing values to parameters and messages, each node may also have its behavior defined or modified in real-time through the use of properties [**?**]. These properties define the behavior for messages and parameters by setting a value range, repetition filtering, the type of units used to express values, and how automation is applied.

Every node in Jamoma is addressable using Open Sound Control (OSC) [**?**][**?**].

The Jamoma Core comprises a number of external objects for the Max [**?**] environment, and a shared library which is accessible by any of the externals. Within the shared library, there are a series of "Libs" which provide key functionalities for developing multidimensional tools. These are:

- FunctionLib: a library of FunctionUnits which map an input value to an output value
- RampLib: a library of driving mechanisms (RampUnits) which use the FunctionLib to automate value transformations over time.
- DataspaceLib: a library by which a parameter or message can be given a class that describes the type of data it represents. Values may then be set by any of a number of DataspaceUnits to allow control of a parameter in any of a number of ways.

1.1. TTBlue

Talk about the basics that are provided by TTBlue in 1 paragraph.

Talk about how the libs are built on that, and why, in a 2nd paragraph.

2. FUNCTIONLIB

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

There are plans to introduce exponential functions.

3. RAMPLIB

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.

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 **??**. 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.

4. DATASPACELIB

It will be good for us to try and explain all the interactions here ;-)

4.1. TemperatureDataspace

La de da...

5. INTERDEPENDENCIES

This paper discusses the structure and development of the interrelated libraries that are used to implement this system.

String value	Numeric value
hello icmc	1073

Table 1. Table captions should be placed below the table

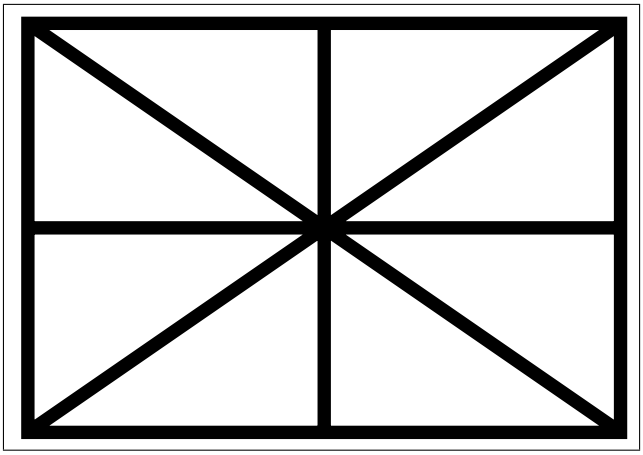

Figure 1. Figure captions should be placed below the figure

6. 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

7. CITATIONS

All bibliographical references should be listed at the end, inside a section named "REFERENCES", numbered and in alphabetic order. Also, all references listed should be cited in the text. When referring to a document, type the number in square brackets [1]

8. REFERENCES

[1] Author, E. "The title of the conference paper", *Proceedings of the International Computer Music Conference*, Miami, USA, 2004.

[2] Someone, A. *Title of the book*. Publisher, Belfast, 2007.