USING DYNAMICALLY-BOUND STRUCTURES TO MANAGE PARAMETER PROPERTIES IN JAMOMA

First author
School
Department

Second author
Company
Address

Third author
Company
Address

ABSTRACT

A prototype implementation in Jamoma is presented and discussed.

1. INTRODUCTION

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. However, we find that this capability to control the values of parameters is inadequate for many of our artistic endeavors, and does not reflect the analogous tools and methods of artists working with physical materials. For this reason we have started to explore alternatives ways of working with dynamic structures.

A possible partial solution 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. This paper presents a prototype of these ideas, implemented in Jamoma, a modular framework for Max/MSP [3]. In addition to defining behaviors, the behaviors are themselves interdependent upon each other requiring a flexible and dynamically bound code base for the implementation.

2. TERMINOLOGY

Before moving on we see the need to clarify some of the terminology being used...

Jamoma is a *Model-View-Controller* [1] framework [1], that provides a modular structure for the Max/MSP 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* (i.e. nodes with state) and *messages* (i.e. nodes which are stateless). For the duration of this paper we will often refer only to parameters, but the concepts generally apply to both parameters and messages.

[1] `http://en.wikipedia.org/wiki/Model-view-controller`

In addition to providing values to parameters, each node may also have its behavior defined or modified in real-time through the use of *properties* and *methods* [5]. These properties define the behavior for parameters by setting a value range, repetition filtering, the type of units used to express values, and how automation is applied.

2.1. Properties and Methods in Jamoma

Following is a list of the properties and methods of a parameter.

`:/value`	The value of the parameter.
`:/value/stepsize`	The size of the step taken by t
`:/value/inc`	Increase the value of the para
`:/value/dec`	Decrease the value of the para
`:/value/default`	The initial value of a paramet
`:/type`	The type of data represented b
`:/priority`	The value of the parameter.
`:/ui/freeze`	The value of the parameter.
`:/ui/refresh`	The value of the parameter.
`:/ramp/drive`	The value of the parameter.
`:/ramp/function`	The value of the parameter.
`:/repetitions`	The value of the parameter.
`:/range/bounds`	The value of the parameter.
`:/range/clip`	The value of the parameter.
`:/description`	The value of the parameter.
`:/node/type`	The value of the parameter.
`:/node/name`	The value of the parameter.
`:/dataspace`	The value of the parameter.
`:/dataspace/unit/active`	The value of the parameter.
`:/dataspace/unit/native`	The value of the parameter.

3. DYNAMIC BINDING IN JAMOMA

3.1. The Subscription System

Parameters are bound dynamically to a module. When a Max patcher loads a module, that module creates a *hub* object which acts as the controller for everything in the module. Then as parameters are created they subscribe to the hub to form a bi-directional connection for exchanging data. The existence of parameters may even change through the use of scripting over the course of time, and in response to changes in other parameters.

3.2. Parameter Components

In Jamoma, the parameter is implemented as a Max external called *jcom.parameter*. Within this jcom.parameter the ramp and dataspace properties are implemented internally as dynamically bound objects using the TTBlue framework [2]. The TTBlue framework is a C++ library that implements a dynamic messaging layer rather than using statically-linked C/C++ function calls [4].

By using dynamically linked components inside of the parameter, it is possible to switch between many different options for a give parameter property, while each option may implement an entirely different set of methods. This will be demonstrated in upcoming sections.

4. THE JAMOMA LIBRARIES

The functionality we refer to within a parameter is implemented in a shared library. Within the shared library the different functionalities are grouped in a series of "Libs". These libraries provide the key functionalities for developing multidimensional tools. To date, the libs are composed of:

- 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.

4.1. The Function Library

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.

[2] http://www.electrotap.com/ttblue/

4.2. The Ramp Library

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 4.1. 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.

4.3. The Dataspace Library

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

4.4. TemperatureDataspace

La de da...

5. INTERDEPENDENCIES

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

	Cosine	Linear	Lowpass	Power	Tanh
Scheduler		x			
Queue					
None					
Async					

Table 1. Table captions should be placed below the table

5.1. 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 does 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`

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

[2]

7. REFERENCES

[1] G. E. Krasner and S. T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, 1988.

[2] A. Momeni and D. Wessel. Characterizing and controlling musical material intuitively with geometric models. In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression*, pages 54–62, Montreal, Quebec, Canada, May 2003.

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

[4] T. Place and T. Lossius? Ttblue: A dynamically-bound dsp library in c++. In *Submitted to DAFX 2008*, Helsinki, Finland, 2008.

[5] T. Place, T. Lossius, A. R. Jensenius, N. Peters, and P. Baltazar. Addressing classes by differentiating values and properties in osc. In *Submitted to NIME 2008*, Genova, IT, 2008.