A FLEXIBLE AND DYNAMIC C++ FRAMEWORK AND LIBRARY FOR DIGITAL AUDIO SIGNAL PROCESSING

ABSTRACT

This paper presents an object-oriented, reflective, light-weight application programming interface for C++, with an emphasis on real-time signal processing. It makes use of polymorphic typing, dynamic binding, and introspection to create a cross-platform environment pulling ideas from languages such as Smalltalk and Objective-C while remaining within the bounds of the portable and cross-platform C++ context. The Jamoma Foundation and DSP Library provide a flexible framework and runtime environment, as well as an expanding collection of unit generators for synthesis, processing, and analysis. This library has been used in both open source and commercial software projects over the past seven years including Electrotap's Tap.Tools[1], Cycling '74's Hipno[2], and the Jamoma Modular Framework[3].

1. INTRODUCTION

"The SMC Roadmap identifies two broad research challenges: (1) To design better sound objects and environments and (2) To understand, model, and improve human interaction with sound and music." [12] The Jamoma DSP library directly addresses the first task as means by which to address the second task.

1.1. History

Came out of Tap.Tools and Hipno development. This was pre-Electrotap, so maybe the credit here should somehow go to 74Objects?

2. THE JAMOMA "PLATFORM"

We have to discuss all of the pieces here because they constantly come up later in the paper.

3. STRUCTURE / API

How is this whole beast structured.

[1] http://74objects.com/taptools
[2] http://74objects.com/hipno
[3] http://jamoma.org

3.0.1. An Example Class: the DC Blocker?

Or maybe a FunctionLib thing to show off the calculate method?

4. COMPONENTS

4.1. The Standard Libraries

The FilterLib The EffectsLib The MathLib DataspaceLib FunctionLib etc.

4.2. Unit Testing

baked right in (though currently on a branch that needs to be finished and merged to master...)

4.3. Serialization

I had started this code, but where is it now?

4.4. Ruby Language Bindings

This is cool

5. COMPARISON WITH OTHER DSP FRAMEWORKS

Or: Why in the world do we need another DSP framework?

5.1. Synthesis ToolKit

"The Synthesis ToolKit offers an array of unit generators for filtering, input/output, etc., as well as examples of new and classic synthesis and effects algorithms for research, teaching, performance, and composition purposes."[3]

Written in C++ and is crossplatform.

A defining difference between the STK and the Jamoma Foundation/DSP frameworks is that the STK is a statically-bound system based on method calls that are linked at compile-time. Jamoma's frameworks, on the other hand, are dynamically-bound and use a message passing system inspired by Smalltalk[7] and Objective-C[4].

As of version 4.2.0, the STK offers multichannel and frame-based sample processing[11]. Still, the StkFrames

type and tick() method operates on single sets of multichannel frames[4], whereas a Jamoma DSP's process method is designed to work with N multichannel signal frames of input and N multichannel frames of output.

At this time the STK offers two potential benefits over Jamoma DSP. First is the extensive library of unit generators, particularly for synthesis. The second is that due to static linking there is a slight performance advantage when making function calls rather than sending messages. On desktop computer the difference is unlikely to be discernible, but on mobile devices it is possible that the difference will become noticeable.

5.1.1. License

The license for the STK is reasonably liberal. At the time that the Jamoma DSP library was initially written, however, the license for the STK specified non-commercial use and thus using the STK was not an option for the authors.

5.2. CLAM

"CLAM stands for C++ Library for Audio and Music and it is a full-fledged software framework for research and application development in the audio and music domain with applicability also to the broader multimedia domain. It offers a conceptual model; algorithms for analyzing, synthesizing and transforming audio signals; and tools for handling audio and music streams and creating cross-platform applications."

"CLAM, a C++ software framework, that offers a complete development and research platform for the audio and music domain. It offers an abstract model for audio systems and includes a repository of processing algorithms and data types as well as all the nec- essary tools for audio and control input/output. The frame- work offers tools that enable the exploitation of all these features to easily build cross-platform applications or rapid prototypes for media processing algorithms and systems." [1]

Like Jamoma DSP, it is cross-platform (Mac, Linux, and Windows) and uses automatic integrated building, testing, versioning systems, and generally subscribes to Agile Development practices[5].

5.2.1. MetaModel

A key feature of CLAM is it's so-called "metamodel", which is a programming layer for creating hierarchical graphs of objects to produce a signal processing chain. Jamoma DSP does not define the way in which one must produce signal processing topographies. Instead, the process of creating objects and connecting them are envisioned and implemented orthogonally and the graph is created using a separate framework known as Jamoma Multicore.

Thanks to this decoupling of Jamoma DSP and Jamoma Multicore you aren't "boxed-in" to any particular way of creating connections between objects.

5.2.2. Class Design

Objects include "support for synchronous data processing and asynchronous event-driven control as well as a configuration mechanism and an explicit life cycle state model".

Jamoma DSP objects do this as well, via the "calculate" and "process" methods. Also, like CLAM, all objects provide an interface for working on data and support "metaobject-like facilities such as reflection and serialization".

Well, actually, nothing is synchronous or asynchonous in Jamoma DSP/Foundation. We are synchronously-agnostic. Multicore is cable of operating the lower-level DSP in a synchronous matter, as can MSP or Pd or AU etc...

Similar to CLAM, Jamoma Foundation objects have lifecycle state management. In Jamoma DSP an object can be flagged as valid and/or busy. CLAM differentiates between "controls" which are attributes that can be changed at any time and "configurations" which require the object to not be busy ("running" in CLAM parlance). Jamoma DSP does not make this distinction externally: both are simply considered "attributes" and the locking or other requirements are handled internally.

5.2.3. Object Networks

The Jamoma Foundation punts on this issue. CLAM does queing and connecting for data objects and processing objects etc. We consider this to be another layer, and so we don't address it directly in this paper.

What CLAM calls a 'composite', an object which itself instantiates other classes internally, is fully supported by Jamoma DSP.

5.2.4. API

Unlike CLAM, Jamoma DSP offers a simple and minimal Application Programming Interface. To be fair, this is in part due to the fact that creating connections and networks of objects is not a part of the Jamoma Foundation or DSP APIs. But we also have a flatter namespace, making only differentiation between attributes and messages,

5.2.5. Summary

Full featured, but very complex. It tries to be a framework, complete with visual editors, for building entire applications.

[4] as of version 4.4.1, downloaded from https://ccrma.stanford.edu/software/stk/download.html on 7 December 2009

[5] http://en.wikipedia.org/wiki/Agile$_{s}oftware_{d}evelopment$

In Jamoma we want simple and clear, while flexible underneath. We're a focused object runtime and DSP layer. Like Ruby on Rails we value convention over configuration. So if you can follow the default conventions you don't have to know all this complex stuff that is going on.

We also make a clear de-coupling between the framework for implementing unit generators and the framework that manages a graph of these unit generators – each of which may be freely interchanged.

5.3. Marsyas

Marsyas is a software framework for building efficient complex audio processing systems and applications [14]. "Audio processing systems are defined hierarchically through composition using implicit patching. Both the specification of the processing network and the control of it while data is flowing through can be performed at runtime without requiring recompilation."

"It is based on a dataflow model of computa- tion in which any audio processing system is represented as a large network of interconnected basic audio process- ing units." Just like Max/MSP, Pd, Chuck, etc.

One difference to Max/MSP and Pd is that the signal network can be reconfigured dynamically without requiring a 'recompile' of the signal chain. This is addressed through Jamoma Multicore – Jamoma DSP is low level and is agnostic about how objects are combined into a network or topology.

However, objects can be recombined and structured at runtime, offering the same kind of flexibility and "expressive power".

5.3.1. Implicit Patching

One thing that makes Marsyas special is its notion of "Implicit Patching". In this paradigm unit generators are added to a collection and their interaction with the signal processing graph is determined according to a pattern such as 'series', 'fanout', etc. [2].

Currently Jamoma DSP (and Multicore) operate solely through an 'explicit' patching paradigm similar to most other frameworks. "In explicit patching the user would first create the modules and then connect them by explicit patching statements." Due to the flexibility of the dynamically bound objects, however, it is quite easy to see how the implementation of pattern-based collections might be defined.

5.3.2. Dynamic Discovery and Access to Modules and Controls

In much the same way as Jamoma DSP, all control for a module are published and made accessible. What controls are available can be queried for – both for the name and the type [13].

In Marsyas the modules are organized in a heirarchy and address with an OSC-like string. This very similar to the work we are doing with Jamoma 0.6 and the NodeLib, but it's unclear how to state all of that here. What is cool is that you can find any DSP object instantiated in the system. It would be easy for us to implement this at the top level, but I'm not sure how we would determine the path structure NodeLib type of thing for them when objects instantiate other objects inside of them.

Both Marsyas and Jamoma DSP are able to extend existing objects by adding new controls or attributes at runtime to extend instances or create proxy controls. This is something that Objective-C can do, but that Max/MSP or Pd cannot do at this time.

5.3.3. User Interface Hooks

Marsyas is somewhat tightly bound to the Qt[6] toolkit for handling support of user interface integration with its classes. Jamoma DSP takes a different approach. The Jamoma Foundation implements at its core the ability to register observers for any class according to the Observer Pattern[5]. This has a number of benefits:

- makes it easier to tie into any user interface framework, or to make your own - doesn't require linking to third-party software to get threading help

5.3.4. MatLab Engine

Marsyas implements a singleton wrapper class for the MATLAB Engine API, enabling Marsyas developers to easily and conveniently send and receive data (i.e. in- tegers, doubles, vectors and matrices) to/from MATLAB in run-time. This is cool – Eno says "It is just a matter of work".

5.3.5. MaxMSP

Marsyas tries to create a whole runtime environment a graph inside of an external. This is different than the way we usually use the DSP Lib: we wrap the objects and then let MSP take care of the audio graph. In Multicore we we manage our own audio graph but use Max/MSP's patcher interface to control it be creating a set of peer objects. Do we somehow forward reference the DAFx paper here?

5.3.6. Bindings to other languages

Marsyas uses SWIG[7] to make control of it's runtime available to other programming environments. The Jamoma Foundation does not currently use SWIG, basically because I (tim) don't feel like figuring it out. We do however implement Ruby language bindings natively, and so we can offer

[6] http://www.trolltech.com/products/qt/

[7] http://www.swig.org

more natural and direct tie-ins to the Ruby language. At least that's the theory...

5.3.7. Additional Info..

Marsyas is released under the terms of the GNU GPL. This is fairly restrictive license, prohibiting commercial use. Jamoma is licensed under the GNU LGPL which allows commercial applications to be be created and distributed.

+ cross-platform, but - on windows it *requires* cygwin (we don't)

5.4. Music Kit

Is there anything interesting or still relevant about this besides the Objective-C thing? [6]

6. APPLICATIONS

6.1. Spatialization

Spatialization at the Encoding/Decoding and Authoring layers[9]. In fact, through the development of the NodeLib we are now able to work also at the Scene Description Layer by communicating as SpatDIF[8].

6.2. User Interface Development

Yup yup yup yup...

6.3. AudioUnit Creation

Blah blah blah...

6.4. External Object Generation for Max/MSP and PureData

Class wrappers... What about SuperCollider? I guess we should actually do it first before we claim that we can.

6.5. Ruby and Ruby on Rails

Server-side web application framework. :-)

6.6. ViMiC

In the summer of 2009 we combined the application domains of spatialization, graphical rendering, and AudioUnit creation to create a plug-in version of ViMiC[10].

6.7. Rapid Prototyping Environment

Because we can put together objects in the environment of our choice: Max, Pd, a DAW if we use AudioUnits, and then even a web-browser using Ruby on Rails. Then we can move the code from one environment to another easily, or port it back to C++ with minimal effort.

7. FURTHER DEVELOPMENT

7.1. UI

Automated UI editor generation through Jamoma Graphics.

7.2. Multicore

Not sure what to say here yet – need to do some assessment on Multicore (fun fun fun).

7.3. ClassWrappers

For more environments, such as VST and SuperCollider

7.4. Standard Library Expansion

Spectral processing
Granular processing
More effects (reverb, pitch-shifting, chorus)
SynthesisLib

7.5. Figures, Tables and Captions

String value	Numeric value
hello ICMC	1073

Table 1. Table captions are placed below the table.

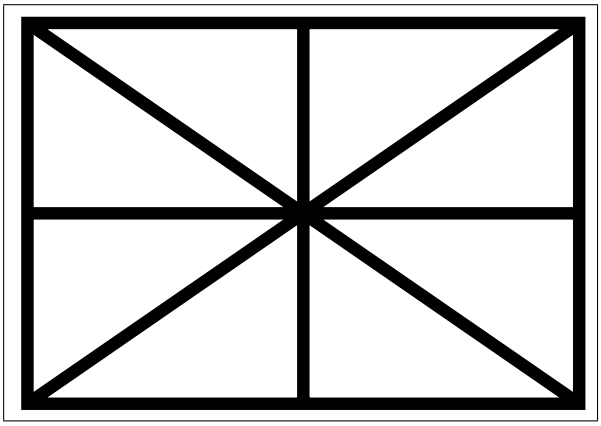

Figure 1. Figure captions are placed below the figure.

Place tables/figures in text as close to the reference as possible (see Figure 1 and Table 1). They may extend across both columns to a maximum width of 17.5cm (6.9").

8. EQUATIONS

Equations should be placed on separated lines and numbered. The number should be on the right side.

$$E = mc^2 \tag{1}$$

9. SUMMARY

Jamoma Foundation and DSP provide a flexible, user-extendable, runtime environment for creating and using audio and digital signal processing objects. Due to its advanced use of dynamic binding and message-passing paradigm, the building blocks can be reconfigured at runtime without requiring recompilation, but with the unit generators themselves compiled as C++ and performing block-processing we retain the performance of a compiled language.

The power of this runtime is demonstrated through the ability to compile objects for Max, Pd, AudioUnits, VST, etc. The future includes Multicore.

10. ACKNOWLEDGEMENTS

Dave Watson, Joshua Kit Clayton, BEK, GMEA

11. REFERENCES

[1] X. Amatraian, P. Arumi, and D. Garcia, "A framework for efficient and rapid development of cross-platform audio applications," *Multimedia Systems*, vol. 14, pp. 15–32, 2008.

[2] S. Bray and G. Tzanetakis, "Implicit patching for dataflow-based audio analysis and synthesis," in *Proceedings of the 2005 International Computer Music Conference*, 2005.

[3] P. Cook and G. Scavone, "The synthesis toolkit (stk)," in *Proceedings of the 1999 International Computer Music Conference*. Beijing, China: ICMA, 1999.

[4] B. J. Cox, *Object-Oriented Programming: An Evolutionary Approach*. Addison Wesley, 1987.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[6] D. Jaffe and L. Boynton, "An overview of the sound and music kits for the next computer," *Computer Music Journal*, vol. 13, no. 2, pp. 48–55, Summer 1989.

[7] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view-controller user interface paradigm in smalltalk-80," *JOOP*, August September 1988.

[8] N. Peters, "Proposing SpatDIF - the spatial sound description interchange format," in *Proceedings of the 2008 International Computer Music Conference*, Belfast, UK, 2008.

[9] N. Peters, T. Lossius, J. Schacher, P. Baltazar, C. Bascou, and T. Place, "A stratified approach for sound spatialization," in *Proceedings of 6th Sound and Music Computing Conference*, Porto, Portugal, 2009.

[10] N. Peters, T. Matthews, J. Braasch, and S. McAdams, "Spatial sound rendering in max/msp with vimic," in *Proceedings of the 2008 International Computer Music Conference*, Belfast, UK, 2008.

[11] G. Scavone and P. Cook, "Rtmidi, rtaudio, and a synthesis toolkit (stk) update," in *Proceedings of the 2005 International Computer Music Conference*. Barcelona, Spain: ICMA, 2005.

[12] X. Serra, "The origins of dafx and its future within the sound and music computing field," in *Proc. of the 10th Int. Conference on Digital Audio Effects (DAFx-07)*, 2007.

[13] G. Tzanetakis, *MARSYAS-0.2: a case study in implementing Music Information Retrieval Systems*. ?, 2006?

[14] G. Tzanetakis, R. Jones, C. Castillo, L. G. Martins, L. F. Teixeira, and M. Lagrange, "Interoperability and the marsyas 0.2 runtime," in *Proceedings of the 2008 International Computer Music Conference*, 2008.