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


## ABSTRACT

This paper presents an object-oriented, reflective, lightweight 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[18], and the Jamoma Modular Framework[17].


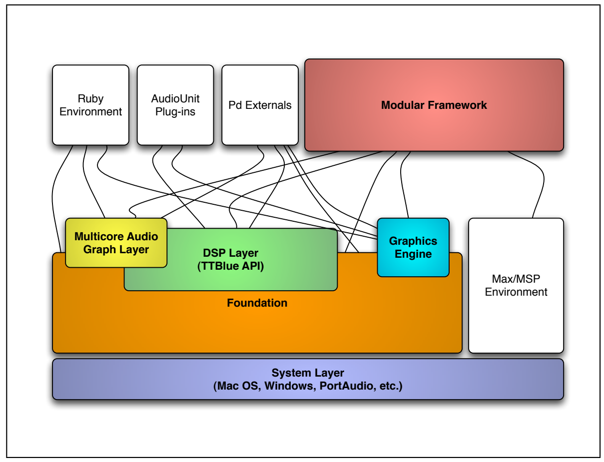


**Figure 1**. The Jamoma Platform as Layered Architecture.

## 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." [24] 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.

The Jamoma Platform comprises a number of initiatives working together (see Figure **??**).

## 3. STRUCTURE / API

### 3.1. Design

The design of the Jamoma Foundation and DSP Library strives to adhere to Dieter Rams' "ten principles for good design"[2]

- Good design is innovative
- Good design makes a product useful
- Good design is aesthetic
- Good design helps us to understand a product
- Good design is unobtrusive
- Good design is honest
- Good design is long-lasting
- Good design is consequent to the last detail
- Good design is concerned with the environment
- Good design is as little design as possible

### 3.2. The API

The basic calls: lifecycle, attributes, messages

[1] http://electrotap.com/taptools

[2] http://www.vitsoe.com/en/gb/about/dieterrams/gooddesign

#### 3.2.1. *Attributes*

Attributes are not merely maintaining the state of a single value but are multifaceted entities whose behavior is modified through the use of properties[19].

#### 3.2.2. *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?

First, let's limit ourselves to the C family of languages, which is to say C, C++, Objective-C. These languages are well established, capable of producing efficient compiled binaries, widely available, and very portable. One consideration is that we wish for the DSP library to be capable of running not only in the desktop context buy also on embedded processors and mobile devices. Many languages, such as D, are not available for many embedded systems environments. The same applies to Java. Interpreted languages also present challenges for some embedded processors.

That said, we also wish to see what we can learn from DSP libraries created for other language families. JSyn is perhaps the most well-known of several initiatives based on the Java language[8, 3]. JSyn is particularly adept at running in Web browsers. One of the drawbacks of JSyn is the somewhat ambiguous and anti-commerical nature of the licensing.

### 5.1. Frameworks vs. Environments

Jamoma Foundation/DSP is not an environment. It is a framework that you can use to create an environment, or to extend an existing environment, but it is not itself and environment. That same can be said of the STK but other examples here, such as Marsyas and CLAM, are really full environments in the same way the ChucK or PureData is an environment.

### 5.2. 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."[4]

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[11] and Objective-C[5].

As of version 4.2.0, the STK offers multichannel and frame-based sample processing[23]. Still, the StkFrames type and tick() method operates on single sets of multichannel frames[3], 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.

#### 5.2.1. *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.

### 5.3. 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

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

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[4].

#### *5.3.1. 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.

#### *5.3.2. 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 life-cycle 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.

#### *5.3.3. 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.

#### *5.3.4. 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,

#### *5.3.5. Summary*

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

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.

### 5.4. Marsyas

Marsyas is a software framework for building efficient complex audio processing systems and applications [26]. "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.

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

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

#### 5.4.1. *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.

#### 5.4.2. *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 [25].

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.

#### 5.4.3. *User Interface Hooks*

Marsyas is somewhat tightly bound to the Qt[5] 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[7]. 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

#### 5.4.4. *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”.

#### 5.4.5. *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?

#### 5.4.6. *Bindings to other languages*

Marsyas uses SWIG[6] 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 more natural and direct tie-ins to the Ruby language. At least that's the theory...

#### 5.4.7. *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)

### 5.5. NeXT Music Kit

The NeXT MusicKit, together with the SndKit, provides a library of objects for DSP and MIDI applications written in the Objective-C language[9, 10]. The Music Kit comprises not only a layer for creating audio and midi unit generators but also a hardware abstraction layer. Despite the demise of NeXT Computer, the Music Kit continues to be maintained and updated to work on current operating systems that support the Objective-C Runtime[7].

As a framework implemented in Objective-C, Music Kit inherits the reflective, object-oriented, dynamically-bound message passing API that is implemented by the Jamoma Foundation. However, building on top of Objective-C instead of C++ also presents portability challenges. Jamoma Foundation/DSP should run on a myriad of embedded and mobile devices, and also natively compile using Microsoft's

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

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

[7] http://sourceforge.net/projects/musickit/

MSVC compiler. These tasks are not easily performed, if performed at all, when using the Objective-C language[8].

### 5.6. The Max Family

The Max family includes both MSP[28] and PureData[21].

"The Max paradigm can be described as a way of combining pre-designed building blocks into configurations useful for real-time computer music performance."[22]

This description is more similar to Multicore than to DSP. In fact, The Max environments also define APIs for creating unit generators and provide libraries of these unit generators (or "pre-designed building blocks"). The environments also define an API for message passing in much the same way that the Jamoma Foundation provides.

### 5.7. Domain-Specific Text-Based Languages

These are similar to the Max Family except that they create networks of Unit Generators using a text interface. I don't really want to talk about CSound. Is that really neccesary? Can I just mention CSound and SuperCollider together in one sentence?

#### *5.7.1. SuperCollider*

Is there anything particularly pertinent here?[12]

SuperCollider combines the unit generators, the audio engine, and an object-oriented language with semantics similar to C and Smalltalk. SuperCollider is incredibly powerful because of the language constructs and idioms that manipulate the unit generators. The API for creating unit generators is similar to many other environments we have reviewed.

#### *5.7.2. ChucK*

Probably a bit more here that is pertinent. Ge Wang might balk at calling ChucK a domain-specific language, but that's what it really is.

"the design of ChucK strives to hide the mundane aspects of pro- gramming, and expose true control"[27].

Unfortunately, ChucK is not suitable for use by Jamoma due to the excessive restrictiveness of the GNU GPL, under which it is licensed.

To it's own admission, execution speed is not the primary priority for ChucK. As such it does not perform frame-based signal computation but computes at every sample. This contributes to ChucK's strong timing model, but at the expense of slower number-crunching for audio.

### 5.8. JSyn

Like the STK but for Java. Need Source.

[8] The authors also look upon Apple's corporate control of the Objective-C language and runtime with some degree of skepticism, as compared to the open consortium that is mediating the C++ language's evolution

### 5.9. Audio Plug-ins

VST, RTAS, LADSSPA, and AudioUnits are all, in essence, APIs for creating UnitGenerators. Somehow it doesn't seem like they should count though. Why is that?

### 5.10. Kronos

This expresses the problem domain well:

" the musician may want to change the program during its execution. This was possible in the analog music studio, where swapping out patch cords often resulted in immediate gratification. In the digital world programs often have to be aborted, edited, re- compiled, linked and launched. The all-important musical hacking suffers from such a heavy compilation cycle, making a traditional programming language less desirable for real time artistic expres- sion." [13]

The Jamoma Foundation and other dynamically-bound frameworks, such as PureData, Objective-C, or Marsyas, solve this problem by precompiling the unit generators and then directing messages to these objects at runtime. Kronos takes an alternative approach where the graph of objects, indeed even the unit generators themselves, are not precompiled at all but rather compiled 'Just in Time' from a custom meta-language.

This results in better performance from the code, while still maintaining much of the flexibility offered by a dynamically-bound runtime. Indeed, the published performance results are convincing. However, I [tim] remain skeptical of the true runtime-flexibility, as just-in-time compilation still requires compilation every time you change the interconnections between objects.

## 6. APPLICATIONS

### 6.1. Spatialization

Spatialization at the Encoding/Decoding and Authoring layers[15]. In fact, through the development of the NodeLib we are now able to work also at the Scene Description Layer by communicating as SpatDIF[14].

### 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[16].

### 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. Scheduler

Initial work on a scheduler for the DSP library was begun in 2007. Competing priorities have left it unfinished... More interesting models for scheduling, such as the the model implemented in ChucK provide impetus for further research into new approaches to this topic.

### 7.6. Tools

Unit testing Integrated Analysis and Benchmarking – perhaps reference the upchuck operator in ChucK?

### 7.7. Web Browser Support

We have begun implementing support for server-side integration of the Jamoma Runtime via the Ruby language bindings and Ruby on Rails[9]. To fully leverage the DSP library and Jamoma Foundation in a web-browser we need the ability to invoke the runtime on the client-side of the equation through a web browser plug-in similar to that done in the iARS Project[6].

### 7.8. NodeLib

As a means of addressing parts of the system via OSC. Somewhat like what Marsyas does. An implementation of the ideas presented in[20].

### 7.9. Figures, Tables and Captions

| String value | Numeric value |
|---|---|
| hello ICMC | 1073 |

**Table 1**. Table captions are placed below the table.

## 8. 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.

## 9. ACKNOWLEDGEMENTS

Dave Watson, Joshua Kit Clayton, Tho Delahogue

## 10. 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.

[9] http://rubyonrails.org

[3] P. Burk, “Jsyn – a real-time synthesis api for java,” in *Proceedings of the 1998 International Computer Music Conference*, 1998s.

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

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

[6] C. Frauenberger and W. Ritsch, “A real-time audio rendering system for the internet (iars), embedded in an electronic music library (iaem),” in *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03)*, 2003.

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

[8] M. Guillemard, C. Ruwwe, and U. Zölzer, “J-dafx - digital audio effects in java,” in *Proc. of the 8th Int. Conference on Digital Audio Effects (DAFx-05)*, 2005.

[9] 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.

[10] D. A. Jaffe, “Musical and extra-musical applications of the next music kit,” in *Proceedings of the 1991 International Computer Music Conference*. NeXT Computer Inc., 1991.

[11] 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.

[12] J. McCartney, “Supercollider: a new real time synthesis language,” in *Proceedings of the 1996 International Computer Music Conference*, 1996, p. 3.

[13] V. Norilo and M. Laurson, “Kronos - a vectorizing compiler for music dsp,” in *Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx-09)*, 2009.

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

[15] 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, pp. 219–224.

[16] 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.

[17] T. Place and T. Lossius, “Jamoma: A modular standard for structuring patches in Max,” in *Proceedings of the 2006 International Computer Music Conference*, New Orleans, US, 2006.

[18] T. Place, N. Wolek, and J. Allison, *Hipno: Getting Started*, Cycling’74 and Electrotap, 2005. [Online]. Available: http://www.cycling74.com

[19] T. Place, T. Lossius, A. R. Jensenius, and N. Peters, “Flexible control of composite parameters in max/msp,” in *Proceeding of the International Computer Music Conference*, T. I. C. M. Association, Ed., 2008, pp. 233–236.

[20] T. Place, T. Lossius, A. R. Jensenius, N. Peters, and P. Baltazar, “Addressing Classes by Differentiating Values and Properties in OSC,” in *Proceedings of the 2008 Conference on New Interfaces for Musical Expression (NIME-08)*, Genova, Italy, 2008, pp. 181–184.

[21] M. Puckette, “Pure data: another integrated computer music environment.” in *Proceedings of the 1996 International Computer Music Conference*, 1996.

[22] ——, “Max at seventeen,” *Computer Music Journal*, vol. 26, no. 4, pp. 31–43, 2002.

[23] 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.

[24] 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.

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

[26] 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.

[27] G. Wang, “The chuck audio programming lanuage: A strongly-timed and on-the-fly envon/mentality,” Ph.D. dissertation, Princeton University, 2008.

[28] D. Zicarelli, “An extensible real-time signal processing environment for max,” in *Proceedings of the 1998 International Computer Music Conference*. Ann Arbor, Michigan, USA: San Francisco: ICMA, 1998, pp. 463–466.