CLAM: A Framework for Efficient and Rapid Development of Cross-platform Audio Applications

Xavier Amatriain
CREATE
University of California Santa Barbara
Santa Barbara, CA
xavier@create.ucsb.edu

Pau Arumi
Music Technology Group, UPF
c/Ocata, 1
08003 Barcelona, Spain
parumi@iua.upf.edu

David Garcia
Music Technology Group, UPF
c/Ocata, 1
08003 Barcelona, Spain
dgarcia@iua.upf.edu

ABSTRACT

CLAM is a C++ framework that offers a complete development and research platform for the audio and music domain. Apart from offering an abstract model for audio systems, it also includes a repository of processing algorithms and data types as well as a number of tools such as audio or MIDI input/output. All these features can be exploited to build cross-platform applications or to build rapid prototypes to test signal and media processing algorithms and systems. The framework also includes a number of stand-alone applications that can be used for tasks such as audio analysis/synthesis, plug-in development or metadata annotation.

In this article we give a brief overview of CLAM's features and applications.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Domain-specific architectures

General Terms

Design

Keywords

Frameworks, Multimedia, Audio

1. INTRODUCTION

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. 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.

The CLAM framework is cross-platform. All the code is ANSI C++ and it is regularly compiled under GNU/Linux, Windows and Mac OSX.

CLAM offers a processing kernel that includes an *infrastructure* and processing and data *repositories*. In that sense, CLAM is both a *black-box* and a *white-box* framework. It is black-box because already built-in components included in the repositories can be connected with minimum programmer effort in order to build new applications. And it is *white-box* because the abstract classes that make up the infrastructure can be easily derived to extend the framework components with new processes or data classes.

CLAM also includes a number of tools for services such as audio i/o or XML serialization and applications that have served as a testbed and validation of the framework.

The CLAM infrastructure is a direct implementation of the 4MS metamodel, which will be explained in the following section. In the next sections we will also review the CLAM repositories, its tools and applications. Please refer to CLAM's website (www.clam.iua.upf.edu) for further information, documentation, and downloads.

2. CLAM'S METAMODEL

The Object-Oriented Metamodel for Multimedia Processing, 4MS for short, provides the conceptual framework (metamodel) for a hierarchy of models of media data processing system architectures in an effective and general way. The metamodel is an abstraction of many ideas found in the CLAM framework but also of an extensive review of similar frameworks and collaborations with their authors. Although derived and based in particular for audio and music frameworks, it presents a comprehensive conceptual framework for media signal processing applications. For a more detailed description of the metamodel and how it relates to different frameworks see Xavier Amatriain's PhD [1] .

The 4MS metamodel is based on a classification of signal processing objects into two categories: *Processing* objects that operate on data and control, and *Processing Data* objects that passively hold media content. Processing objects encapsulate a process or algorithm; they include support for synchronous data processing and asynchronous event-driven control as well as a configuration mechanism and an explicit life cycle state model. On the other hand, Processing Data objects offer a homogeneous interface to media data, and support for meta object facilities such as reflection and serialization.

Although the metamodel clearly distinguishes between two different kinds of objects the managing of Processing Data constructs can be almost transparent for the user. Therefore, we can view a 4MS system as a set of Processing objects connected in a graph called *Network*.

Processing objects are connected through intermediate channels. These channels are the only mechanism for communicating between Processing objects and with the outside world. Messages are enqueued (produced) and dequeued (consumed) in these channels, which act as FIFO queues.

MM'06, October 23–27, 2006, Santa Barbara, California, USA.
ACM 1-59593-447-2/06/0010.

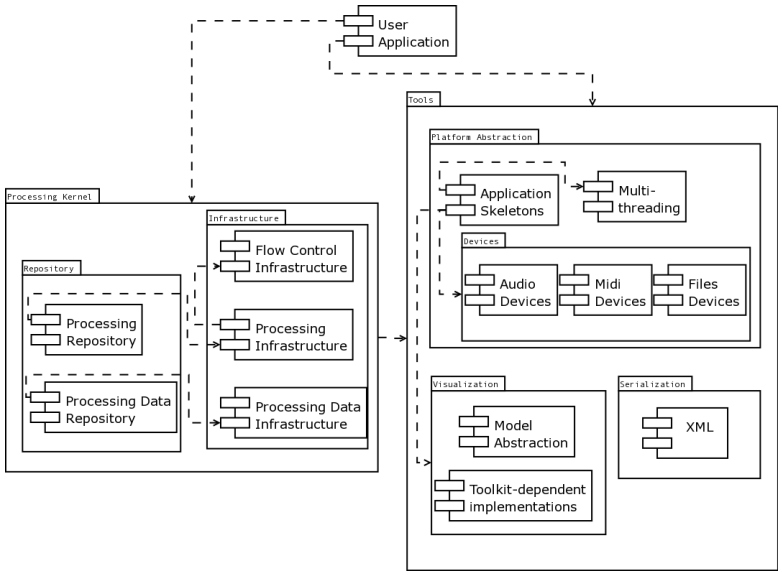

Figure 1: The CLAM framework components

The metamodel offers two kinds of connection mechanisms: *ports* and *controls*. Ports transmit data and have a synchronous data flow nature while controls transmit events and have an asynchronous nature. By synchronous, we mean that messages get produced and consumed at a predictable —if not fixed— rate. And by asynchronous we mean that such a rate doesn't exist and the communication follows an event-driven schema.

But apart from the incoming and outcoming data, some other entity —probably the user through a GUI slider— might want to change some parameters of the algorithm. This control events will arrive, unlike the audio stream, sparsely or in bursts. In this case the processing object will receive these events through various (input) control channels: one for the gain amount, another for the frequency, etc.

The data flows through the ports when a processing is fired (by receiving a *Do()* message).

Processing objects can consume and produce at different rates and consume an arbitrary number of tokens at each firing. Connecting these processing objects is not a problem as long as the ports are of the same data type. The data flow is handled by a *FlowControl* entity that figures out how to schedule the firings in a way that avoids firing a processing with not enough data in its input ports or not enough space into its output ports.

3. REPOSITORIES

The *Processing Repository* contains a large set of ready-to-use processing algorithms, and the *Processing Data Repository* contains all the classes that act as data containers to be input or output to the processing algorithms.

The Processing Repository includes around 150 different Processing classes, classified in the following categories: Analysis, ArithmeticOperators, AudioFileIO, AudioIO, Controls, Generators, MIDIIO, Plugins, SDIFIO, Synthesis, and Transformations. Although the repository has a strong bias toward spectral-domain processing because of our group's background and interests, there are enough encapsulated algorithms and tools so as to cover a broad range of possible applications.

On the other hand, in the Processing Data Repository we offer the encapsulated versions of the most commonly used data types such as Audio, Spectrum, SpectralPeaks, Envelope or Segment. It is interesting to note that all of these classes make use of the data infrastructure and are therefore able to offer services such as a homogeneous interface or built-in automatic XML persistence.

4. TOOLS

Apart from the infrastructure and the repositories, which together make up the CLAM *processing kernel* CLAM also includes a number of tools that can be necessary to build an audio application.

4.1 XML

Any CLAM *Component* can be stored to XML. Furthermore, Processing Data and Processing Configurations make use of a macro-derived mechanism that provides automatic XML support without having to add a single line of code [4].

4.2 GUI

When designing CLAM we had to think about ways of integrating the core of the framework tools with a graphical user interface that may be used as a front-end to the framework functionalities. CLAM offers a toolkit-independent support through the CLAM Visualization Module. This general Visualization infrastructure is completed by some already implemented presentations and widgets. These are offered both for the FLTK toolkit and the Trolltech's Qt framework . An example of such utilities are convenient debugging tools called Plots. Plots offer ready-to-use independent widgets that include the presentation of the main Processing Data in the CLAM framework such as audio, spectrum, spectral peaks...

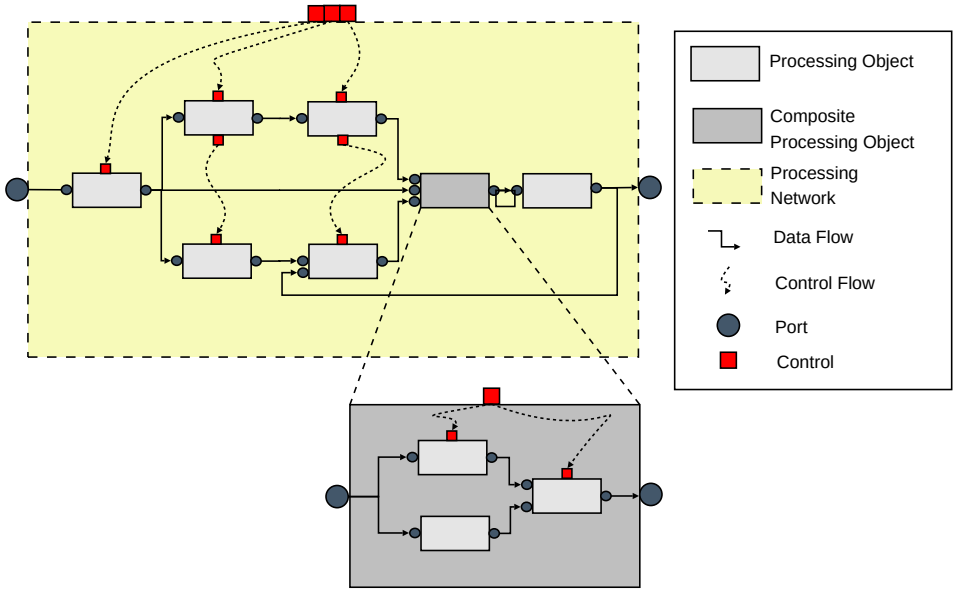

Figure 2: a 4MS processing network

4.3 Platform Abstraction

Under this category we include all those CLAM tools that encapsulate system-level functionalities and allow a CLAM user to access them transparently from the operating system or platform.

Using these tools a number of services –such as Audio input/output, MIDI input/output or SDIF file support– can be added to an application and then used on different operating systems without changing a single line of code.

5. APPLICATIONS

The framework has been tested on —but also has been driven by— a number of applications. Most of them will be introduced in the following paragraphs. The last subsection shows the CLAM visual building tools. Though initially considered separate applications, they now allow visually building applications without writing any line of code thus becoming part of the framework.

5.1 SMS Analysis/Synthesis

The main goal of the application is to analyze, transform and synthesize back a given sound using the Sinusoidal plus Residual model [2]. In order to do so the application reads an XML configuration file, and an audio file or a previously analyzed sdif file. The input sound is analyzed, transformed in the spectral domain according to a transformation score and then synthesized back.

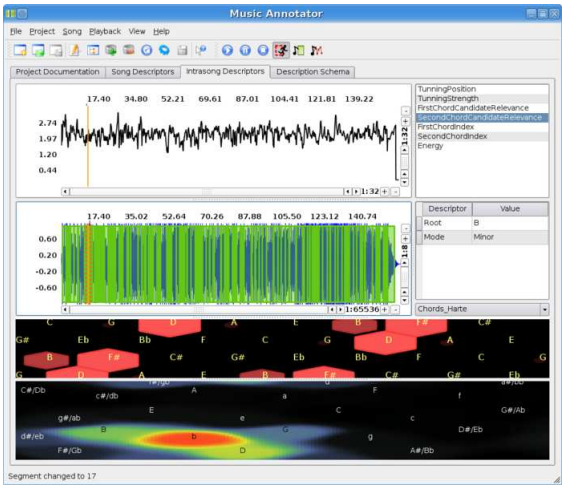

Figure 3: Editing low-level descriptors and segments with the CLAM Music Annotator

5.2 The Music Annotator

The CLAM Music Annotator [3] is a tool for editing audio descriptors. The application can be used as a platform for launching extraction algorithms that analyze the signal and produce different kinds of descriptors. These processes can be either local or web services. But most importantly, the Annotator includes a powerful GUI to manually edit the result of these algorithms from the audio sample level to the song level.

5.3 SALTO

SALTO is a software based synthesizer that is also based on the Sinusoidal plus Residual technique. It implements a general architecture for these synthesizers but it is cur-

rently only prepared to produce high quality sax and trumpet synthesis. Pre-analyzed data are loaded upon initialization. The synthesizer responds to incoming MIDI data or to musical data stored in an XML file. SALTO can be used as a regular synthesizer on real-time as it accepts messages coming from a regular MIDI keyboard or a MIDI breath controller.

5.4 Spectral Delay

SpectralDelay is also known as CLAM's Dummy Test. In this application it was not important to actually implement an impressive application but rather to show what can be accomplished using the CLAM framework. The SpectralDelay implements a delay in the spectral domain, dividing the audio signal into three bands and allowing for each band to be delayed separately.

5.5 Others

Apart from the main sample applications CLAM has been used in many different projects that are not included in the public version either because the projects themselves have not reached a stable stage or because their results are protected by non-disclosure agreements with third parties. In the following paragraphs we will outline these other users of CLAM.

Rappid was a testing workbench for the CLAM framework in high demanding situations. Rappid was tested in a live-concert situation. Rappid was used as an essential part of a composition for harp, viola and tape, presented at the Multiphonies 2002 cycle of concerts in Paris.

The Time Machine project implemented a high quality time stretching algorithm that was later integrated and included in a commercial product. The algorithm uses multi-band processing and works in real-time. It is a clear example of how the core of CLAM processing can be used in isolation as it lacks of any GUI or audio input/output infrastructure.

The Vocal Processor is VST plug-in for singing voice transformations. This prototype was a chance to test CLAM integration into VST API and also to check the efficiency of the framework in highly demanding situations. Most transformations are implemented in the frequency domain and the plug-in must work in real-time, consuming as few resources as possible.

The CUIDADO IST European project was completely developed with CLAM. The focus of the project was on automatic analysis of audio files. In particular rhythmic and melodic descriptions were implemented. The Open Drama project was another IST European project that used CLAM extensively. The project focus was on finding new interactive ways to present opera. In particular, a prototype application was built to create an MPEG-7 compliant description of a complete opera play.

6. CLAM AS A RAPID PROTOTYPING ENVIRONMENT

The latest developments in CLAM have brought *visual building* capabilities into the framework. These allow the user to concentrate on the research algorithms and not on application development. Visual patching is also valuable for rapid application prototyping of applications and audio-plug-ins.

CLAM's visual builder is known as the NetworkEditor (see Figure 4). It allows to generate an application—or its processing engine—by graphically connecting objects in a patch. Another application called Prototyper acts as the glue between a graphical GUI designing tool (such as qt Designer) and the processing engine defined with the NetworkEditor.

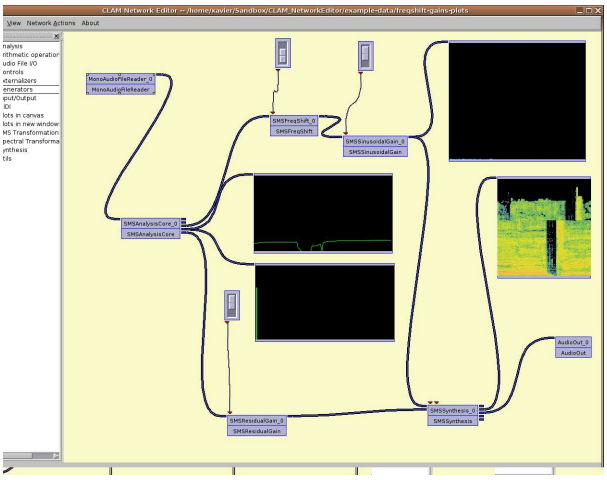

Figure 4: NetworkEditor, the CLAM visual builder

7. CONCLUSIONS

As seem in the previous sections CLAM has proven useful in many applications and is becoming more and more easy to use even before it has reached its first stable 1.0 release (planned for the end of 2006). And so, we expect new projects to begin using the framework .

8. ACKNOWLEDGEMENTS

The authors wish to recognize all the people who have contributed to the development of the CLAM framework. A non-exhaustive list should at least include Maarten de Boer, Ismael Mosquera, Xavier Oliver,Miguel Ramírez, Enrique Robledo and Xavi Rubio.

9. REFERENCES

[1] X. Amatriain. *An Object-Oriented Metamodel for Digital Signal Processing with a focus on Audio and Music*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Spain, 2005.

[2] X. Amatriain, J. Bonada, A. Loscos, and X. Serra. *DAFX: Digital Audio Effects (Udo Zölzer ed.)*, chapter Spectral Processing, pages 373–438. John Wiley and Sons, Ltd., 2002.

[3] X. Amatriain, J. Massaguer, D. Garcia, and I. Mosquera. The clam annotator: A cross-platform audio descriptors editing tool. In *Proceedings of the 2005 International Symposium on Music Information Retrieval, ISMIR '05*, 2005.

[4] D. Garcia and X. Amatrian. XML as a means of control for audio processing, synthesis and analysis. In *Proceedings of the MOSART Workshop on Current Research Directions in Computer Music*, Barcelona, Spain, 2001.