CLAM, Yet Another Library for Audio and Music Processing?

Xavier Amatriain
Music Technology Group
Pompeu Fabra University
Barcelona, Spain
xamat@iua.upf.es

Pau Arumí
Music Technology Group
Pompeu Fabra University
Barcelona, Spain
parumi@iua.upf.es

Miguel Ramírez
Music Technology Group
Pompeu Fabra University
Barcelona, Spain
mramirez@iua.upf.es

ABSTRACT

CLAM (C++ Library for Audio and Music) is a framework that aims to offer extensible, generic and efficient design and implementation solutions for developing Audio and Music applications as well as for doing more complex research related with the field. Although similar libraries exist, some particularities make CLAM of high interest for anyone interested in the field.

Keywords

Development framework, DSP, audio, GPL, Free Software, XML

1. INTRODUCTION: OO DOES NOT MEAN INEFFICIENT

The Music Technology Group of the Pompeu Fabra University [1] is a research group where more than forty engineers and programmers are involved in different projects focused on the development of algorithms and applications for Signal and Music Processing. Most of these are implemented in C++.

Two years ago, it was clear that the amount and quality of the lines of code related with different projects was becoming hardly manageable. Although the code had been written using an OO-might-be language[2] and some classes existed here and there, basic design principles had not been observed and the result was highly unstructured, difficult to understand and hardly reusable. Flexibility and reusability had been sacrificed in the sake of performance efficiency. This fact made extremely difficult and time-consuming to integrate newcomers or new projects into the group. Bearing those ideas in mind, the CLAM project was started [3]. Since then, an average of six programmers-developers have been working on it.

For designing the framework we had to fight against an idea that still has high acceptance in the DSP community: unstructured C-like code assures high performance at run-time. One of the main goals of our work has been to prove that a clean and structured design in general, and OO techniques in particular, do not have to imply an overhead in computational efficiency nor a limitation in flexibility for implementing different models belonging to a particular domain.

The framework has been already used successfully for a number of internal projects – some of them with high run-time performance requirements- like high quality time-stretching, sax synthesizer and high-level feature analysis[4], and it seems to have reached a somewhat mature stage. The project is also due to see its first public release in November 2002, in the course of the AGNULA IST European project[5].

2. ANOTHER AUDIO LIBRARY ?

What makes CLAM different from other similar solutions that already exist? (see references [6] thru [10] to find out about "related" projects)

To begin with, CLAM is truly object oriented. Extensive software engineering techniques [11] have been applied in order to design a framework that is both highly (re)usable and understandable. Although the term "sound object" has been around for many years[12], and OO techniques have also been applied in many audio and music related applications[13], none of these have conceptually applied the "everything is an object" maxima[14]. In our framework, all data types and processing or flow control entities are objects (see [15] for a conceptual background of the CLAM framework).

CLAM is comprehensive since it not only includes classes for processing but also for audio and MIDI input/output, XML serialization services, algorithm and data visualization and interaction, and multithreading handling.

CLAM deals with a wide variety of extensible data types that range from low-level signals (such as audio or spectrum) to higher-level semantic structures (such as musical phrase or segment).

The framework is cross-platform. All the code is ANSI C++ and it is regularly compiled under Linux, Windows and MacOS using the most commonly used compilers. Even the code for input/output, visualization and multithreading is cross-platform down to the lowest possible layer.

The project is licensed under the GPL (GNU Public License) terms and conditions[16]. Although we maintain the option of double licensing the framework (i.e. offering an alternative commercial license), everything offered in the public version will be GPL and the project will thus become free software, open-source, and collaborative.

CLAM is bound to survive. Even though its public success is by no means guaranteed, CLAM will surely remain the basis for all future developments in the MTG and thus will be maintained and updated on a regular basis.

CLAM is designed to offer two different modes. In the non-supervised mode CLAM is used as a regular C++ library allowing extension, flexibility and user-controlled optimization. The other mode (supervised) is intended to work as a rapid prototyping application with automatic scheduling and flow control. Although the latter is still not fully functional, many design decisions have been driven by its compatibility.

3. CLAM'S COMPONENTS

CLAM brings the world of software design and engineering to DSP developers who could care less about it. For doing so, it offers some general infrastructure like ADT's, XML serialization, or a GUI module. But, most importantly, it forces users to follow some "good coding principles" and it provides a general model for easy (re)usability.

Nevertheless, the main trait of CLAM is the ability to process multiple data types related to the audio and music domain. All these data types are subclasses of the *ProcessingData* class. The main goals when designing this class hierarchy can be summarized in: (1) Force and automatically derive a common interface for all data classes without much programmer's effort; (2) Implement a tree-like structure or composite pattern[17] in order to offer automatic serialization services; (3) Allow class attributes to be dynamically instantiated at run-time without forcing the use of less-efficient and more error prone C++ pointers.

The solution was to implement a base class that combines the use of C macros with OO techniques like template meta-programming and static dispatching. This way, all the above objectives are accomplished with a minimum programmer's effort. On the other hand, object serialization has been accomplished using XML as a general-purpose format[18]. The decision of using this format has mainly been influenced by the recent upcoming of the MPEG-7 standard for multimedia description[19].

ProcessingData objects are used as the only possible inputs and outputs to *Processing* objects. All processing in CLAM is performed inside a *Processing* object. The *Processing* classes encapsulate DSP algorithms and the available base class hierarchy offers services for synchronous data flow and asynchronous event-driven flow, scalability, interconnection and state queries, hiding most of the complexity from the library user.

Finally, the GUI module implements a model abstraction based on a modified version of the MVC called Model-View-Presentation. The decoupling between the view and the presentation is accomplished through the implementation of a template functor based callback library[20]. It also implements some particular presentations for basic data types (like audio or spectrum) using FLTK[21] and OpenGL.

4. ACKNOWLEDGEMENTS

The work reported in this paper has been partially funded by the IST European programs AGNULA and CUIDADO.

5. REFERENCES

[1] UPF's MTG homepage: http://www.iua.upf.es/mtg

[2] Stroustrup, Bjarne. Why C++ is not only an object-oriented programming language. In OOPSLA'95 Proceedings

[3] CLAM website: http://www.iua.upf.es/mtg/clam

[4] Amatriain, X.; de Boer, M.; Robledo, E.; García, D. CLAM: An OO Framework for Developing Audio and Music Applications. In OOPSLA 2002 Proceedings (Companion Material). Seattle, 2002.

[5] AGNULA website: http://www.agnula.org

[6] Pope, S. T. The Siren Music/Sound Package for Squeak Smalltalk. In OOPSLA'98 Proceedings.

[7] The SndObj homepage: http://www.may.ie/academic/music/musictec/SndObj/

[8] OSW: Open Sound World homepage: http://osw.sourceforge.net/

[9] Jmax homepage: http://www.ircam.fr/equipes/temps-reel/jmax/

[10] Pure Data (PD) homepage: http://www.pure-data.org/

[11] Sommerville, I. Software Enginnering 6th Edition. Pearson Ed. August 2000.

[12] Schaeffer, Pierre; Traité des Objets Musicaux. Editions Du Seuil. 1966.

[13] Pope, Stephen Travis (ed). The well-tempered object, Musical Applications of Object-Oriented Technology. MIT Press. 1991.

[14] Kay, A. The Early History of Smalltalk. In Proceedings of 2nd ACM SIGPLAN History of Programming Languages Conference. ACM SIGPLAN Notices 28(3): 69-75. 1993

[15] Amatriain, X.; Herrera, P. Transmitting Audio Content as Sound Objects. In Proceedings of AES 22nd Conference on Virtual, Synthetic, and Entertainment Audio. Helsinki, 2001.

[16] Free Software Foundation. Gnu general public license (gpl) terms and conditions. http://www.gnu.org/copyleft/gpl.html.

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

[18] Garcia, David; Amatriain, Xavier. 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.

[19] Martínez, Jose M., Overview of the MPEG-7 Standard, document number: ISO/IEC JTC1/SC29/WG11 N4031. http://www.cselt.it/mpeg/standards/mpeg-7/mpeg-7.htm

[20] Hickey, R. Callbacks in c++: Using template functors. C++ Report (1995).

[21] Fast Light Toolkit Homepage: http://www.fltk.org