OpenMusic for Linux and MacOS X

Gerardo M. Sarria M. and Jose Fernando Diago

Free Software and Music Representation Teams
IRCAM
Paris, France

Universidad Javeriana-Cali
Cali, Colombia

This document describes the development of the OpenMusic (OM) port to Linux and MacOS X. We begin explaining the main characteristics of OM, then we show the changes made for each platform (for Linux was chosen the compiler CMUCL and Gtk+ as graphical interface, and for MacOS X the new version 5.0 of MCL was enough for OM to be ported) and we finished with some conclusions and considerations for future work.

Key Words: OpenMusic, Linux, MacOS X, Computer-aided composition

0. INTRODUCTION

A computer application is often conceived to serve as a tool for users that may need it, but it"s also a dependant entity: It depends on the type of computer, the operating system and sometimes on other applications and/or libraries.

OpenMusic (OM) is a visual programming environment for computer-aided composition. It was originally developed under MacOS 9.x using Digitools MCL (a commercial Common Lisp compiler). MCL provides many advantages under MacOS including direct access to the Macintosh GUI, and some libraries inherited from its predecessor (Coral Common Lisp). These advantages were fully exploited during OM"s original development, but they represent a clear dependency on the compiler and the operating system.

The OM source code has been released under the GPL, which allows porting the application under different operating systems and compilers. Linux was a perfect choice for the port, because it provides the user and the developer not only with a free operating system, but also with plenty of free development tools and libraries. The port to MacOS X was planned afterwards, since the original source code and compiler where MacOS 9.x dependant.

The aim of this paper is to present the strategy followed during the port, beginning with a brief explanation about how OM was originally designed and developed, following by explaining some basic details of implementation under Linux and MacOS X, and finally giving some conclusions and considerations for future work and, possibly, new ports.

1. OPENMUSIC

OM was developed on top of Common Lisp, as a graphic interface to this language, thus the user can develop programs (called patches in OM) using any common lisp function and/or library, specially developed libraries and editors for computer-aided composition, constrain solving libraries, etc.

As you can see in [6], the OM main directory contains 5 principal folders:

- Build-image: Contains the files and resources used to make an image of OM.
- Code: In this folder are placed the sources of the OM.
- Image: This folder contains the OM image, the init file and some shared libraries.
- User Library: User Libraries must be placed in this folder.
- WorkSpaces: OM looks for the User"s workspaces in this folder by default.

The OM implementation is divided in two parts: The kernel and the projects. In the code folder, there is two subfolders containing these parts of OM. In the kernel are defined the classes and methods for the language meta-objects (i.e. OMPatch, OMClass, OMBox, etc.), the way to visualisaze these meta-objects, either as simple icons or as container, and the graphic manipulation for creating or modifying the meta-objects. The projects are specialized set of classes and methods directly written in Common-Lisp. There exist three projects in OM: the BasicProject, the MusicProject and the ConstraintProject. The OM structure starting from the CLOS implementation can be viewed in the figure 1 (taked from [6]).

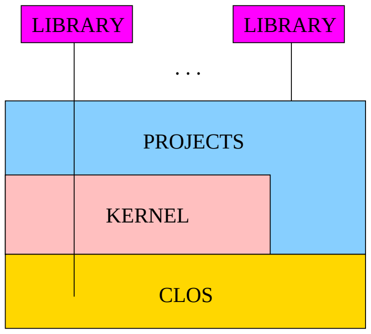

FIGURE 1

The code was developed in a certain way that takes full advantage of the original compiler, which means using specific MCL libraries for writing not only the GUI, but also some not graphic code. To develop an initial strategy for the port, the code had to be explored and understood; OM has a basic class hierarchy, which is described in the figure 2 (adapted from [6] using the Rumbaugh visual representation [13]).

The implementation of these basic classes has no graphical code, however, many of these classes use MCL dependant code such as points, the menu, the events and access to the system and the resources.

The basic class hierarchy for the graphical part is described in the figure 3 (adapted from [6]).

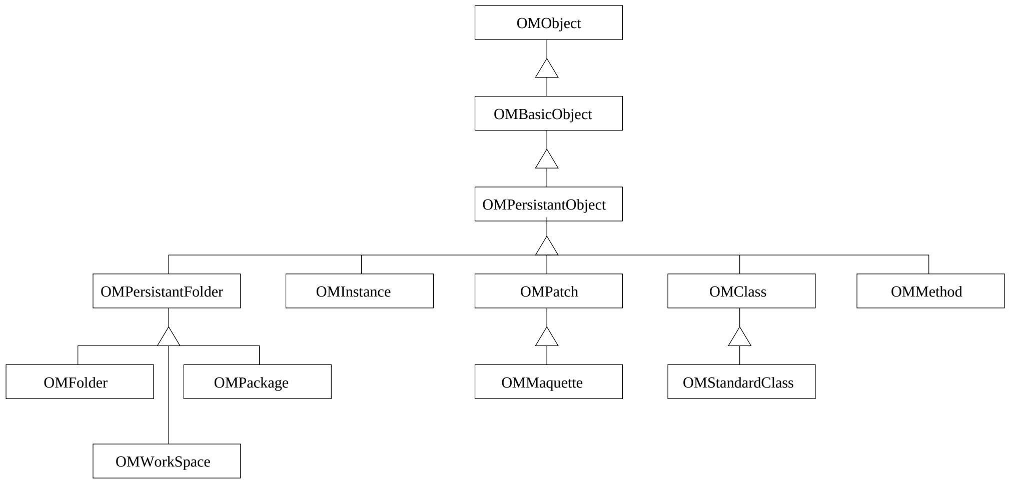

FIGURE 2

The implementation of these classes is tightly integrated with MCL in two ways: First, the implementation itself followed the MCL graphic class model (an MVC-like model. It divides a window into views and subviews, links these views with model objects etc.), and second, the code which describes logical operations between these classes (events, drag & drop, etc.), was implemented using the same class model. After getting to know the code, a final conclusion arrived: The only way to separate the code into its logical part and its system specific part (system access, GUI) was a complete redesign of the whole application. Since the aim was to do the port in a short-time period, instead of redesigning the application it was chosen to adapt the existing code.

2. OM FOR LINUX

The Linux port started with two basic facts given by the original OM code: A free GUI library to implement the graphical part had to be found and the non-portable code of OM had to be adapted to be compiled under a free Common Lisp implementation.

There are many free GUI libraries available under Linux, the one chosen for the project was GTK+, which provides not only a wide set of widgets but also the library itself was available to be used under CMUCL or CLISP (two free Common Lisp implementations). CMUCL was chosen as the compiler for the port, since it fulfilled the requirements imposed by the OM code.

After choosing the development tools, a basic implementation strategy was adopted: Due to the tight integration between the operational and the graphic part of OM, the original code had to be left as unmodified as possible, to avoid a complete re-design of the application. To make this possible, we created a directory for each system (in this case: Linux and MacOS), the code was carefully read and the no-ANSI parts were moved to the MacOS directory in a file named OMPortability that we explain below. The rest of the code (ANSI Common Lisp) was left so that other developers could port it to different platforms.

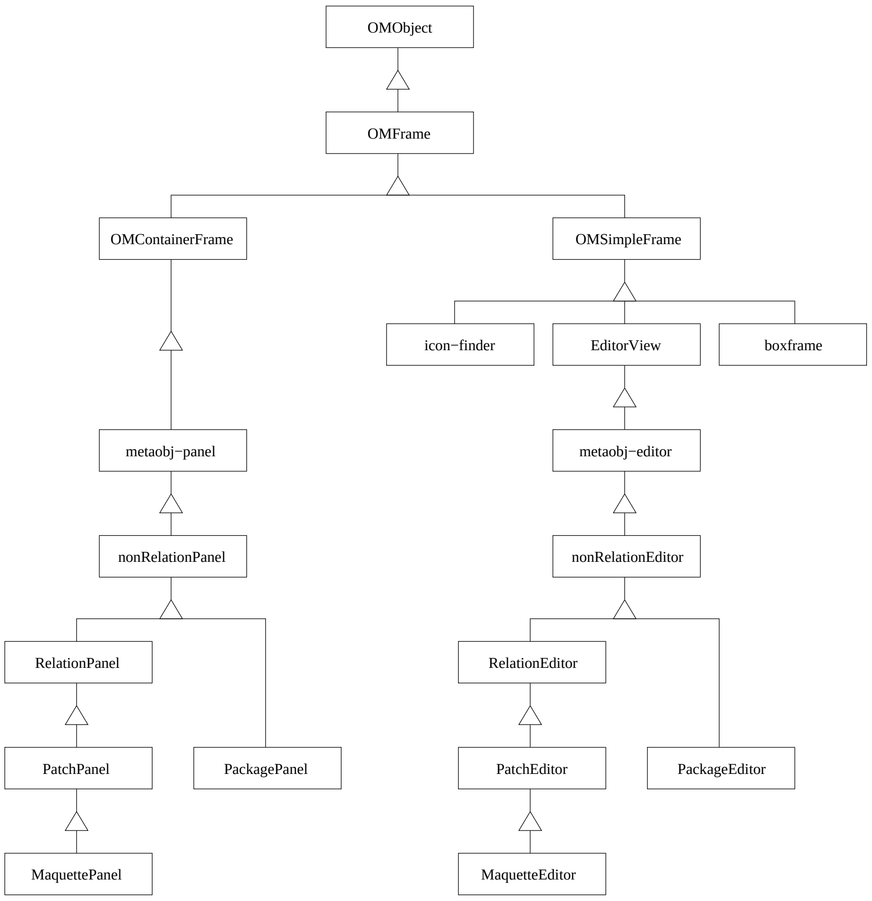

FIGURE 3

The platform dependent code was divided into blocks. The blocks description is as follows (figure 4):

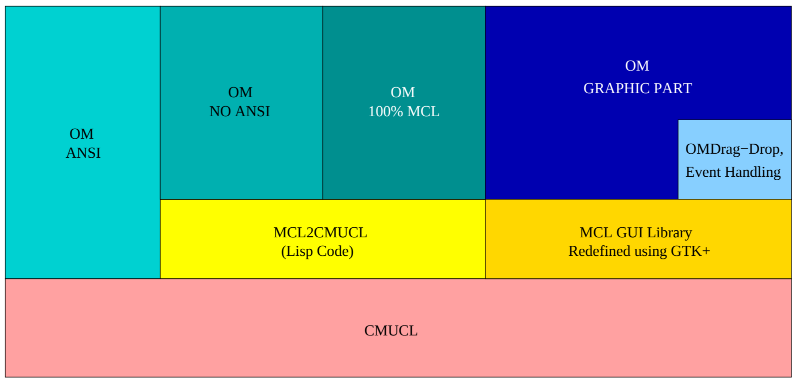

FIGURE 4

- OM ANSI: The code that was written right as [14].
- OM NO ANSI: Some functions were used in the OM"s original implementation (MacOS 9.x) but there are other functions in CMUCL that can be used for the same purpose. For this kind of functions, we make a wrapper for each function in the file Mcl2Cmucl.lisp (to declare the functions with the same name that MCL but in the body of the function to call the function of CMUCL).
- OM 100% MCL: The are other functions no defined in the standard ANSI (there are no corresponding function in CMUCL). These functions had to be reimplemented using the packages availables in CMUCL in the file Mcl2Cmucl.lisp too.
- OM GRAPHIC PART: MCL have there own classes for creating graphical interfaces, and OpenMusic depends on this classes. Gtk+ includes all the necessary components for implement the same MCL classes and maintain the OpenMusic original scheme. Therefore we had to reimplemented the classes using the Gtk+ widgets.

And these blocks was implemented using the following files:

- *Class Constructors*: Provides the initialization methods for the OM graphic classes, it uses directly the GTK+ library and connects the appropriate signals to the widgets so events can be handled. Also there are OM-Linux specific actions implemented inside the class initialization to emulate the behavior obtained trough the use of MCL libraries in OM-Mac.
- *Events*: Contains redefinitions for event-handling methods, so they can work properly using the GTK-based graphic implementation.
- *MCL to CMUCL*: Contains class and function definitions that permit to emulate some MCL functionality inside CMUCL, and there are some functionalities that had to be modified, for example the points now are used with the macro ompoint and not with #@.
- *Compatibility*: Contains OM adapted functions, methods, classes and variables, so they can work properly on OM-Linux.
- *Portability*: Contains methods and functions that must be implemented completely different for each compiler.
- *Unix Tools*: Contains functions that was made only for Linux (and not exists in the Mac version).

The files are placed inside a special folder (this folder is named with the name of the platform) for each OM part (kernel, basic projects, music projects, constraint projects). However, other files should be in this special folder, for exaple, in Linux there was a problem with the metaclasses in the classes OMClass and OMStandardClass. This problem was solved creating other metaclasses and modifying the whole file OMClass.lisp. Consequently, the new OMClass.lisp was in the folder Linux.

Finally, in the BuildImage directory, it was created a folder for each platform, because the way to compile OM under Linux if very different from MacOS: there is a Makefile which is called by the command make and creates CMUCL-precompiled images of the kernel and the projects.

3. OM FOR MACOS X

For MacOS X there are no very significant changes. The OM original compilator (MCL) was port to this platform. This means that it can be compiled and run the same way that OpenMusic for MacOS 9.

However, this architecture has its own special directories in the kernel and the projects, and the same modules except for

- *MCL to CMUCL*: There is no reason to remake the functions,
- *Class Constructors*: The methods for initialize the classes are in the source of MCL,
- *Compatibility*: The original source code was made for MCL, so this code is 100% compatible, and
- *Unix Tools*: There is a *Mac Tools* file with the same meaning.

4. CONCLUSIONS AND FUTURE WORK

The aim of the project (an implementation of the software OpenMusic in the operating systems Linux and MacOS X) was completly achieved. The OM port has been developed as an open source project and is distributed under the GPL license. The sources are always available trough CVS, the main site for the project is http://sourceforge.net/projects/ircam-openmusic. The site is mainly a developers site, theres documentation available for compiling and installing OM Linux.

The project itself needs developers and testers, so bug submissions and/or corrections are a must. It's important to take into account that right now the OM code has been almost unified for both platforms, but there are still low-level issues that are particular and crucial for each platform, this is valid for all future changes and enhancements.

REFERENCES

1. Carlos Agon Amado. *An Environment for Computer Assisted & Composition.* PhD thesis, Paris VI, 1998.
2. Inc. Apple Computer. *Inside Macintosh: Macintosh ToolBox Essentials.* Apple Computer, Inc., 1992.
3. Inc. Apple Computer. *Inside Macintosh: More Macintosh ToolBox.* Apple Computer, Inc., 1993.
4. Inc. Digitool. *Macintosh Common Lisp Reference.* Digitool, Inc., 1996.
5. Tony Gale and Ian Main. *Gtk 1.2 Tutorial.* http://www.gtk.org/tutorial, March 2001.
6. Ircam Users Group. *OpenMusic Developer's Documentation*, 1998.
7. Ircam Users Group. *OpenMusic User's Manual and Tutorial*, 1999.
8. Sonya E. Keene. *Object-Oriented Programming in Common Lisp.* Symbolics, Inc., 1989.
9. Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*, chapter 5 y 6. MIT Press, 1991.
10. Robert A. MacLachlan. *CMU Common Lisp User's Manual.* http://cvs2.cons.org/ftp-area/cmucl/doc/cmu-user/, 2003.
11. Roger S. Pressman. *Software Engineering.* McGraw-Hill, 1988.
12. Camilo Rueda, Carlos Agon, Gerard Assayag, and Mickael Laurson. *Computer assisted composition at Ircam: Patch Work and OpenMusic.* 1999.
13. James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Loransen. *Object-Oriented Modeling and Design.* Prentice Hall, Inc., 1991.
14. Guy L. Steele. *Common Lisp the Language, 2nd Edition.* Digital Press, 1990.
15. Gtk+ Team. *Gtk+ API Reference.* http://www.gtk.org/api, 2001.