OpenMusic 5: A Cross-Platform Release of the Computer-Assisted Composition Environment.

Jean Bresson, Carlos Agon, Gerard Assayag

Musical Representations Team

IRCAM – Centre G. Pompidou – 1, place I. Stravinsky – 75004 Paris – France

{bresson,agon,assayag}@ircam.fr

Abstract. *This paper presents the computer-assisted composition environment OpenMusic and introduces OM 5.0, a new cross-platform release. The characteristics of this system will be exposed, with examples of applications in music composition and analysis.*

Resumo. *Este artigo apresenta o ambiente de composição assistida por computador OpenMusic, e introduz OM 5.0, uma nova versão do programa, compatível Mac e PC. Serão expostas também algumas características do sistema tendo em vista aplicações para a composição e análise musical.*

1. Introduction

The Ircam's Computer-Aided Composition (CAC) project aims at connecting formal computing tools and musical thought, in order to provide the composers with programming languages adapted to their specific needs, and to allow them to formalize, develop, and experiment their musical ideas [Assayag 1998]. Our experience demonstrates that the definition of computing models corresponding to musical situations can lead to a rich creative and analytic approach of composing processes.

In this framework, several environments have been developed, around the core concept of programming languages, facilitating the formulation of musical concepts: Formes (an object oriented environment in Lisp for high level control of sound synthesis, [Rodet and Cointe 1984]), PreForm/Esquisse (an extension of Formes with graphical interfaces, providing multidimensional objects and functions to manipulate them) , Crime (a CAC environment providing musical abstractions and symbolic score descriptions, [Assayag et al. 1985]), PatchWork and OpenMusic [Agon et al. 1999]. These languages have been enriched and extended with graphical interfaces, along with the evolution of computer science, allowing the introduction of visual programming.

OpenMusic is an example of a complete programming language dedicated to music composition. In this article, we present an overview of this programming/composition environment (section 2) and some examples of applications (section 3). In section 4 we introduce OM 5, the new cross-platform release of OpenMusic.

2. The OpenMusic visual programming language

2.1. A visual language based on Lisp

PatchWork [Laurson and Duthen 1989] is a graphical interface of the Common Lisp programming language, in which programs are developed graphically by means of functional boxes and connections. In this environment, any Lisp function can be represented as a graphical box, and connected to other boxes to constitute a program. This graphical program is a connected acyclic graph which corresponds to a Lisp form. The data structures resulting from the functional boxes can be retained and associated with graphical editors, which allows their visualization and edition in a musical notation.

OpenMusic [Agon 1998] was designed and developed in CLOS (Common Lisp Object System [Steele 1998]) on top of the PatchWork model. It is now a powerful visual language, allowing various programming models and applications.

2.2. Functional programming

The basis of OpenMusic, inherited from PatchWork, is purely functional. Functional programming is an intuitive approach that has been widely used by the composers. It allows to define operations that are applied on multidimensional data structures, in order to create new ones.

In OpenMusic, a *patch* is a visual algorithm, in which boxes represent functional calls, and connections are functional compositions. The evaluation of a box causes a sequence of successive evaluations, corresponding to the execution of the program.

The visual language provides a set of graphical control structures, such as loops and conditional controls, as well as the possibility to manipulate programming concepts like abstraction or recursion. Through functional abstraction, some elements of the program can become variables, which leads to the definition of functional objects. These objects can then be embedded into other programs or abstractions (see Figure 1).

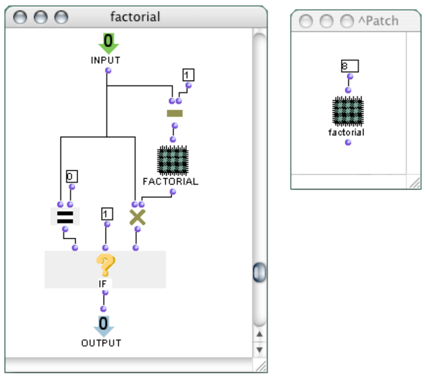


Figure 1. Functional programming in OpenMusic: the example of the "factorial" algorithm illustrates some of the language features : conditional controls, abstraction, recursion. To create an abstraction, the user adds inputs and outputs to the patch, which will then appear on the patch abstraction box. This abstraction can then be applied in another patch or in the patch itself.

2.3. Object-Oriented programming

Built on the Common Lisp Object System, the visual language is an object-oriented environment enriched with classes and generic functions [Agon 2003].

Some basic classes are provided by the environment to represent musical structures (notes, chords, sounds, break-point functions, etc.) These classes can be used in the development of the graphical programs through the use of *factory* boxes. A factory is a special box which inlets correspond to the public slots of the corresponding class. Its evaluation creates and stores an instance of this class (see Figure 2).

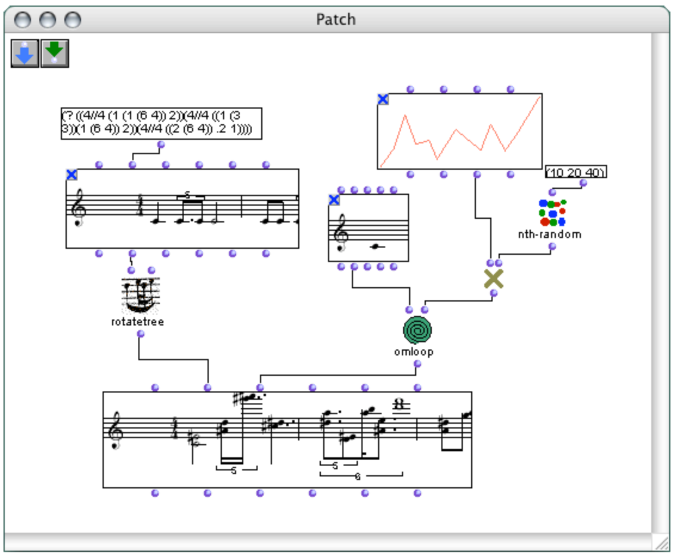
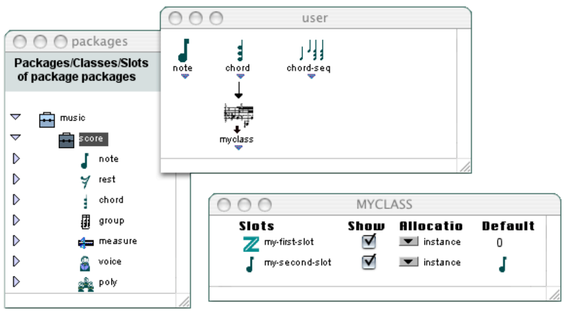


Figure 2. A patch using some of the OM classes. In this example, a *voice* instance is created by computing a rhythmic pattern and a melodic profile.

The user can also create his/her own classes, using slot definition tools, and by setting inheritance relationships. The OpenMusic classes can thus be extended with user defined subclasses (Figure 3-a). The polymorphism feature of CLOS is integrated in the language through the possibility to create generic functions and methods (Figure 3-b).

(a)



(b)

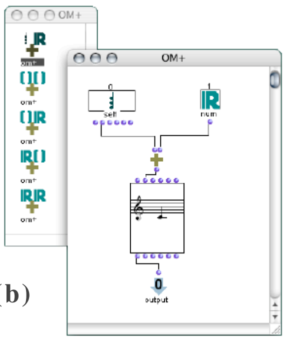


Figure 3. Object-Oriented programming - (a) Inheritance: definition of a class extending an OM class - (b) Polymorphism: definition of a new method for the *om+* generic function.

The object-oriented language is extended with a graphical MOP (Meta-Object Programming) [Agon and Assayag 2003], allowing one to interact with the language elements and providing access to the visual components' properties and behavior.

2.4. Constraint programming

Constraint programming aroused a great interest among composers and computer music researchers. In OpenMusic, composers can graphically define a constraint satisfaction problem (CSP), and try to solve it using different constraint resolution systems: Situation [Rueda and Bonnet 1998], Screamer [Siskind and McAllester 1993], or OMClouds [Truchet et al. 2003]).

2.5. Music representation and notation

The representation of musical structures is a concrete expression of the information transmitted across the system. In OpenMusic this representation can be audibly or visually rendered with audio and MIDI players (the MIDI manipulations and renderings are implemented using the MidiShare system [Orlarey and Lequay 1989]) or with graphical editors associated to the main OM classes (musical structures, break-point functions, sounds, etc.) Musical notation editors provide the user with interactive editing and navigation into the hierarchical structure of the score (see Figure 4).

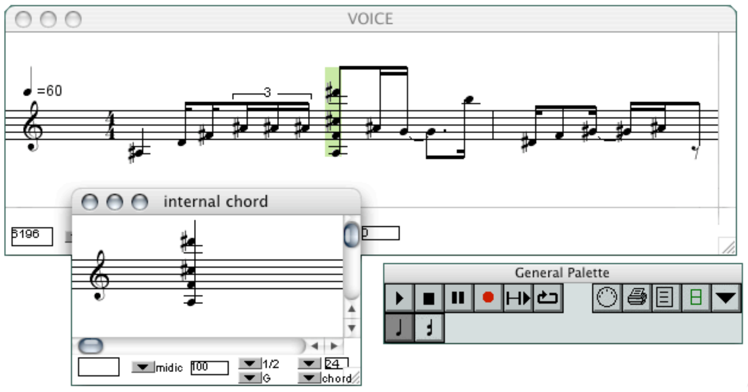


Figure 4. OpenMusic score editor.

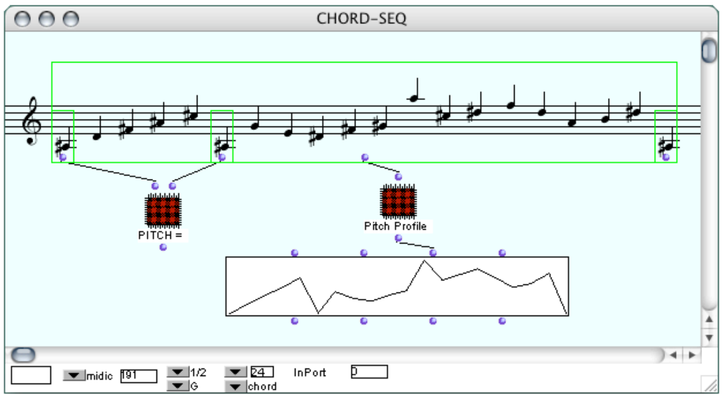


Figure 5. Programming inside the score editor.

The integration of musical editors with visual programs can be done in different ways:

- By including the editors in patches; as a way to control the value of the components of a computation tree (as in Figure 2).

- By including visual programs within the score editor itself; by defining relationships between different sub-structures of a score (Figure 5).

2.6. Temporal aspects: the *Maquette*

The *maquette* [Agon and Assayag 2002] is an original concept designed to unify both the program and the score in a common representation. A *maquette* is a bi-dimensional space containing blocks called "temporal boxes" (see Figure 6). Such boxes can contain either :

- simple musical objects (instances of the OM musical classes),
- temporal patches (patches having an output in the temporal context),
- embedded *maquettes* (for constructing hierarchical temporal structures).

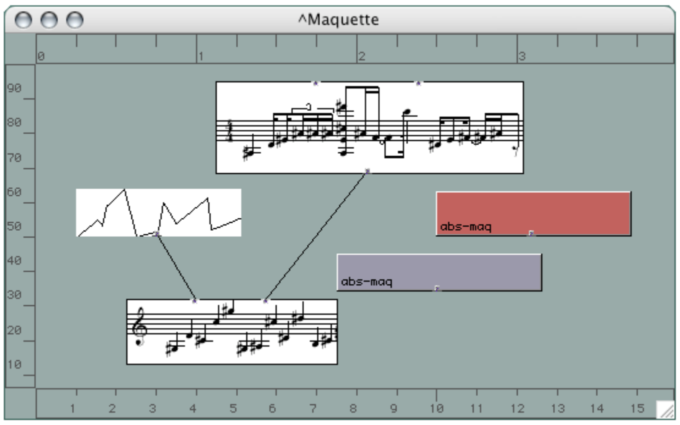


Figure 6. A *maquette.*

In the *maquette*, the horizontal axis represents time, so that the position and horizontal extension of the temporal boxes can be associated with offsets and duration in the *maquette*'s time referential. The MOP allows one to access the graphical characteristics of a temporal box and to put it in relation to its contents. For example, the vertical position and extent, can be assigned to any musical (frequency, intensity) or functional (an input for the programs execution) parameter.

Boxes can have inputs and outputs and be connected in the maquette. It allows functional relationships between the temporal objects, revealing further types of musical semantics. A maquette can then be evaluated as a graphical program which takes into account the temporal properties of its elements.

Eventually, the computed maquette can be executed (i.e. played) by MIDI and audio players.

3. OpenMusic Applications

3.1. Models for musical formalisms: examples of compositional applications

OpenMusic has been used in a large panel of applications and musical pieces. Many projects and toolboxes, dedicated to special musical purposes and specific formal systems, have been developed. Some of these projects are available in the software release as "user libraries".

The *Esquisse* library, for example, was created by Tristan Murail for his works on harmonic structures. *OMChaos* and *OMAlea* were developed by Mikhail Malt to build harmonic materials with stochastic processes and non-linear models [Malt 1994]. They were used for the composition of *Lambda 3.99* and *Actrinou*. The *Profile* and *Morphologie* libraries were used by Jacopo Baboni-Schilingi for the creation and manipulation of melodic profiles.

Rhythmic issues (quantification, rhythmic manipulation, rhythmic canons construction) also inspired the work of many composers. For instance, Karim Haddad manipulates the OpenMusic rhythmic trees representation [Agon et al. 2002] to create complex rhythmic structures with the *OMTrees* library (e.g. in *...und wozu Dichter in durftiger Zeit?...*).

Various musical applications of constraint programming have also been developed in OpenMusic, for example by Örjan Sandred with the *OMRC* library which is specialized for finding structures corresponding to rhythmic constraints (used for example in the piece *Kalejdoskop*.) The OM constraint systems were also used with harmonic constraints (e.g. by Antoine Bonnet in *Epitaphe*, with the *Situation* constraint solver).

3.2. Control of sound synthesis and writing electronic music

The development of Digital Signal Processing technologies brought to music creation a new field of exploration, and extended compositional activities to the composition of the sound itself [Risset 2002]. The connection between musical signals, and symbolic objects and concepts became an interesting challenge for both musicians and researchers. This problem is one of our current research axes in OpenMusic. We consider it from a variety of angles.

The importation of sound description data in the compositional environment allows the composer to use musical material coming from real sounds, by transforming it into symbolic entities. Sound-related material can be imported either as *sound* boxes (from which data can be extracted as simple sound samples or analysed to produce sound analysis data), or as *SDIF* boxes. The SDIF interface [Bresson and Agon 2004] allows one to import SDIF sound description data [Wright et al. 1998] coming from an analysis processed by external tools and softwares, and to convert them into symbolic musical structures (see Figure 7-a).

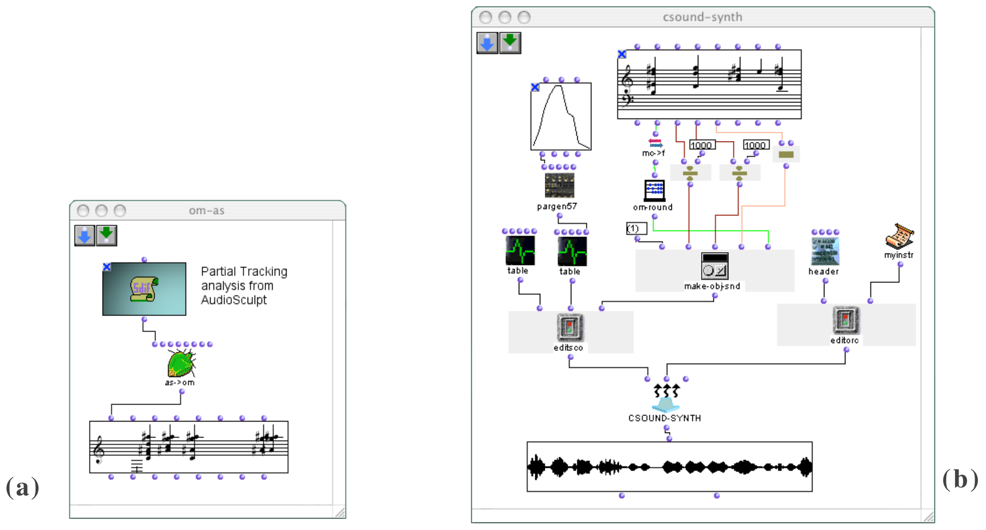


Figure 7. Sound analysis/synthesis in OM - (a) An importation of partial tracking analysis data and conversion into a polyphonic musical sequence [Hannape 1995] - (b) Sound synthesis with CSound using symbolic data and operations.

The control of sound synthesis processes in OM allows one to take advantage of the computational and expressive power of its visual language for creating and manipulating sound synthesis parameter data. OM then generally delegates the sound synthesis processing to external synthesizers, either by formatting parameters files, or through direct interfaces. Some of these interfaces are associated with dedicated libraries: OM2CSound for the graphical design of CSound instruments and scores [Boulanger 2000] (see Figure 7-b), OM-AS for creating parameter files for the SuperVP phase vocoder [Depalle and Poirot 1991], OMModalys for creating Modalys physical synthesis patches [Eckel et al. 1995]. These tools have been used in the creation of several electro-acoustic pieces of Karim Haddad, Hans Tutschku, Mauro Lanza, and more.

OMChroma is an other original approach to high level control of sound synthesis based on Marco Stroppa's *Chroma* system [Agon et al. 2000]. Using object-oriented programming techniques, this system allows a powerful instantiation of classes representing complex data sets which can be sent to several external synthesizers. OMChroma is used in several compositions by Marco Stroppa (e.g. *Come Natura di Foglia*, etc.)

Sound spatialization can also be involved in computer-assisted composition. OpenMusic provides special classes for designing three dimensional curves and trajectories. This spatialization data can be sent to the Ircam's *Spatialisateur* [Jot and Warusfel 1995] through the *OMSpat* library (e.g. Brian Ferneyhough, in *Stelae for the failed times*).

3.3. Musical analysis

OpenMusic is used in musicology as a support for experiments on formal and mathematical models of music. The reconstitution of musical formalisms and musical pieces in this environment provides an intuitive and interactive approach of music analysis. Interesting models of works by Iannis Xenakis, for example, (*Achorripsis*, *Herma*, *Nomos Alpha*), could be recreated [Agon et al. 2004] (see Figure 8).

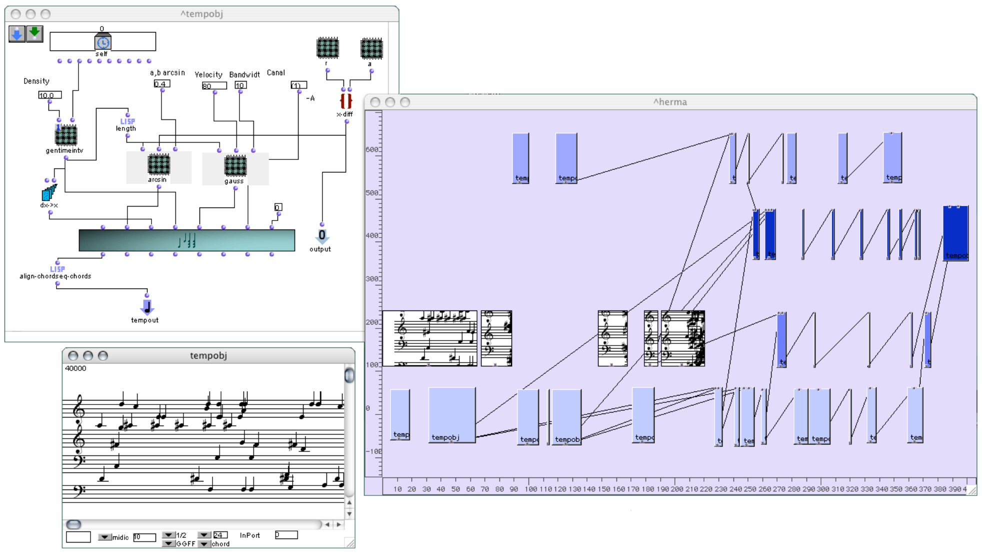


Figure 8. *Herma* (I. Xenakis). The composition's structure is represented in a *maquette*: the pitch set operations and probabilistic rules are implemented by means of graphical programs and connections between blocks. Each box can be considered as a graphical program (top-left window) in the computational flow, or as a musical object in time (bottom-left window).

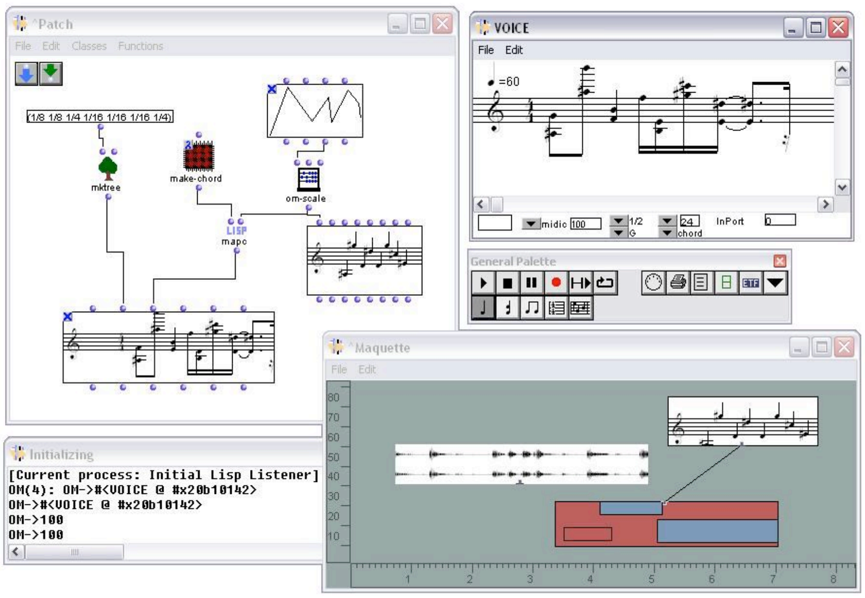
OpenMusic is also an experimental environment for various typres of research on harmonic analysis, pattern recognition, neural networks, and musical style analysis and simulations, in several institutions and universities.

4. OpenMusic 5

4.1. A cross-platform environment

The OpenMusic 5.0 release contains major internal changes. In this new version, the code has been reorganized and divided in two distinct parts: the *API* code, and the *OM* code.

The *API* (Application Programming Interface) code contains low level primitives and structures which have been identified as being dependent on the System and/or on the Lisp implementation. These related primitives are generally non Common Lisp code (previously specific to the Macintosh platform). They can be grouped into several categories: windows, frames and graphic objects; dialog items and menus; graphical primitive structures; drawing tools; images and icons; user interactions (keyboard, mouse), drag&drop, and events handling; meta-objects programming tools; system tools and communication with external libraries. OpenMusic was historically developed on Digitool's Macintosh Common Lisp (MCL), so this new API has been specified following the MCL model in order to maintain the original OpenMusic code structure and functional system.



```
[Current process: Initial Lisp Listener]
OM(4): OM->#<VOICE @ #x20b10142>
OM->#<VOICE @ #x20b10142>
OM->100
OM->100
```

Figure 9. OpenMusic on Windows XP.

The *OM* code contains the functional kernel of OpenMusic (the visual programming language), and some "projects" (code modules dedicated to specific applications : music, sound synthesis, musical analysis, constraint programming, etc.) This part represents the core of OpenMusic, and has been rewritten following the API specification.

The API now allows an abstraction of the system-related constraints while writing OpenMusic code. This code is thus interpreted for a target platform or Lisp implementation, providing the API is implemented in it. This is the case for Macintosh (with MCL) and Windows (with Allegro Common Lisp) (see Figure 9). A Linux version would need the adaptation of the API on this platform; preliminary trials have been done using SBCL and GTK.

Patches from older versions of OM can still be loaded and are automatically converted, when saved, following the new specifications. Patches can also be transferred between the Mac and PC versions.

4.2. Examples of new features in OM 5.0

The OM 5 audio library, based on GRAME's *LibAudioStream* project, extends the audio support in OpenMusic. Advanced audio stream processing operations (mixing, sequencing, effects, etc.) are available. The sound objects can now be assigned to an audio channel, so that audio tracks can be simulated and controlled using a graphical mixing console (see Figure 10).

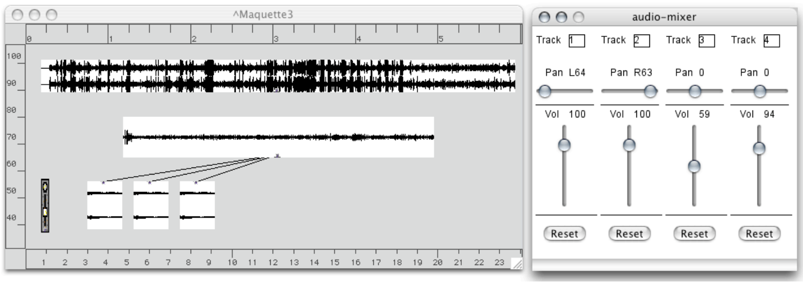


Figure 10. Interactive control of audio objects.

The different available tools for sound analysis and synthesis have been gathered around direct external interfaces (CSound, SuperVP, OSC, etc.) and a set of shared globals variables and preferences. The SDIF toolbox is also enriched with new classes and functions allowing one to store and write structured SDIF data within patches (see Figure 11.)

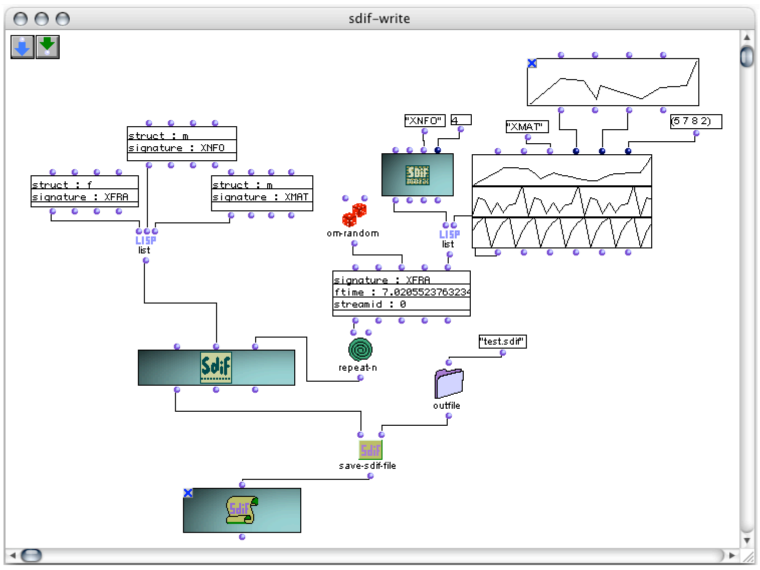


Figure 11. Programming with SDIF data structures.

Research in mathematical music theory carried out with OpenMusic (set theory, classifications, canons, etc.) has been implemented in a new set of mathematical tools .

The OM musical classes have also been extended with a tonality model. By following the hierarchical architecture of the musical objects, this tonal model contains knowledge about relative tonality, which can be used for musical notation in score editors (see Figure 12), or to compute properties and operations in the tonal field (tonal transpositions, etc.)

Microtonal notation is now available in the score editors. Micro-intervals up to 16ths of a tone can be manipulated and displayed (Figure 12).

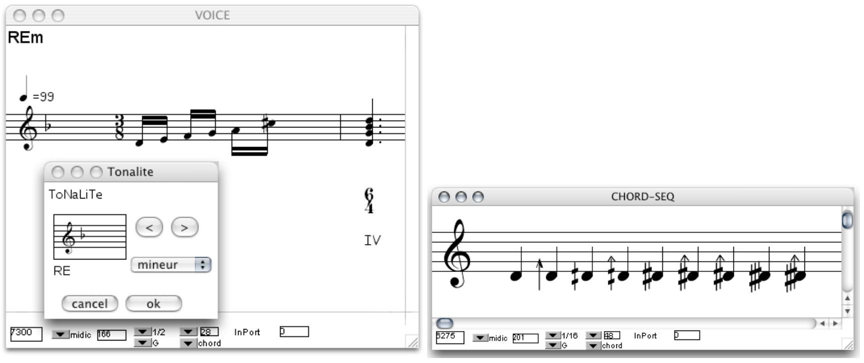


Figure 12. Representation of tonality and microtonality in score editors.

The *maquette*'s flexibility was also improved for example with the possibility to set functional relationships throught the hierarchical levels. Current developments are oriented towards a better integration of the programs in the *maquette*.

5. Conclusion

OpenMusic has proven to be a practical and innovative tool for music composition, allowing musical material and musical processes to coexist in the same representation.

Many composers have been using OpenMusic, each with his/her specific tools and methodologies. The *OM Composer's Book* [Agon et al. 2005], to be published in 2005-2006, is a collection of user experiences in which each chapter, written by a composer, describes his/her usage of OpenMusic for composition.

OpenMusic is also used in musical research, and is taught in composition classes in several universities and conservatories (Columbia University, Harvard, Stanford, Musikhochschule Stuttgart, Berkeley, CNSM Paris, etc.)

OM 5.0 constitutes a new step in the development of OpenMusic, and may promote its diffusion with the extension to new platforms.

References

Agon, C. (1998) "OpenMusic: Un Langage Visuel pour la Composition Assistée par Ordinateur", PhD. Thesis, Université Paris VI.

Agon, C., Assayag, G., Laurson, M. and Rueda, C. (1999) "Computer Assisted Composition at Ircam: PatchWork & OpenMusic", Computer Music Journal 23(5).

Agon, C., Stroppa, M. and Assayag, G. (2000) "High Level Musical Control of Sound Synthesis in OpenMusic", Proceedings of the International Computer Music Conference, Berlin, Germany, 2000.

Agon, C. and Assayag, G. (2002) "Programmation Visuelle et Editeurs Musicaux pour la Composition Assistée par Ordinateur", IHM'02, Poitiers, France, ACM Computer Press.

Agon, C., Haddad, K. and Assayag, G. (2002) "Representation and Rendering of Rhythmic Structures", WedelMusic Darmstadt, IEEE Computer Press.

Agon, C. (2003) "Object-Oriented Programming in OpenMusic", in Topos of Music, Mazzola G., Birkhäuser Verlag Ed.

Agon, C. and Assayag, G. (2003) "OM: A Graphical Extension of CLOS using the MOP", Proceedings ICL '03, New York.

Agon, C., Andreatta, M., Assayag, G. and Schaub, S. (2004) "Formal Aspects of Iannis Xenakis' "Symbolic Music": A Computer-Aided Exploration of Compositional Processes", Journal of New Music Research, 33(2).

Agon, C., Assayag, G. and Bresson, J. (ed.) (2005) "The OM Book", Delatour Editions, to be published.

Assayag, G., Castellengo, M. and Malherbe, C. (1985) "Functional Integration of Complex Instrumental Sounds in Music Writing", Proceedings of the International Computer Music Conference, Burnaby, Canada, 1985.

Assayag, G. (1998) "Computer Assisted Composition Today", 1st Symposium on Music and Computers, Corfu, 1998.

Boulanger, R. (ed.) (2000) "The Csound Book", MIT Press.

Bresson, J. and Agon, C. (2004) "SDIF Sound Description Data Representation and Manipulation in Computer Assisted Composition", Proceedings of the International Computer Music Conference, Miami, USA, 2004.

Depalle, Ph. And Poirot, G. (1991) "A Modular System for Analysis, Processing and Synthesis of Sound Signals", Proceedings of the International Computer Music Conference, Montreal, Canada, 1991.

Eckel, G., Iovino, F. and Caussé, R. (1995) "Sound Synthesis by Physical Modelling with Modalys", Proceedings of the International Symposium on Music Acoustics, Dourdan, France.

Hannape, P. (1995) "Integration des representations temps/frequence et des representations musicales symboliques", in "Recherches et applications en informatique musicale", M. Chemillier and F. Pachet (ed.), 1995.

Jot, J.-M. and Warusfel, O. (1995) "A Real-Time Spatial Sound Processor for Music and Virtual Reality Applications", Proceedings of the International Computer Music Conference, Banff, Canada, 1995.

Laurson, M. and Duthen, J. (1989) "Patchwork, a Graphic Language in PreForm", Proceedings of the International Computer Music Conference, Ohio State University, USA, 1989.

Malt, M. (1994) "Modêlos Matemáticos e composição Assistida por Computador, Sistemas Estocásticos e Sistemas Caóticos", in Primeiro Simposio de Computação e Música, Caxambu, MG, Brazil

Orlarey, Y. and Lequay, H. (1989) "MidiShare", Proceedings of the International Computer Music Conference, Columbus, USA, 1989.

Risset, J.C. (2002) "Computing Musical Sounds", in Assayag et al. (ed.) "Mathematics and Music", Springer, 2002.

Rodet, X. and Cointe, P. (1984) "Formes: Composition and Scheduling of Processes", Computer Music Journal Vol. 8(3).

Siskind, J. M. and McAllester, D. (1993) "Nondeterministic Lisp as a Substrait for Constraint Logic Programming", AAAI-93, pp133-138.

Steele, G. L. (1998) "Common LISP The language, second edition", Digital Press, USA.

Truchet, C., Assayag, G. and Codognet, Ph (2003) "OMClouds, petits nuages de contraintes dans OpenMusic", Actes des Journées d'Informatique Musicale 2003, Montbeliard, France.

Wright, M., Chaudhary, A., Freed, A., Wessel, D., Rodet, X., Virolle, D., Woehrmann, R. and Serra, X. (1998) "New applications of the Sound Description Interchange Format", Proceedings of the International Computer Music Conference, Ann Arbor, USA, 1998.