Next-generation Computer-aided Composition Environment: A New Implementation of OpenMusic

Jean Bresson, Dimitri Bouche, Thibaut Carpentier, Diemo Schwarz, Jérémie Garcia

Next-generation Computer-aided Composition Environment: A New Implementation of OpenMusic

Jean Bresson, Dimitri Bouche, Thibaut Carpentier, Diemo Schwarz
UMR STMS: IRCAM/CNRS/UPMC Sorbonne Universités, Paris, France
{bresson,bouche,carpenti,schwarz}@ircam.fr

Jérémie Garcia
Université de Toulouse – ENAC, France
jeremie.garcia@enac.fr

ABSTRACT

o7 is a new implementation of the OpenMusic computer-aided composition environment. This paper gives a general presentation of its visual programming framework, musical structures, and in-built graphical editors. It also describes embedded features for the interactive and dynamic execution of musical processes, and for digital signal processing.

1. INTRODUCTION

Computer-aided composition software provide composers with computer tools and formalisms for the generation or transformation of musical material [1]. After several generations of software, current computer-aided composition frameworks like OpenMusic [2] have taken the form of versatile domain-specific programming languages combining expressive and computational power with the possibility to interactively visualise and edit musical information through symbolic representations.

Released in the late 90s and successor of the Patchwork visual programming environment [3], OpenMusic (OM) is today a widespread and common element of the contemporary music composers' toolbox. OM can be considered a graphical front-end of the Common Lisp programming language [4], extended with musical functionalities and data structures. [1] Initially dedicated to the processing of symbolic musical material (musical scores), it is now operating in a broad range of areas such as sound processing and synthesis, spatial audio, mathematical music theory, and others. A fairly developed and active user community lives within user groups and institutional contexts worldwide. [2]

This relative success and maturity shall not prevent from looking forward: in this paper, we present a new implementation of the environment, introducing significant evolutions of the computer-aided composition framework and concepts. o7 is the main deliverable of a research project focused on the idea of *interactivity* in compositional processes and the development of innovative tools for the timed control of music and signal processing [7]. The resulting software constitutes an original blend of traditional and more prospective computer-aided composition features. [3]

This paper is organised as follows: In Section 2 we first go through general characteristics of the visual programming environment. Section 3 then focuses on new embedded data structures for computer-aided composition. In Section 4 we describe advanced interactive features related to the execution and rendering mechanisms of the software. Finally, Section 5 is dedicated to digital signal processing.

2. ENVIRONMENT AND VISUAL PROGRAMMING FRAMEWORK

An ambition at the beginning of this project was to simplify and modernize the OM visual programming environment, in order to facilitate its access for new users and to improve the general user experience of more experimented ones.

Formally, visual programs (also called *patches*) mostly follow the existing OpenMusic specification [4] — for this reason we call this work a new *implementation* of the visual language. They are shaped as standard directed acyclic graphs made of functional boxes and object constructors connected by "patch cords" [5] (see Figure 1).

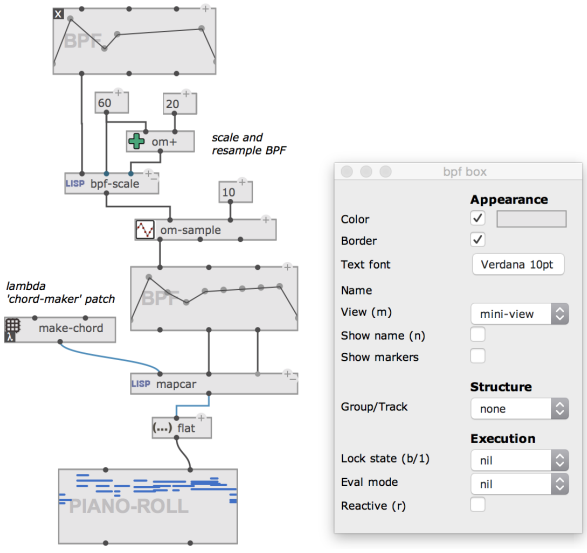


Figure 1. A simple patch processing an input break-point function (BPF), and generating a sequence of MIDI chords. The inspector at the right displays and provides access to the *properties* of a selected component in the visual program.

[1] Readers unfamiliar with the OM should also consult previous publications and works such as [5] or the OM online documentation at http://repmus.ircam.fr/openmusic/.

[2] The *OM Composer's Book* series provides a rich overview of the practice and use of the environment over the past decades through the experience of an international sample of composers [6].

[3] We actually consider this software as a research prototype more than a full-featured implementation of OpenMusic. We provisionally call it o7 (with no "M") so that its release and distribution do not interfere with the use of current operating versions of OpenMusic (OM6).
Try it out! → https://github.com/j-bresson/o7

An objective for the new interface was to provide centralized and homogenized access to the information about the visual program components. All the objects of the visual language share a basic system of mutable "properties". Thanks to a dynamic interface-building mechanism based on the declared *type*, *defaults*, *documentation* etc., of such properties, selected attributes of any class can be visualized and modified in an inspector window (as in Figure 1), in dedicated editors (see for instance in Figure 2), or presented as general system preferences (in this case the *default* value for the attribute is displayed and edited).

Editor windows. Editors are usually attached to musical structures manipulated in OM visual programs. They are constructed using flexible and embeddable graphical layouts, in such a way that compound editors can be easily assembled for objects containing other kinds of object's editors as sub-components. Some examples will be given and displayed in the following sections.

File system and persistence. The existing file system and persistence mechanisms in OM are based on the concept of *workspace*. These were mostly discarded and replaced by standalone documents, so that users can easily step in the environment just by opening and reading/executing documents and programs.

Documents are therefore handled by a centralized system keeping track of used files and resources, cross-dependencies (programs used in other programs), as well as package or external library dependencies. We use a markup language-style format to store the documents, written as structured embedded lists in text files, which makes the resources and visual programs both machine- and human-readable.

3. MUSICAL STRUCTURES

An initial set of data structures has been designed for the environment, by composition of abstract super-classes determining fundamental features and behaviours of the musical objects.

Timed sequences. One of the main abstract super-classes, shared by most musical objects, is called TIMED-SEQUENCE [8]. Every object inheriting from this class is reducible to an ordered set of time-stamped events, which can be:

- Visualized and manipulated as such on a timeline-based representation.
- Transformed into a set of timed actions (predefined or programmed by the user) during the execution (rendering/playback) of the musical object.

The first musical objects created in this framework are the standard BPF (break-point function), automations, 2D/3D curves, etc. Figure 2 shows a simple patch instantiating a BPC object (2D curve), attached to a custom action to be performed for each point at executing this curve.

The DATA-STREAM is another kind of TIMED-SEQUENCE, gathering shared features for container objects simply consisting of a sequence of timed data, and offering flexible representation and rendering functionality. We call DATA-FRAME the type of elements in this sequence: a super-class for atomic musical objects such as MIDI-NOTE, OSC-BUNDLE, SDIF-FRAME or other musical data specified at some point of a time-line. The graphical representation

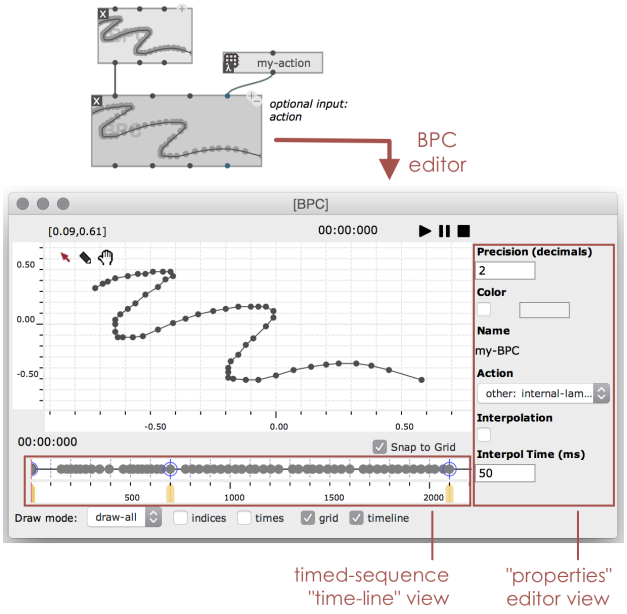
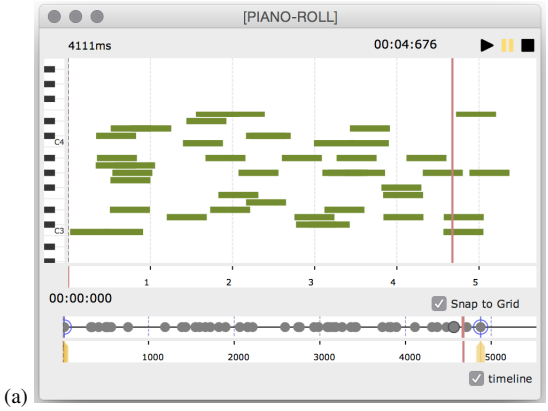
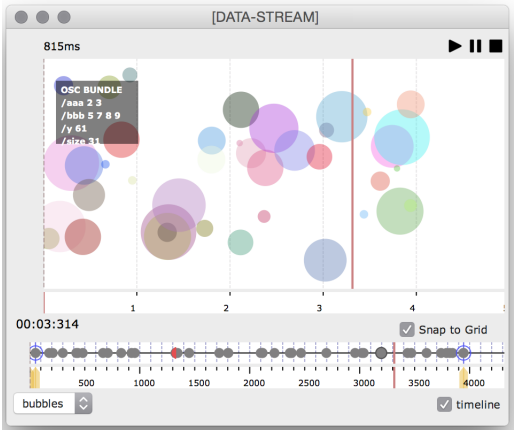


Figure 2. Instantiation and visualization/editing of a BPC (break-point curve). An action (*my-action*, a lambda-function defined by a visual program) is set to be applied to each point at reading the curve. The BPC editor includes the auto-generated *properties* pane at the right, and the time-line representation inherited from TIMED-SEQUENCE at the bottom.



(a)



(b)

Figure 3. Two kinds of DATA-STREAMS: (a) MIDI piano-roll and (b) custom representation of a stream of OSC-bundles. In (b) the radius of the circles depicts the amount of data contained in the bundles, and their color and vertical position follow other arbitrary rules. The contents of a data chunk can be inspected by hovering the mouse over it in the editor.

and behaviour of the DATA-STREAM contents in the editor can be easily specialised, facilitating the implementation of notes in a PIANO-ROLL, for instance, or of more customized/user-programmed graphics (see Figure 3).

Compound objects and containers. The COLLECTION object is proposed as a basic container for arbitrary sets of objects of any given type (generalizing class-specific containers such as BPF-LIB, BPC-LIB, etc. featured in OM).

The compound-layout editor model described above facilitates building editors for such composite structures. Figure 4 shows the COLLECTION editor: a simple browser, whose main graphical component is built and updated automatically according to the type of its contents — in this case, a set of 3DC objects (3D curves).

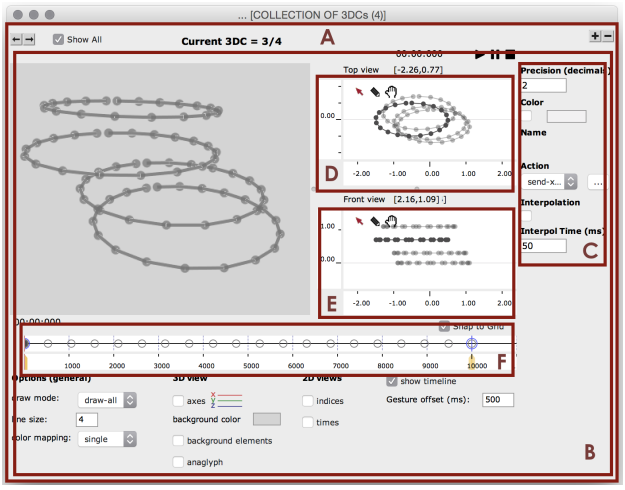


Figure 4. Editor of a COLLECTION containing 3DC objects. The main-window layout embeds a small COLLECTION-specific header (A), and the current contents' editor at the bottom (B). Here the embedded 3DC editor itself embeds a number of sub-editors as graphical components: the standard *properties* pane (C), BPC editors for editing top and front views (D, E), and the time-line editor inherited from TIMED-SEQUENCE (F).

Meta-scores. A major improvement of this implementation of OpenMusic concerns the *maquette* interface: a hybrid visual program / sequencer allowing objects and programs to be arranged in time and connected by functional relations — including dependencies to the temporal layout and context, see [2, 9].

In O7 we call this tool a *meta-score* [10] (see Figure 6) and emphasize a number of features, such as:

- A dual/switchable view including the classic "visual programming" and a track-based, more "sequencer-oriented" display (see Figure 6). The *visual programming* mode (a) highlights functional relations between objects and processes, while the *tracks* mode (b) hides them to emphasize the temporal structure and authoring in a closer way to traditional DAW interfaces.
- Dynamic aspects in execution, through a number of features including on-the-fly computation or construction of the elements laid-out in the time structure (see Section 4).
- Representation and access to the time-structure of internal musical objects, in order to apply local or global time-transformations (shift, stretching, compression, etc.) and synchronization [8].

4. INTERACTIVE PROCESSES

Reactivity and interaction. O7 provides a native and improved support for *reactive visual programs*, an extension of the execution model of OM proposed in [11]. While OM visual programs are evaluated *on-demand* by explicit request of the user in order to update box values, this feature makes for *reactive* outputs of visual program components to propagate notifications of change to downstream-connected components (e.g. after some data modification by the user, change of an input value, reception of an external message, etc. — see Figure 5). This notification ultimately triggers an update request for specific values, which automatically executes the corresponding part of the visual program [12].

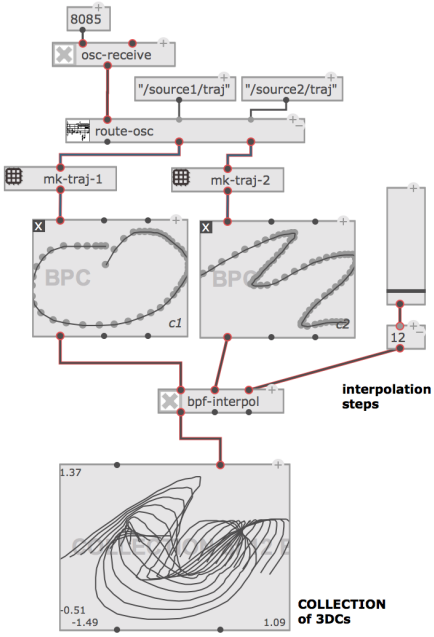


Figure 5. A reactive visual program. Red-highlighted patch cords and input/outputs are *reactive*: they propagate update notifications downwards in the patch, leading to the evaluation and update of the corresponding boxes.

On the example of Figure 5, reactive computations update a set of interpolated curves. They can be triggered upon reception of an OSC message [13] containing some curve description data via the *osc-receive* box, or upon activation by the user of the vertical slider component at the right, which determines the number of interpolation steps.

This new and optional mode of execution somehow reminds of interactive/data-driven multimedia environments such as Max [14]; however, visual programs are still "demand-driven" (only notifications flow downstream in the functional graph) and the overall computation model is preserved: no such thing as periodic audio callbacks or "hard" real-time reactions to events is intended — this is generally not compatible, nor desired in compositional processes and computer-aided composition software.

Communication, control and transfer of data with external systems is enabled through the easy instantiation of active sender/receiver threads. A native OSC support based on CNMAT's *odot* library [15] facilitates the handling of OSC-formatted data and the development of advanced cooperative scenarios with external multimedia frameworks [16].

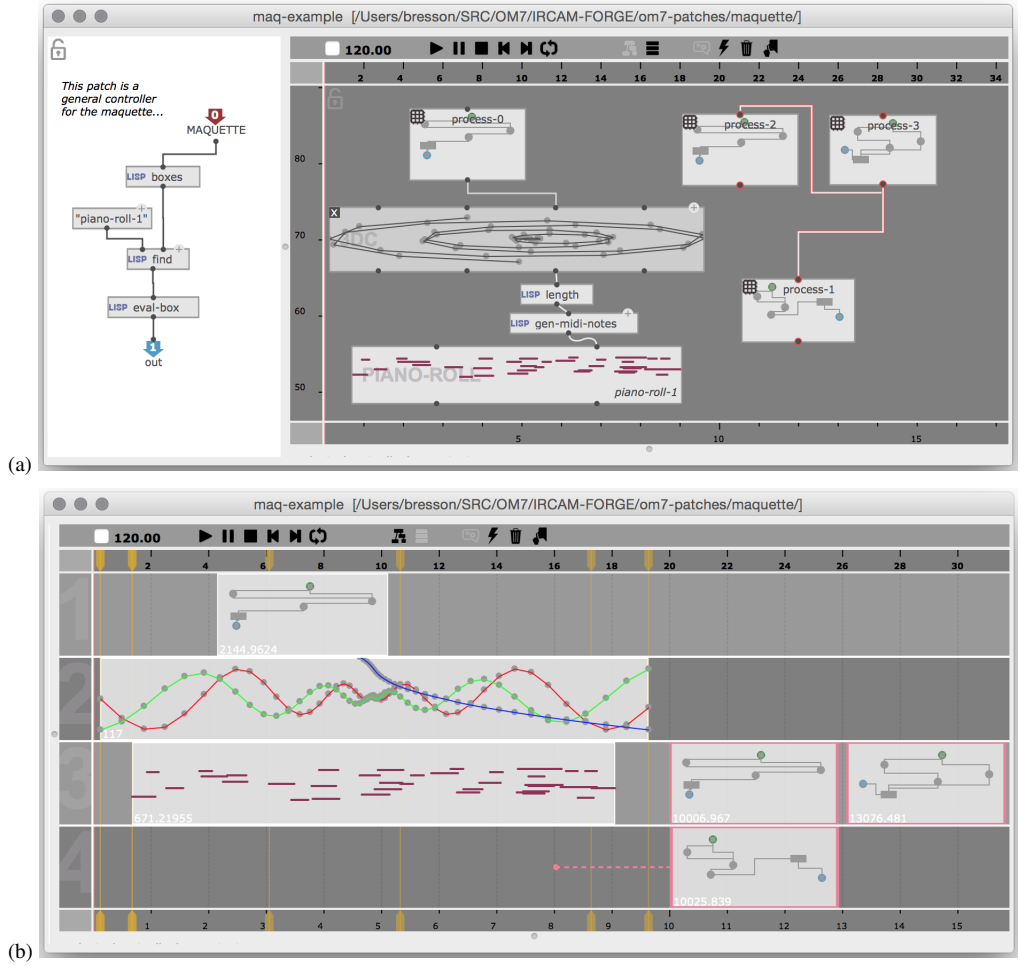


Figure 6. The *meta-score*: a new implementation of the OpenMusic *maquette*. (a) Classic "visual programming" mode. (b) Tracks visualization mode. In both visualization modes, red-framed (*reactive*) boxes can be executed on-the-fly at rendering time. On the left in (a), a control patch hosts the programming of preliminary or runtime actions to execute with the global score structure. Temporal anchors and markers lifted on the top and bottom rulers in (b) are used to move, stretch and synchronize the objects.

Dynamic scheduling. The environment integrates a new scheduling kernel managing the tasks of computation and musical execution (rendering/playback) in the computer-aided composition framework. A multi-threaded engine running at the core of the system concurrently handles in a same execution context the computation of visual programs, the dispatching of actions corresponding to musical objects being rendered (e.g. calls to the audio system or messages sent via MIDI or OSC), as well as the reactive scheduling of user-defined actions and program computations [10]. Musical objects can therefore encapsulate actions or programs producing other musical structures, integrated on-the-fly on the rendering time flow. As a result, dynamic structures such as the ones studied in the previous sections (e.g. *meta-scores* containing both objects and programs, or musical objects containing actions triggering arbitrary computations) can seamlessly execute. [4]

On Figure 6, for instance, the box at the top and the three boxes at the end of the sequence are processes (embedded visual programs). Each process computation might turn its container box value into a new object, integrated on-the-fly in the execution flow. It can also modify the future of the sequence by generating new objects or processes, scheduled at a later time. Depending on user's settings for each box, these processes can be executed either:

- before the rendering of the sequence starts;
- as part of the rendering (i.e. at the time determined by their position in the sequencer);
- as part of the rendering, with some anticipation (as depicts for instance the dashed red segment at the left of the process box in track #4).

5. DIGITAL SIGNAL PROCESSING

Another important aspect of the environment is its enhanced capacity to manage and process audio resources. Sounds are handled as memory-allocated audio buffers which can be freely transferred and processed in the visual programs — a difference with the current OM versions, which essentially deal with sound files on disk and process them via external calls to command-line tools. Audio rendering is also controlled by the scheduling engine, which has the possibility to dynamically address the audio output, synchronously or asynchronously, with a flexible and adaptive anticipation (from a few milliseconds to seconds to several minutes ahead). The different features described in this paper (reactive programming, dynamic objects manipulation,

[4] Here again we differentiate this approach from the one in real-time systems, since computation is not meant to take place instantaneously, and can impact the mid- or long-term future of executions.

scheduling and rendering, ...) are therefore connected to this audio thread, extended and applied to work with audio resources and processing.

In addition to existing DSP libraries (OM-SUPERVP, OM-PM2...) that have been ported to the new environment, dynamic connections have been established with several audio frameworks and libraries.

DSP features were developed for instance through connections with graphical controllers and DSP tools provided by the IRCAM Spat library [17] for panning, spatial audio rendering, and other audio processors like filters, reverbs, etc. Audio processors are controlled using OSC-formatted data structures (*bundles*) and associated to graphical interfaces capable of producing this data. The corresponding DSP objects in O7 are TIMED-SEQUENCES, internally considered as sequences of such OSC control bundles and linked to the audio processing and graphical controller components. They allow to drive the signal processing "in-time" via periodic calls and updates to these external components, either in an offline context (generating sound files and buffers), or as part of a rendering process (typically for real-time audio playback). SPAT-SCENE [18] is one of these DSP objects, providing a mixed offline/dynamic control over spatial sound scenes representation and synthesis (see Figure 7).

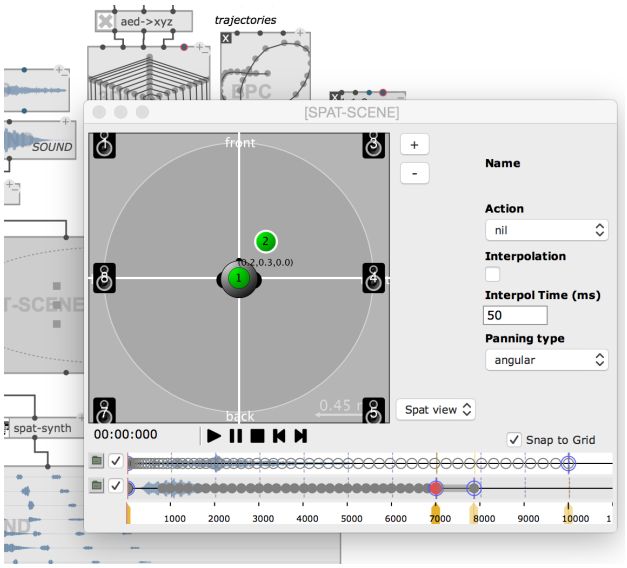


Figure 7. SPAT-SCENE: interface to *Spat* controllers and audio processing.

Figure 8 is another example of DSP using the IAE library [19] for content- or descriptor-based concatenative and granular synthesis [20]. This audio engine is parametrized by a set of sound sources, which are analyzed at initialization of the IAE box. The analysis produces a set of audio descriptors via the *IrcamDescriptors* library [21]. The IAE object can then generate sound grains upon requests of descriptor value(s) and/or time regions (see *iae-synth-desc*, producing a sound buffer on Figure 8). It also acts as a container for larger sequences of such requests (as a sub-class of DATA-STREAM — see the editor window in the lower part of the figure). Requests for sound grains can therefore be performed offline in a deferred-time set-up, or in (soft) real-time, integrating results in the audio playback at execution time — an example of asynchronous connection of the scheduling engine to the audio output thread.

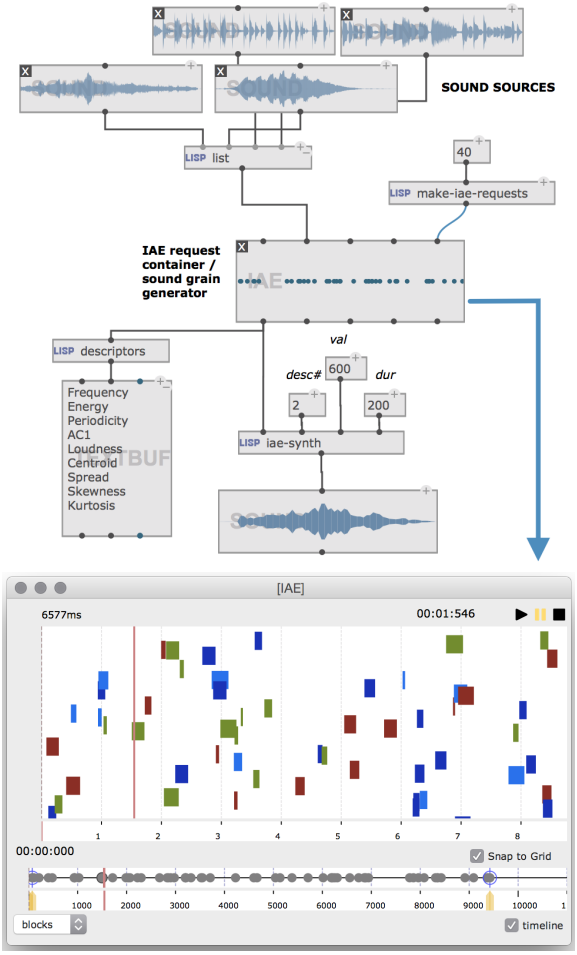


Figure 8. Versatile use of the IAE audio engine for descriptor-based granular synthesis: sound grain generator and container of contents-based granular synthesis requests.

6. CONCLUSIONS

We presented a new implementation of the OpenMusic visual programming language and computer-aided composition environment. The main features we have highlighted in this implementation are:

- An improved and redesigned visual programming framework;
- New modular editors for musical objects, focusing on time structuring and representation;
- A powerful, dynamic computation and scheduling kernel handling musical rendering and program executions;
- A new sequencing environment, merging the main features of the OM *maquette* with dynamic/real-time interactions and DAW-oriented visualization;
- Flexible and extensible audio framework including connections to external DSP libraries.

We believe these features have the potential to set this project as a landmark in the evolution of computer-aided composition software. Some of them have been used and proved relevant through preliminary experiments and applications, for instance in guided improvisation systems or for the structured/interactive control of sound synthesis processes [10]. A detailed example of compositional application is presented in [22].

Although focusing significant parts of our research on interactive aspects of compositional processes, we also wish to emphasize how this approach differentiates itself from real-time computer music systems. Indeed we position it as complementary to recent "real-time computer-aided composition" projects such as the *bach* and *cage* libraries in Max [23], where the real-time paradigm structures musical processes extended to the symbolic domain of computer-aided composition. If the performance of current computer systems tends to assimilate the perceptions of deferred-time and real-time — at least from the user's point of view —, the "deferred-time" approach of computer-aided composition tools, where computation of musical structures is not enslaved to their own execution time, remains for us a fundamental characteristics in the development of formal processes related to music writing.

Finally, the O7 implementation currently includes a subset only of existing OM features and functionalities. For instance, scores or traditional music notation, which are among the most crucial aspects of common computer-aided composition systems and applications, are currently not present in the new environment. For this reason we do not yet consider it a valid/full-featured successor for OpenMusic: [5] in its present state, this software addresses specific applications, and suits specific needs of computer-aided composition users. But work is still in going on: notation in particular (traditional and "extended" music notations) will be one of our main perspective for future developments.

7. REFERENCES

[1] G. Assayag, "Computer Assisted Composition Today," in *1st symposium on music and computers*, Corfu, 1998.

[2] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic," *Computer Music Journal*, vol. 23, no. 3, 1999.

[3] M. Laurson and J. Duthen, "Patchwork, a Graphic Language in PreForm," in *International Computer Music Conference (ICMC'89)*, Columbus, USA, 1989.

[4] J. Bresson, C. Agon, and G. Assayag, "Visual Lisp/CLOS Programming in OpenMusic," *Higher-Order and Symbolic Computation*, vol. 22, no. 1, 2009.

[5] C. Agon, "OpenMusic : Un langage visuel pour la composition musicale assistée par ordinateur," Ph.D. dissertation, Université Pierre et Marie Curie (Paris 6), Paris, France, 1998.

[6] J. Bresson, C. Agon, and G. Assayag, Eds., *The OM Composer's Book (3 volumes)*. Editions Delatour / Ircam-Centre Pompidou, 2006/2008/2016.

[7] J. Bresson, D. Bouche, J. Garcia, T. Carpentier, F. Jacquemard, J. Maccallum, and D. Schwarz, "Projet EFFICACE : Développements et perspectives en composition assistée par ordinateur," in *Actes des Journées d'Informatique Musicale*, Montréal, Canada, 2015.

[8] J. Garcia, D. Bouche, and J. Bresson, "Timed Sequences: A Framework for Computer-Aided Composition with Temporal Structures," in *Proceedings of the International Conference on Technologies for Music Notation and Representation*, A Coruña, Spain, 2017.

[9] J. Bresson and C. Agon, "Temporal Control over Sound Synthesis Processes," in *Proceedings of Sound and Music Computing (SMC'06)*, Marseille, France, 2006.

[10] D. Bouche, J. Nika, A. Chechile, and J. Bresson, "Computer-aided Composition of Musical Processes," *Journal of New Music Research*, vol. 46, no. 1, 2017.

[11] J. Bresson and J.-L. Giavitto, "A Reactive Extension of the OpenMusic Visual Programming Language," *Journal of Visual Languages and Computing*, vol. 25, no. 4, 2014.

[12] J. Bresson, "Reactive Visual Programs for Computer-Aided Music Composition," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Melbourne, Australia, 2014.

[13] M. Wright, "Open Sound Control: an enabling technology for musical networking," *Organised Sound*, vol. 10, no. 3, 2005.

[14] M. Puckette, "Combining Event and Signal Processing in the MAX Graphical Programming Environment," *Computer Music Journal*, vol. 15, no. 3, 1991.

[15] J. MacCallum, R. Gottfried, I. Rostovtsev, J. Bresson, and A. Freed, "Dynamic Message-Oriented Middleware with Open Sound Control and Odot," in *Proceedings of the International Computer Music Conference (ICMC'15)*, Denton, USA, 2015.

[16] J. Bresson, J. MacCallum, and A. Freed, "o.OM: Structured-Functional Communication between Computer Music Systems using OSC and Odot," in *ACM SIGPLAN Workshop on Functional Art, Music, Modeling & Design (FARM) – ICFP'16*, Nara, Japan, 2016.

[17] T. Carpentier, M. Noisternig, and O. Warusfel, "Twenty Years of Ircam Spat: Looking Back, Looking Forward," in *Proceedings of the International Computer Music Conference (ICMC'15)*, Denton, USA, 2015.

[18] J. Garcia, T. Carpentier, and J. Bresson, "Interactive-Compositional Authoring of Sound Spatialization," *Journal of New Music Research*, vol. 46, no. 1, 2017.

[19] N. Schnell, D. Schwarz, R. Cahen, and V. Zappi, "IAE & IAEOU," in *Topophonie research project : Audiographic cluster navigation (2009-2012)*, ser. Les Carnets d'Experimentation de l'ENSCI, R. Cahen, Ed., 2012. [Online]. Available: https://topophonie.com

[20] D. Schwarz, "Corpus-Based Concatenative Synthesis," *IEEE Signal Processing Magazine*, vol. 24, no. 2, 2007.

[21] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the Cuidado project," IRCAM, Paris, France, Tech. Rep., 2004.

[22] S. Agger, J. Bresson, and T. Carpentier, "*Landschaften* – Visualization, Control and Processing of Sounds in 3D Spaces," in *Proceedings of the International Computer Music Conference (ICMC'17)*, 2017.

[23] A. Agostini and D. Ghisi, "A Max Library for Musical Notation and Computer-Aided Composition," *Computer Music Journal*, vol. 39, no. 2, 2015.

[5] OM6 is the official and stable version of the computer-aided composition environment, see `http://repmus.ircam.fr/openmusic/`